ANÀLISI DEL SISTEMA OPERATIU RTLINUX E IMPLEMENTACIÓ D'UN ENTORN DE DESENVOLUPAMENT DE TASQUES EN TEMPS REAL (APLICAT AL CONTROL DE PROCESOS)

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Marc Franco i Farré
i dirigit per
Pedro Balaguer Herrero
Bellaterra, 11 de Juny de 2007

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Pedro Balaguer Herrero

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat:

Bellaterra, 8 de Juny de 2007

Agraïments a la meva família,
a ella que l'he tingut al costat donant-me suport durant el projecte,
als amics que han fet possible que hagi arribat aquí
i al suport constant del Director del Projecte.

Gràcies a ells aquí arriba el final de l'inici
de la carrera de la meva vida.

Índex de contingut

1

Introducció

1. Introducció

Aquest document és la memòria del projecte final de carrera **"Desenvolupament de Software per a Sistemes Operatius en Temps Real lliures"** realitzat a la facultat d'enginyeria informàtica de la Universitat Autònoma de Barcelona (ETSE-UAB). Aquest projecte s'emmarca en el conjunt dels Sistemes Operatius en Temps Real de llicència gratuïta com és RTLinux.

Com a memòria, és un text imprescindible per comprendre el desenvolupament del projecte final de carrera i serveix per veure totes les etapes seguides per arribar a complir els objectius proposats.

Són cada cop més els dispositius que utilitzen les bases dels sistemes en temps real, per controlar sistemes per computador. Un clar exemple es la indústria del automòbil, un turisme actual de classe mitjana inclou pel voltant d'una dotzena d'aquests automatismes (ABS, aribags, etc.). Un altre camp d'aplicació dels sistemes en temps real són els electrodomèstics de nova generació que inclouen sistemes de control i temporització en temps real. És tant gran el creixement d'aquests sistemes en temps real, que avui dia ja dupliquen en número al sistemes convencionals informàtics. Els estudis relaten que aquesta diferència va en augment. La propietat que diferència els sistemes en temps real dels altres és que les seves accions no tant sols han de ser correctes sinó que s'han de produir en uns intervals de temps determinats pels sistemes que controlen. Per exemple, el control d'injecció d'un motor de combustible s'ha de realitzar la injecció de la barreja dins del interval de temps marcat per la rotació del motor, sinó es compleixen aquests intervals el motor no funcionaria. En aquest sistemes es tracta d'un sistema empotrat, els sistemes en temps real normalment són empotrats i tenen bastants restriccions de recursos. Tot i aquestes restriccions de recursos, les aplicacions per aquests sistemes en temps real poden arribar a ser molt grans, els sistemes de control d'un avió es conten les línies de codi de les aplicacions en milions. Aquesta complexitat comporta l'obligació d'utilitzar la enginyeria del Software.

En aquest projecte s'analitza la viabilitat de crear un d'aquests sistemes en temps real utilitzant software lliure (Sistema Operatiu i eines de desenvolupament). S'analitza la possibilitat de desenvolupar sistemes en temps real sense tenir la necessitat de recórrer a eines en les quals s'hagi de pagar llicencies.

El software lliure és una definició molt complexa la qual comporta diferents tipus de llicencies i que no s'entra en discussió en aquest projecte. Simplement remarcar la necessitat d'utilitzar aquest tipus de software per les següents causes.

- La primera causa i potser la més important en aquest punt de la situació personal és la obtenció de llicències de software comercial. Aquests tipus de llicencies poden arribar a ser d'uns preus tant alts que una persona individual no pot pagar per un software el qual es vol analitzar per comprovar-ne la viabilitat de la construcció de software.

- Una altre causa i no menys important que l'anterior és que amb l'obtenció d'eines i software de codi lliure és que es podrà modificar i personalitzar de la manera que es vulgui, per contra, el software comercial és únic i no es pot modificar. Per tant, els software lliure ens dona més flexibilitat per personalitzar-lo a la nostre manera.

> Un enginyer ha de poder optimitzar el software lliure obtingut per les pròpies necessitats e inclòs millorar-lo i exposar-ne les idees. Aquesta és potser una de les grans avantatges del software lliure, la creació de comunitats les quals discuteixen idees d'implementació, d'aquesta manera és més fàcil obtenir software de qualitat.

Personalment no he estat mai una persona que hagi seguit de molt a prop tot aquest entorn de software lliure ni hem considero un usuari expert en Sistemes Operatius com Linux, és més, en el capítol motivacions expresso el repte personal que m'imposo en aquest projecte final de carrera al haver de tractar tot amb un entorn Linux i de software lliure.
Després de tenir clar que són els sistemes en temps real i les avantatges que donen, es fa un anàlisis dels Sistemes Operatius en Temps Real per crear l'entorn de creació de sistemes en Temps Real. L'escollit és RTLinux, un Sistema Operatiu creat per Michael Baravanov i Victor Yodaiken que avui dia continuen el seu desenvolupament en la seva empresa FSMLabs amb una versió comercial, RTLinux/Pro. S'estudien les propietats d'aquest Sistema Operatiu, el funcionament i s'analitza la viabilitat d'utilitzar-lo com entorn per crear tasques en temps real.
Seria de gran importància crear un entorn potent, flexible i sense pagar per llicencies comercials i això s'aconsegueix amb RTLinux. Al final es veu com es pot desenvolupar software per a sistemes en temps real de forma amigable per l'usuari sense que aquest hagi de tenir coneixements profunds de programació en RTLinux.
Aquest projecte és doncs, un estudi inicial de la creació d'aplicacions per tasques en temps real en un entorn de temps real com es RTLinux. A partir d'aquí es senten les bases per a la creació d'un entorn de creació de tasques en temps real complert i de fàcil ús. Aquest projecte no intenta crear un software de última generació ni molt menys. El projecte el que pretén es veure la possibilitat de crear un entorn de temps real i desenvolupar-ne aplicacions. Un cop el lector a llegit el document i està interessat en crear un entorn de creació de tasques en temps real pot seguir el projecte on es deixa aquest. Al final del projecte es proposen millores del que s'ha fet i idees de disseny d'aplicacions per un possible treball futur.

1.1 Motivacions

Després de cursar tots aquests any per complir totes les assignatures del pla d'estudis que s'estableixen com a necessàries per acabar la carrera. Com alumne i futur enginyer, queda fer una última pinzellada als estudis amb aquest projecte i aquest document formal. Durant tots aquests anys s'han après molts conceptes de diferents camps però sobretot ens han ensenyat a aprendre. Un enginyer ha de ser capaç de desenvolupar-se en qualsevol camp en el mínim termini de temps possible per que això es el que es demana quan se surt de la universitat. Per tant els conceptes assimilats tenen una importància, però el gran pes es la capacitat que s'obté per aprendre i el projecte final de carrera intenta reflexa aquesta característica obtinguda durant els anys d'estudiant.
El tema del projecte té diferents motivacions. Una de les més importants i personal és la motivació del repte. Mai he estat una persona que hagi seguit en profunditat l'entorn de Linux, és més, sempre he

utilitzat Windows per la seva comoditat. Ara no entraré a discussions de que es millor i que es pitjor, simplement cadascun serà funcional per cada entorn. Vaig començar a fer el projecte amb uns coneixement mínims de ús de Linux, i he aconseguit compilar un kernel e instal·lar un linux en temps real.

Una motivació es aquest terme, "sistemes en temps real", no en tenia gaire coneixement tampoc i m'atreia molt aquest concepte, saber que es el temps real, perquè és necessari. Al transcurs del projecte cada cop han estat més les motivacions, com més coses noves s'aprenien més en profunditat volia arribar. Un tema prou atractiu i interessant de desenvolupar, unir conceptes de temps real amb software de codi lliure, tot un repte. Sistemes de temps real utilitzats en entorns tant avançats com en les naus enviades de la NASA com la "pathfinder" que portava el sistema operatiu en temps real MarteOS o en entorns tan crítics com aparells mèdics.

Una altre motivació ha estat fer el projecte per acabar la carrera, posar en marxa tots les tècniques d'estudi, anàlisis i aprenentatge que he adquirit durant els anys d'estudiant, de quina millor manera acabar una carrera fent un projecte final i un cop acabat fer una reflexió personal de tot el que s'ha après i que es reflexa en el projecte.

Finalment anomenar un problema de temps que m'he trobat aquest any en l'elaboració del projecte final de carrera i que m'ha obligat a seguir un estricte calendari per poder obtenir els resultats finals en el temps estimat, aquest aspecte m'ha ajudat aprendre a posar en marxa un projecte planificant el temps amb altres projectes personals de manera real.

1.2 Objectius

Són varis els objectius presentats en el projecte amb els quals s'intenta arribar a un objectiu final, concloure en la viabilitat de crear un entorn de desenvolupament de tasques en temps real de forma fàcil tot utilitzant software de codi lliure. Per arribar aquest objectiu final s'ha d'estudiar en profunditat els sistemes en temps real, veure els sistemes operatius que implementen aquests sistemes en temps real, estudiar-ne i analitzar-ne els sistemes operatius que són de software lliure per crear l'entorn desitjat. Un cop creat l'entorn desitjat veure quines possibilitats de programació hi ha actualment per desenvolupar aplicacions per ampliar el nombre d'eines per facilitar al usuari final la creació de tasques en temps real i el posterior anàlisis. Els objectius del projecte s'han anat ampliant i guiant durant el desenvolupament d'aquest.

Al final d'aquest projecte s'han proposat nous objectius els quals estan referenciats en el capítol de conclusions en l'apartat de treball futur els quals poden arribar a ser nous projectes finals de carrera per la seva gran envergadura. S'havia iniciat el projecte amb una visió bastant limitada de les possibilitats que oferien aquests tipus de sistemes.

Com ja s'ha comentat, la idea del projecte era un estudi inicial dels sistemes en temps real i tot el que hi ha darrera. Ara un cop acabat el projecte i s'han complert els objectius, la visió ha augmentat i ha sorgit idees per a futurs treballs.

1.3 Planificació del temps

En aquesta planificació no esta inclosa la definició del projecte degut a que aquesta etapa es preliminar al projecte i tampoc es té en compte el disseny i preparació de la presentació del projecte. A continuació es detallen els punts de la taula per tenir més clar en que consisteix cada concepte.

1.- Estudi inicial dels Sistemes en Temps Real

En aquest primer punt es fa un estudi previ de que són els sistemes en temps real, per a que serveixen, quins camps d'aplicacions tenen. També es fa un estudi més profund de les característiques d'aquests tipus de sistemes, com funcionen (per exemple: tipus de planificació), etc.

2.- Anàlisis dels diferents Sistemes Operatius en Temps Real

Seguidament es passa a veure els Sistemes Operatius en Temps Real emmarcats dins dels Sistemes Operatius. Es fa un estudi previ del funcionament bàsic dels Sistemes Operatius per comprendre les avantatges que suposaran els Sistemes Operatius en Temps Real. S'analitzen un conjunt de Sistemes Operatius en Temps Real actuals ja siguin de software lliure o comercials. Es fa un posterior anàlisis per elegir-ne un pel projecte.

3.- Instal·lació i Configuració de RTLinux

En el anàlisis anterior s'elegeix el Sistema Operatiu RTLinux. En aquest punt es passa a veure com compilar e instal·lar RTLinux. Un cop funcionant s'ha de configurar per el nostre ús. Aquest ha estat un dels temes més feixucs potser per la falta d'experiència en entorns Linux. S'ha arribat a perdre molts cops el Sistema Operatiu.

4.- Aprenentatge: ús i programació en RTLinux

Un cop RTLinux ha funcionat correctament s'ha passat aprendre a utilitzar-lo (mòduls carregables, planificadors, fifos, etc..) i programar amb la API que es disposa de RTLinux per programar tasques en temps real.

5.- Anàlisis RTLinux com a Sistema Operatiu en Temps Real

En aquest punt ja es té uns mínims coneixements de funcionament i programació amb RTLinux i cal veure el rendiment que ofereix RTLinux a través de programes per testar la planificació, les fifos, les interrupcions, el jitter, la latencia, etc. del Sistema Operatiu.

6.- Aprenentatge programació Tcl/Tk

Per desenvolupar les aplicacions per facilitar la tasca de l'usuari final en un entorn de temps real com RTLinux s'elegeix el llenguatge de programació Tcl/Tk. En aquest punt s'analitza el perquè de l'elecció d'aquesta eina enfront d'altres i el posterior aprenentatge per assolir uns mínims per estar preparat pel desenvolupament d'aplicacions.

7.- Desenvolupament d'aplicacions

Disseny e implementació de les diferents eines desenvolupades durant el projecte. Eines com l'aplicació que mostra l'execució de les tasques en el planificador de temps real, el Generador de Tasques (la més complerta), les diferents aplicacions de generació de gràfiques dels resultats.

8.- Proves i Conclusions

Proves referents al Sistema Operatiu en Temps Real RTLinux i les aplicacions creades. Generació de tasques amb l'aplicació destinada per aquest efecte. Anàlisis dels resultats per extreure'n les conclusions finals del projecte.

9.- Documentació

Redacció i revisió de la present documentació. Expressió sobre el paper del treball fet i els objectius assolits Una de les parts més complexes del projecte.

Concepte	Quantitat
1. Estudi inicial dels Sistemes en Temps Real.	25h
2. Anàlisis dels diferents Sistemes Operatius en Temps Real.	25h
3. Instal·lació i Configuració de RTLinux	40h
4. Aprenentatge: ús i programació en RTLinux	50h
5. Anàlisis RTLinux com a Sistema Operatiu en Temps Real	20h
6. Aprenentatge programació Tcl/TK	20h
7. Desenvolupament d'aplicacions	80h
8. Proves i Conclusions	50h
9. Documentació	150h
Total	**460h**

1.4 Estructura del Document

El document esta dividit en els següents capítols i annexes que s'expliquen a continuació:

Capítol 1 Introducció

Aquest capítol es l'actual i tracta d'introduir al lector al projecte final de carrera. Es descriuen les motivacions per les quals s'ha fet el projecte, els objectius proposats i una planificació temporal de les tasques a realitzar.

Capítol 2 Sistemes en Temps Real

El capítol dos exposa els coneixements adquirits sobre sistemes en temps real per la posterior preparació d'un entorn de treball en temps real. Aquest tema es purament teòric i introdueix al lector en els conceptes de temps real.

Capítol 3 Preparació d'un Sistema en Temps Real

Aquest capítol exposa els passos a seguir i els coneixements a obtenir per preparar les eines necessàries per obtenir l'entorn de treball en temps real i el perquè de la elecció d'aquestes eines. S'explica com instal·lar el Sistema Operatiu en Temps Real RTLinux i com configurar-lo per personalitzar-lo. Es mostra el funcionament d'aquest sistema operatiu i la programació de la API en temps real. Per altre banda s'explica com instal·lar Tcl/Tk i la manera de programar en aquest tipus de llenguatge.

Capítol 4 Generador de Tasques en Temps Real

Es posa en comú tots els coneixements assolits i es fa ús de totes les eines preparades per desenvolupar una aplicació per generar tasques en temps real de forma fàcil i amigable per l'usuari final. S'explica els detalls de disseny e implementació de l'aplicació creada. Finalment a través d'un exemple de creació de tasques en temps real, s'explica el funcionament de l'eina implementada.

Capítol 5 Exemples i Resultats

Es creen una sèrie d'exemples ja sigui utilitzant l'aplicació creada o manualment per veure la facilitat de crear tasques amb l'aplicació i veure també aspectes del temps real. Es mostren i comenten els resultats obtinguts. Concretament s'exposen cinc exemples que intenten avarca tot el que s'après durant el projecte.

Capítol 6 Conclusions

S'exposen les conclusions extretes dels resultats obtinguts. Es comprova quins objectius s'han assolit. S'exposen millores a implementar e idees que s'han originat durant el transcurs del projecte per a treballs futurs. Finalment es fa una reflexió personal sobre el projecte final de carrera.

Annexes

En aquest punt s'adjunten els codi font de les aplicacions creades i dels exemples generats.

2

Sistemes en Temps Real

---

2. Sistemes de Temps Real

2.1 Introducció

En aquest punt, s'aclareix alguns conceptes previs necessaris per comprendre el que s'aconsegueix amb el projecte, construir un sistema en temps real en un PC de sobretaula amb software de llicencia gratuïta i veure la viabilitat de desenvolupar software en temps real per millorar la interacció de l'usuari amb el sistema.

Es reflecteix tot el que involucra crear un sistema de temps real, es veu que és un sistema de temps real juntament amb les seves característiques, definicions, propietats, exemples, etc. Després veurem que són els Sistemes Operatius i perquè serveixen. Veurem unes classificacions d'aquests en la que trobarem dins d'aquesta els Sistemes Operatius en Temps Real que serà el tipus de Sistema Operatiu que utilitzarem per crear l'entorn de Sistema de Temps Real. Igual que amb els Sistemes Operatius, amb els Sistemes Operatius en Temps Real en veurem característiques, propietats, etc. Un cop es té clar aquests conceptes, es passa a veure unes quantes distribucions actuals de RTOS (Real Time Operating System) tan amb llicencia gratuïta com RTOS d'àmbit comercial.

De tots els Sistemes Operatius en Temps Real que veurem, donarem més importància a RTLinux, que serà el Sistema Operatiu utilitzat en el projecte per crear el nostre Sistema de Temps Real. En veurem una explicació amb més detall de com està estructurat, les característiques i finalment algun exemple d'aplicació. En pròxims capítols, veurem en més detall el funcionament de RTLinux i com programar tasques en temps real amb tot el que va associat.

Tenim diferents tipus de sistemes informàtics, els sistemes basats en el processament per lots, basats en el processament en línia i els de temps real. El primer sistema, el basat en lots, no té importància el punt en el temps en el que s'aconsegueix el resultat, per exemple la facturació d'una empresa. El segon sistema, el basat en processament en línia és desitjable saber en quin punt en el temps s'aconsegueixen els resultats, per exemple els sistemes de compra on-line. Per últim, el que interessa, els sistemes de temps real és necessari saber en quin punt del temps s'obtindran els resultats, han de ser sistemes deterministes, per exemple els semàfors, control aeri, etc..

Els sistemes de temps real són sistemes informàtics que interactuen amb el medi de forma determinística i sempre de forma molt controlada. Aquesta propietat fa que un sistema de temps real sigui òptim per a funcions de supervisió i control, són típiques les tasques d'adquisició de dades o supervisió. Aquestes tasques necessiten tenir uns intervals ben definits i normalment són tasques periòdiques que és en el que es basen principalment els sistemes de temps real.

2.2 Definicions

En el transcurs del temps, diferents científics han definit els sistemes de temps real a la seva manera, seguidament podem veure alguns exemples:

· Laplante:

"Un sistema de temps real és un sistema en que els temps de resposta han de satisfer requisits explícits, i en cas de no respondre dins d'un interval acordat es poden produir conseqüències greus, fins i tot poder arribar al fracàs."

· Burns:

"Un sistema de temps real és un sistema de processament de la informació que ha de respondre a estímuls generats exteriorment dins un període finit i especificat."

· Donald Gillies:

"Un sistema de temps real es aquell que per que les operacions computacionals siguin correctes no depèn tant sols del resultat obtingut, sinó en el temps en que s'obtindrà el resultat. Si el temps en que s'obté el resultat no acompleix els requisits, es dirà que el sistema ha fallat."

Un bon exemple de sistema en temps real es el d'un Robot que ha de recollir peces d'una cinta mecànica. Si el robot arriba tard, la peça ja no estarà. Encara que el procés d'arribar al lloc es correcte, no s'acompleix el requisit temporal i per tant s'ha fallat en l'acció. Si la peça arriba abans també es dirà que el sistema ha fallat ja que el braç del robot pot impedir que la peça finalitzi.
Es freqüent veure com es confon temps real amb sistemes ràpids. Cal doncs deixar clar abans de seguir endavant, que sistema de temps real no es sinònim de sistema ràpid.
La importància d'un sistema en temps real no és la rapidesa en fer una tasca, la importància del temps real és assegurar que el temps definit per fer una tasca es complirà.

2.3 Característiques

Per tenir una idea més clara de com són els sistemes en temps real no n'hi ha prou amb les definicions anteriorment citades. S'ha de profunditzar una mica més analitzant les característiques que acompleixen aquests tipus de sistemes.

· **Determinisme:** El determinisme es clau en els sistemes de temps real. El determinisme és saber quan tardarà una tasca en iniciar-se. Aquest característica es molt important ja que un sistema de temps real ha de tenir una precisió perfecte. Normalment es defineix com al temps abans de respondre a una interrupció. És important saber el temps de resposta a un event extern.

· **Responsivitat:** La responsivitat és el temps que tarda una tasca en executar-se després d'atendre la interrupció.

· **Control:** En sistemes de temps real els usuaris tenen un control més ampli d'aquest. Els processos seran configurats per l'usuari. L'usuari podrà especificar les prioritats, la gestió de la memòria, etc..

· **Confiabilitat:** El sistemes de temps real a part de no tenir errors, ha de ser un sistema estable. Ha de mantenir un marge d'error que no porti a un punt crític. Si el sistema falla, aquest a de mantenir el màxim de dades i si no pot complir totes les tasques, complirà les més crítiques.

Les aplicacions dels sistemes real són molt variades, i contínuament apareixen nous camps d'utilització per aquests. Alguns d'aquests es troben en sectors com les telecomunicacions, sistemes multimedia, robòtica, aviació, automòbils, etc. Hi ha alguns sistemes de temps real que a part de requeriments temporals també tenen requeriments de seguretat crítics.

Quan anem a dissenyar un sistema de temps real passarem per varies fases:

(i) S'identifiques les tasques que s'hauran de realitzar i les restriccions temporals d'aquestes.
(ii) Seguidament es codificaran els programes que executaran les tasques.
(iii) Finalment es mesuraran els temps per veure si s'acompleixen els requisits temporals.

Més endavant es veu com es segueixen aquests tres punts i com és fa un estudi exhaustiu del requeriment temporal i la planificació de les tasques. Tot això és fa mitjançant simulacions i diagrames de temps.

2.4 Classificació dels sistemes en temps real

El sistemes en temps real es poden classificar de diferents maneres. Seguidament es veuen les diferents característiques utilitzades per a classificar-los. És tindrà en compte diferents aspectes, el temps, el tipus de processament, la duresa en el tractament dels errors, etc.

1) La primera classificació és farà depenent de la duresa en el tractament d'errors que es presentin en els sistemes:

· Soft real-time systems: Aquests tipus de sistemes poden tolerar un incompliment en el temps de resposta, amb la seva penalització pertinent, es a dir, es poden perdre plaços a vegades. En aquests sistemes s'assegura que les tasques més crítiques s'executaran. Les dades s'executen en memòries no volàtils. Un exemple de sistema de temps real soft és una tasca que fa l'adquisició de dades d'un determinat sistema.

· Hard real-time systems: En aquests sistemes, una resposta fora del plaç no té valor i resultarà la falla del sistema. Un exemple és el control de frenada d'un automòbil. Si aquesta tasca no s'acompleix dins un plaç especificat els resultats seran desastrosos.

· Firm real-time systems: En aquests sistemes es poden perdre plaços ocasionalment, és té però que una resposta fora de plaç no tindrà cap valor. Aquest tipus de sistemes són per exemple sistemes en temps real multimedia.

2) També es poden classificar segons l'escala de temps utilitzada. Es diferencien tres tipus sistemes en temps real:

· Basats en el rellotge: S'invoca i executa repetidament a intervals constants a partir d'una invocació inicial, tasques periòdiques.

· Basats en events: Les accions es realitzen a partir d'un event, per exemple, una acció comença quan es llegeix una dada del sensor, bàsicament tasques aperiòdiques.

· Interactius: L'execució de les accions es fa en temps irregulars, per exemple, un operari inserint dades.

3) Es poden classificar per la manera de processar les dades:

· Sistemes centralitzats: Són sistemes en que tant sols hi ha un node pel processament, es l'encarregat d'atendre totes les peticions, les tasques que s'executen ho fan en memòria compartida. S'utilitza poc temps en la comunicació entre tasques.

· Sistemes distribuïts: Són sistemes en els que hi ha varis nodes, units a través d'una xarxa que els comunica, les tasques s'executen en els diferents nodes. Hi ha un important consum en temps per la comunicació entre tasques.

4) Una altra manera de classificar els sistemes en temps real serà segons l'estratègia de planificació que s'utilitza :

· Sistemes estàtics: En aquests tipus de sistemes, les característiques i la seva naturalesa son coneguts abans, en temps de disseny es planificarà la seva execució. El sistema no admet noves tasques en el moment de l'execució. Així que tot el pla d'execució es previst abans de començar a executar les tasques.

· Sistemes dinàmics: Aquests tipus de sistemes admeten tasques noves en l'execució. Quan detecta una nova tasca, el sistema analitzarà si pot executar-la complint els requisits temporals

sense afectar les altres tasques ja previstes en el disseny. Aquests tipus de sistemes tenen un planificador complex.

5) Per últim, es classifiquen els sistemes en temps real segons la manera com interactuen amb el medi:

· Sistemes embeguts: Són sub-sistemes d'un sistema més complex. Un exemple de sistema embegut seria el sistema de control d'injecció de combustible d'un automòbil.

· Sistemes no embeguts: Són sistemes únics e independents del hardware al qual corren.

2.5 Tasques de temps real

Les activitats que es realitzen en un sistema de temps real s'anomenen tasques.
Les tasques en temps real tenen diferents propietats que les defineixen.

- Funcionals: Que fan les tasques.
- Temporals: Quan es realitzen.
- Altres: fiabilitat, seguretat...

Les tasques de temps real tenen un comportament temporal específic el qual s'especifica mitjançant els seus atributs temporals. Aquests atributs temporals són l'esquema d'activació i el plaç per executar-se la tasca.

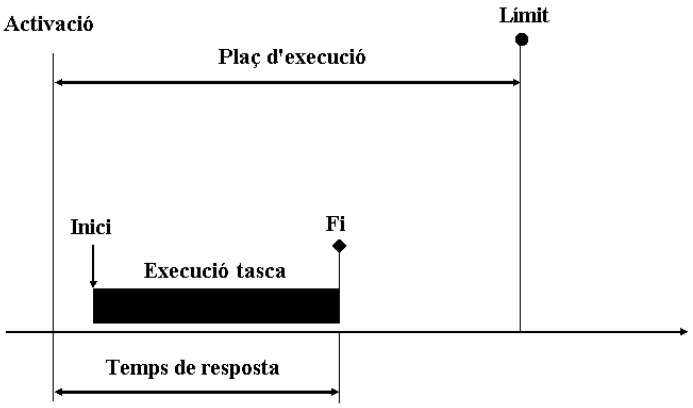


Figura 2.1: *Exemple execució tasca en temps real*

La figura 2.1 il·lustra l'execució d'una tasca en temps real. Es veu a la part superior com hi ha un rang on es limita el plaç d'execució.

La tasca s'ha d'executar dins d'aquest plaç. També es pot diferenciar bé entre el temps d'execució i el temps de resposta d'una tasca en temps real. El temps d'execució és el temps de còmput, el temps que tarda la tasca en executar-se íntegrament, en canvi, el temps de resposta d'una tasca en temps real compren el temps d'execució de la tasca i el temps que passa des de que s'activa la tasca fins que es comença a executar.

Anteriorment es parlava dels atributs temporals de les tasques els quals donen el comportament temporal de cada tasca. Aquests atributs temporals són:

· Activació: Tenim dos tipus d'activació d'una tasca, les periòdiques i les aperiòdiques. Les tasques periòdiques s'activen a intervals regulars, amb període T. Les tasques aperiòdiques s'activen cada cop que succeeix un event.

· Plaç de resposta: Hi ha dos tipus de plaços de resposta. Quan hi ha un temps límit per acabar, aleshores es parla de plaç absolut. Quan el plaç té un interval des de l'activació es parla de plaç relatiu.

Per a produir-se una tasca es necessari que hi hagi un event, periòdic de rellotge o un event extern. Quan es produeix un event entren en joc el període de jitter i la latència.

· Latència en un event: La latència davant d'un event es el temps que està des de que es produeix la interrupció fins que es comença a executar la rutina de tractament de la interrupció. Un event pot ser una interrupció hardware o software. La latència davant una interrupció hardware és el temps des de que es produeix la interrupció fins que s'executa la primera instrucció de la rutina de tractament, degut a retards en el bus. En canvi, la latència davant una interrupció software és el temps des de que la senyal es generada fins que la primera instrucció de las tasca es executada, en aquest tipus el retard és degut al temps d'accés als registres del processador.

· Període de jitter: El període de jitter són les variacions en el temps entre la mateixa tasca quan s'executa de forma repetida. Una altre manera de dir-ho, el jitter és la diferència entre el temps esperat per que un event passi i el temps en que passa realment l'event.

En la següent figura és mostra gràficament quan actua la latència d'un event i el període de jitter comentats anteriorment.

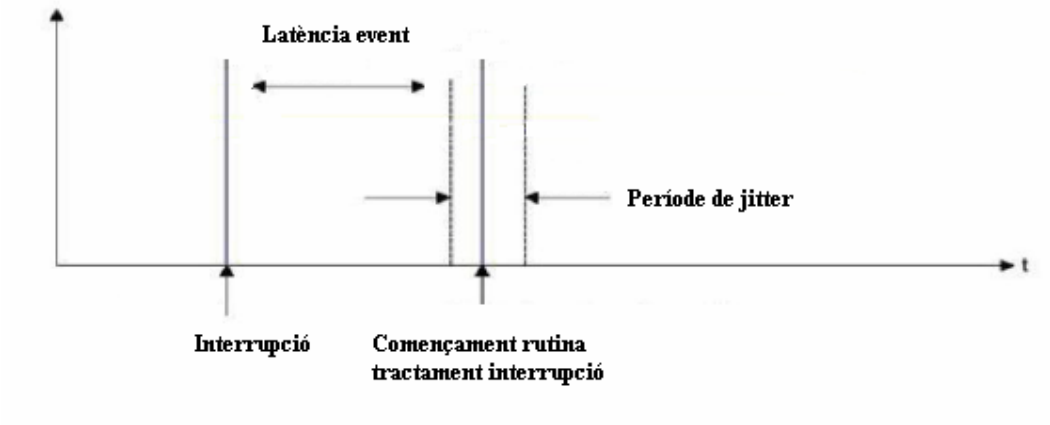


Figura 2.2: *Event interrupció i període de jitter.*

2.6 Planificació en sistemes de Temps Real

La planificació és l'assignació de recursos i temps a unes tasques de forma que es compleixin determinats requisits d'eficiència. Els requisits que tenen els planificadors depèn del sistema en que treballa el planificador. En sistemes sense temps real, els requisits buscats són els de minimitzar els temps d'execució. En canvi, els sistemes en temps real, els requisits buscats seran els de compliment d'uns límits temporals.

Els mètodes de planificació tenen un algoritme de planificació que donarà l'ordre en que s'executaran les tasques i un mètode per analitza el comportament temporal del sistema

La planificació d'un sistema pot ser tan dinàmica com estàtica:

- Planificació estàtica: Els requisits temporals s'analitzen abans de l'execució i determina l'ordre. Aquesta planificació minimitza la carrega en temps d'execució però per altre banda, es poc flexible.

- Planificació Dinàmica: L'ordre en que s'executaran les tasques es resol en temps d'execució i l'anàlisi temporal també es farà en temps d'execució. Aquests tipus de planificadors aporten una carrega considerable en temps d'execució.

De planificadors estàtics hi ha **RM** (Rate Monotonic) i **DM** (Deadline Monotonic) i de planificadors dinàmics hi ha **EDF** (Earliest Deadline First) i **LLF** (Least Laxity First). Seguidament es mostra un esquema de la classificació de les polítiques de planificació

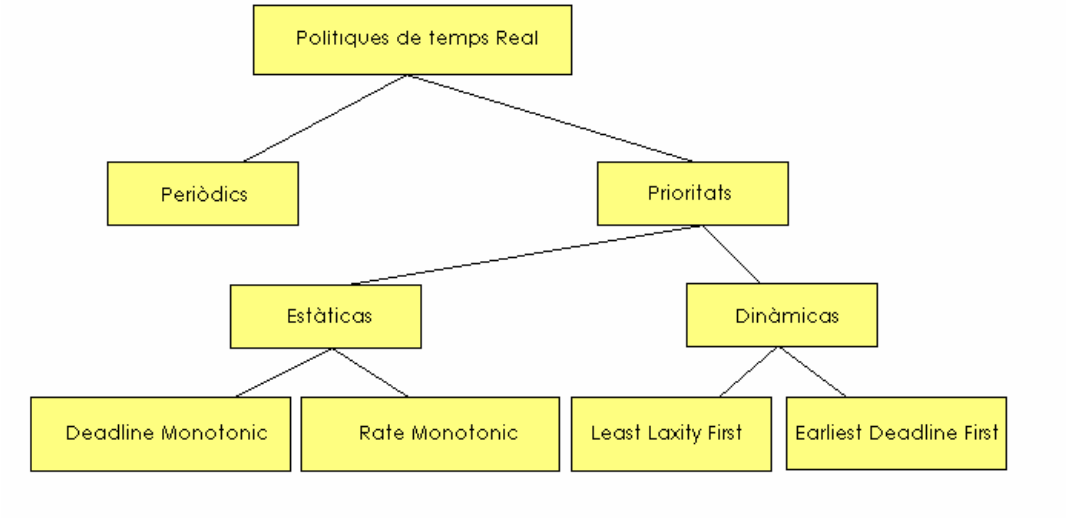


Figura 2.3: *Classificació Polítiques de temps Real*

Si les tasques s'executen cíclicament aleshores la planificació es fa sobre el primer hiperperiòde en temps de disseny, el mínim comú múltiple del període, i com que estem en un sistema en que totes les tasques s'executen de forma periòdica, després del primer hiperperiòde es torna a repetir les seqüència.
També hi ha les polítiques basades en prioritats, aquest es un dels sistemes més usats. Planificadors amb prioritats n'hi ha de dos tipus, estàtiques i dinàmiques. Les estàtiques treballaran amb prioritats estàtiques en canvi les dinàmiques ho faran amb prioritats dinàmiques en temps d'execució.

2.6.1 Prioritats estàtiques

En aquest tipus de sistemes, el límit de temps d'execució es el període de la tasca. Les tasques són independents, per tant, no es comparteixen recursos. Hi ha dos tipus de planificadors per a prioritats estàtiques.

1.- Rate Monotonic: En la fase de disseny, s'assignen les prioritats de forma inversa als períodes. En temps d'execució s'activa la tasca amb més alta prioritat. Aquest tipus de planificador és òptim per a sistemes amb prioritats estàtiques. Rate Monotonic té algoritmes per garantir que la planificació serà correcta.
El **test de garantia RM** diu que si es compleix una certa desigualtat per un cert conjunt de tasques, aquestes seran planificables. La desigualtat ha de ser entre el factor d'utilització i el límit garantit d'utilització. Aquest tipus de test serà suficient però no necessari, es poden trobar escenaris en que un conjunt de tasques no compleixin aquest test però que siguin panificables.
El **test de temps de finalització** un escenari de planificació serà planificable sota qualsevol assignació de prioritats si totes les tasques compleixen el seu plaç màxim d'execució en el pitjor cas.

El pitjor temps d'execució és el temps en que la tasca tarda més en finalitzar, això passa quan totes les tasques de prioritat més alta s'activen alhora conjuntament amb aquesta. Aquest test serà suficient i necessari al contrari del test anterior.

2.- Deadline Monotonic: Les prioritats s'assignaran de forma inversament proporcional al plaç d'execució. En aquest tipus de planificació només serà vàlid utilitzar el test de temps de finalització. El Rate Monotonic es convertirà en un cas particular del Deadline Monotonic. El DM serà òptim per planificadors amb prioritats fixes que compleixin que el temps màxim d'execució sigui menor o igual al període d'execució de les tasques. Es pot afirmar que si un conjunt de tasques no es planificable amb l'assignació de prioritats segons DM no serà planificable amb cap altre manera d'assignar les prioritats.

Un cop vistos els dos sistemes de planificació per a prioritats estàtiques, s'ha de tenir en compte una sèrie de punts importants que afectaran a la manera de construït els algoritmes de planificació explicats anteriorment.

(i) Canvis de context: El temps en posar i treure una tasca del processador no es nul·la. En sistemes de planificació basat en prioritats tant sols hi ha canvis de context quan s'activin tasques amb major prioritat. Per a cada activació d'una tasca amb major prioritat resultarà dos canvis de context.

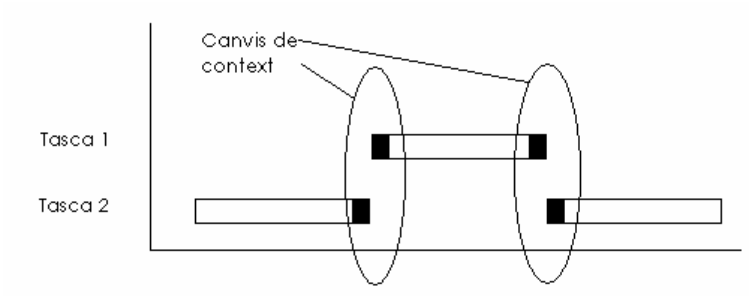


Figura 2.4: *Esquema canvi de context*

(ii) Gestió de recursos: En els sistemes de temps real les tasques no són independents entre elles. Aquestes tasques comparteixen dades entre elles usant l'exclusió mútua i han de sincronitzar-se per col·laborar entre elles. Per a realitzar tot això de forma eficient i segura la solució passarà per protegir les seccions crítiques mitjançant semàfors. L'exclusió mútua evita que diferents tasques accedeixin a recursos compartits en les seccions crítiques. Un dels problemes de l'exclusió mútua és la inversió de prioritat. La **inversió de prioritat** passa quan una tasca de major prioritat necessita entrar en una secció crítica controlada per una tasca de menor prioritat, la tasca de major prioritat haurà de bloquejar-se a l'espera que s'alliberi. En el següent exemple s'il·lustra millor la inversió de prioritats.

Existeixen 3 tasques:

A: tasca amb alta prioritat

B: tasca amb mitjana prioritat

C: tasca IDLE que només s'executa sinó hi ha cap tasca en la cua.

La tasca A i C accedeixen a un mateix recurs, per tant aquestes tasques usaran semàfors per assegurar l'exclusió mútua.

En un moment donat C s'està executant i entra en secció crítica fent un wait (mutex), mentre C està en la secció crítica es produeix un event que fa que A s'assigni al processador i com que fa un wait (mutex) es queda bloquejat esperant que C faci un signal (mutex) alliberant el semàfor, però resulta que abans que s'executi C altre cop, B s'activa i com que no hi ha cap tasca en el processador, aquesta l'agafa. Al cap d'una estona de tenir B en el processador, tenim el següent estat:

A: Bloquejada esperant que C alliberi el mutex.

B: Executant-se com si fos la tasca amb major prioritat.

C: bloquejada esperant que no hi hagi cap tasca al processador.

En aquest moment hi ha la inversió de prioritat, B s'està executant com si tingues més prioritat que totes les altres tasques per causa del bloqueig de A.

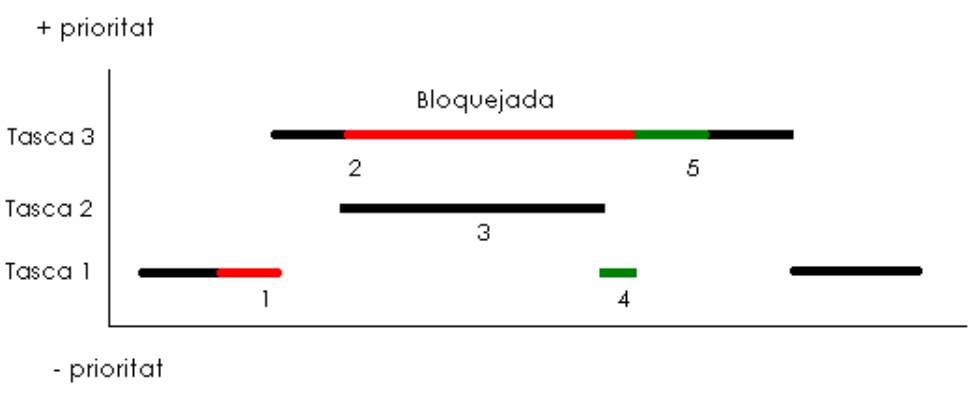


Figura 2.5: *Inversió de prioritats*

Aquest problema va passar amb la Mars Pathfinder, l'aparell enviat el 14 de Juliol de 1997 el qual experimentava freqüents resets que provocaven la perduda de dades recollides. Quan la Mars Pathfinder tenia que recollir dades i enviar-les es trobava que la tasca de recollir dades no tenia temps d'acabar per que se li havia assignat una baixa prioritat, per arreglar-ho tant sols van haver d'augmentar la prioritat d'aquesta tasca i així no entraria en l'efecte anomenat inversió de prioritats.

En la figura anterior es veu un exemple de inversió de prioritats:

1.- La tasca 1 entra en secció crítica controlada per un mutex.

2.- Tasca tres s'activa però el mutex de la secció crítica està bloquejat per la tasca 1

3.- Tasca dos s'executa ja que no té secció crítica i no hi ha cap tasca en execució. Aquesta tasca està funcionant com si fos la tasca amb més prioritat i està retardant l'execució de la tasca 3.

4.- S'allibera el semàfor de la secció crítica en la tasca 1.

5.- De seguida s'allibera en la tasca 3 i es comença a executar ja que es la tasca amb més prioritat. Aquesta tasca s'està executant amb retard.

Existeixen varies formes de corregir l'error de la inversió de prioritats, es poden donar diferents prioritats en les seccions crítiques com es va fer en la Mars Pathfinder, ordenar l'ordre d'execució offline o fer que les seccions crítiques siguin interrumpibles.

(iii) Servei aperiòdic: Hi ha un temps que no utilitzen les tasques crítiques periòdiques que es pot usar per: portar un log del sistema per monitoritzar el que està passant, refinar l'execució de les tasques, comprovar integritat del sistema, etc. Tot això es farà sense posar en perill l'execució de les tasques crítiques. Es poden classificar el servei aperiòdic en tres:

1.- Servidor en background: En aquest tipus de serveis aperiòdics, les tasques es van posant en una cua fins que queda plena, les tasques tant sols seran executades quan no hi hagin tasques crítiques pendents per ser executades. Per realitzar un servidor en background l'únic que s'ha de fer és crear una tasca servidora amb la prioritat més baixa, aquesta tasca espera rebre tasques aperiòdiques i les va executant.

2.- Servidor per Polling: En aquest mètode el que fa es afegir una tasca periòdica com a tasca crítica. Els valors de període, temps de còmput i la prioritat s'elegeixen de forma que el conjunt sigui planificable. Les tasques aperiòdiques es van guardant en una cua i quan el servidor entra en execució comença a executar les tasques que te guardades fins que no té més tasques o finalitza el seu temps de còmput.

3.- Servidor Diferit: Aquest mètode es com el servidor per Polling però ara la tasca de servidor se li assignarà la prioritat més gran.

2.6.2 Prioritats dinàmiques

En polítiques de planificació amb prioritats dinàmiques les tasques no tenen una prioritat inicial. La prioritat de cada activació s'escollirà de forma dinàmica. Per tant, es pot tenir que varies activacions de la mateixa tasca es podran fer amb diferents prioritats. Normalment serà mes eficient que les polítiques de planificació amb prioritats estàtiques. Existeixen dos polítiques de prioritats dinàmiques.

1.- Earliest Deadline First: Les tasques s'executen per ordre dels seus respectius temps límits. La primera tasca que s'executarà serà aquella que abans finalitza el seu temps per ser executada. Aquesta decisió es pren en temps d'execució. El desavantatge que té aquest mètode es l'existència de freqüents comparacions de temps de finalització de plaç. L'avantatge d'utilitzar EDF és la facilitat de l'anàlisi que s'ha de fer per escollir la prioritat de la tasca.

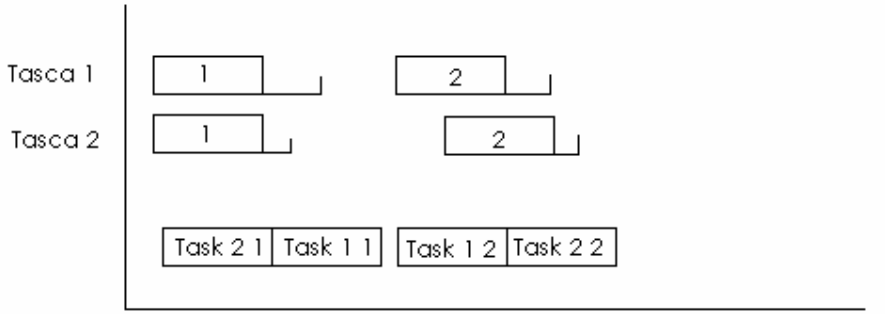


Figura 2.6: *Exemple EDF.*

En l'exemple de la il·lustració es mostra una exemple d'execució de dos tasques utilitzant el mètode de planificació EDF. Es mostra com primer s'executa la tasca 2 ja que el seu plaç d'execució acaba abans. En la segona execució, la tasca 1 serà la que s'executi abans ja que el plaç d'execució d'aquesta arriba abans. En el moment d'execució s'han de fer els càlculs pertinents per saber en cada moment quina tasca s'haurà d'executar abans.
Com ja s'ha comentat anteriorment, tota planificació porta associat un algoritme i un mètode per verificar si el conjunt serà planificable amb l'algoritme. L'algoritme EDF té associada una formula per saber si serà planificable. La formula diu que el sumatori dels quocients entre el temps de còmput i el període de totes les tasques ha de ser més petit o igual que 1 per a que el conjunt sigui planificable.

2.- Least Laxity First: Aquest mètode consisteix en assignar la prioritat a les tasques de forma inversament proporcional a la seva holgura. La holgura d'una tasca es el temps de mes que té una tasca per executar-se sense afectar els temps de la planificació. S'assignarà doncs, la prioritat més alta a la tasca amb holgura més petita.
Aquest algoritme al igual que EDF pot aconseguir un 100% de factor d'utilització. En el mètode EDF no es necessari conèixer el temps de còmput de cada tasca, en canvi amb LLF aquest paràmetre serà requerit. Això serà una desavantatge ja que l'holgura es calcula en base al temps de còmput, aquest càlcul es inexacte ja que s'assumirà el pitjor cas per a calcular l'holgura.

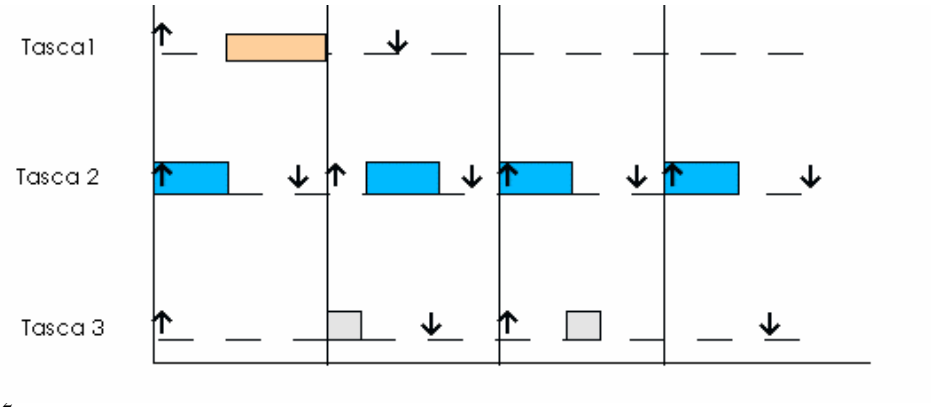


Figura 2.7: *Exemple LLF.*

En l'exemple es mostra l'execució de tres tasques sota la política Least Laxity First, es comprovar com sempre s'executa la tasca amb menys holgura, és a dir, la tasca que menys temps de més té per ser executada, es veu com la tasca 2 és la que té menys temps i per això s'executa abans. Després en el segon període, s'executa dos vegades per una que s'executa la tasca 3.

2.7 Exemples de sistemes en Temps Real

Finalment només quedarà veure uns exemples de sistemes en temps real. Es poden necessitar aquests tipus de sistemes en diferents escenaris.

1) Control d'un flux: En aquest exemple es té una canonada amb una vàlvula. Es necessita un sistema que controli el cabal d'aigua que passa per la canonada. Segons el valor del cabal s'obrirà o es tancarà la vàlvula per que passi l'aigua. És necessari un sistema de temps real que mesuri el cabal d'aigua i prengui les decisions de forma precisa per a que al final de la canonada no arribi a sobrecarregar-se.

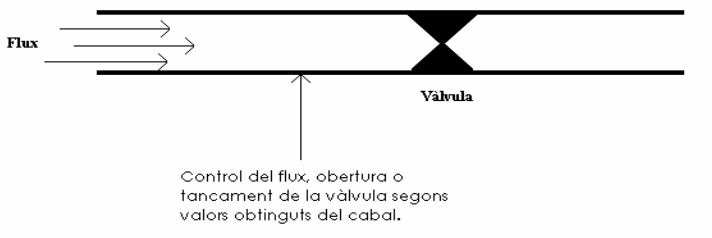


Figura 2.8: *Sistema de temps real: Control d'un flux*

2) Control d'un procés: En aquest tipus de sistemes de temps real hi ha uns processos a controlar dins d'un sistema complet.

En l'exemple hi ha un dipòsit amb aigua en el que s'ha de controlar el volum d'aigua que hi ha i la temperatura, per fer-ho es disposa d'un sensor de temperatura i un de altura. Depenent dels valors obtinguts per els sensors l'ordinador central prendrà unes decisions.

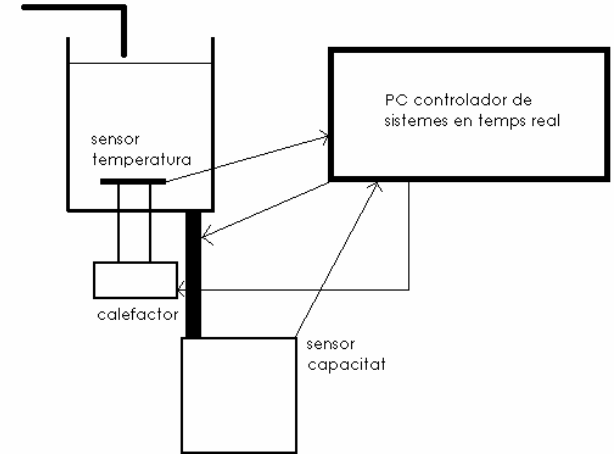


Figura 2.9: *Sistema de temps real: Control d'un procés*

3) Control en fabricació: Un altre escenari en el que es requereix un sistema de control en temps real es en un procés de fabricació. Hi ha un ordinador que controlarà el procés. Aquest tipus d'escenari es estudiat en assignatures com planificació de la producció. Existeix una seqüència d'operacions des de que entren els materials per fer les peces fins que aquestes surten. Tots aquestes operacions seran controlades per un computador central.

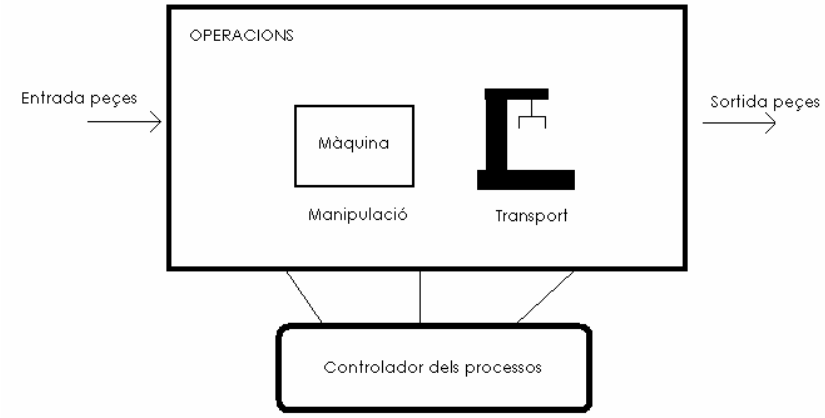


Figura 2.10: *Sistema de temps real: Control de producció en la fabricació.*

2.8 Sistemes Operatius en Temps Real

Els Sistemes Operatius de Temps Real es poden trobar en molts llocs i són tant útils com tots els altres Sistemes Operatius. Es poden trobar en controls aeris, instruments mèdics, a la indústria (petrolera, química, etc.), en l'automobilisme, etc. Són moltes les aplicacions que es donen als sistemes de temps real. És de gran importància seguir els avenços fets en aquest camp ja que en el futur dels Sistemes Operatius conviuran els Sistemes de temps Real distribuïts multiprocessador. El bloc principal del projecte dona gran importància als Sistemes Operatius de Temps Real ja que el projecte es basa en estudiar el temps real i com aplicar-lo amb un ordinador personal i fer-ho amb eines de codi lliure.

El gran augment de complexitat en els aparell microelectrònics fa que es necessiti d'un control exhaustiu d'aquests, si amb això s'afegeix que aquests aparells poden ser encastats i poden controlar tasques tan crítiques com la navegació d'un avió o el control de l'òrbita d'un satèl·lit, i tenen la necessitat de respondre en temps determinats es fa necessari l'ús de Sistemes Operatius en Temps Real.

Per tenir un Sistema Operatiu en Temps Real no es precisa d'una gran computadora, per exemple, el sistema operatiu que governa una llançadora especial pot usar un hardware amb 2mb de memòria, i microprocessador no gaire potent. La importància d'un Sistema Operatiu en Temps Real no es la rapidesa d'aquest, per tant no necessita un hardware molt avançat, tant sols necessita el hardware necessari per complir uns temps determinats, per tant, el hardware vindrà determinat pel sistema a controlar.

2.8.1 Sistemes Operatius

El hardware per si sol no serveix de res, te la necessitat de tenir associat un software per que tingui alguna utilitat. El Hardware amb ajuda del Software podrà guardar, processar, i recuperar informació, és a dir manipular la informació. El Software per computador es pot classificar en dos classes: els programes de sistema que controla l'operativa del hardware i els programes d'aplicació els quals resolen problemes pels usuaris. Exemples de programes d'aplicació n'hi ha a infinitats, programes de gestor, de finances, etc.. Però el que interessa pel projecte són els programes de sistemes, l'exemple clar de programa de sistema és el Sistema Operatiu el qual controla tots els recursos del computador i el qual donarà la base per a construir els programes d'aplicació.

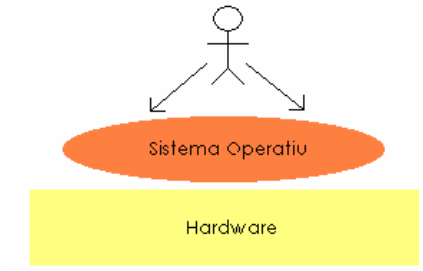


Figura 2.11: *Sistema Operatiu (capa intermèdia)*

En l'origen de la història dels computadors, els sistemes operatius no existien i la introducció d'un programa d'aplicació es convertia en un increïble esforç que tant sols algunes persones ben preparades podien fer-ho. Quan es volien crear programes d'aplicació per resoldre problemes pràctics, es tardava tant que no sortia a compte. Amb tota aquesta situació es va començar a pensar en crear una capa entre el hardware i l'usuari, així l'usuari podrà fer un ús eficient del hardware de forma senzilla. Aquesta capa intermèdia serà el que es coneix actualment com el Sistema Operatiu.

2.8.1.1 Definicions

No existeix cap forma única de definir que és un Sistema Operatiu, hi ha diferents definicions que són vàlides.

(i) Es pot imaginar un Sistema Operatiu com els programes instal·lats en el software que fan usable el hardware. El hardware només fa còmput i el Sistema Operatiu dóna la facilitat d'usar aquest còmput al usuari i administren els recursos d'aquest hardware per obtenir un bon rendiment.

(ii) Els Sistemes Operatius són administradors dels recursos hardware, computador, E/S, mitjans d'emmagatzematge, dispositius de comunicació, etc..

(iii) El Sistema Operatiu és un intermediari entre l'usuari i el hardware, i l'objectiu d'aquest es proporcionar al usuari una forma fàcil per a que pugui crear programes d'aplicació. L'objectiu principal del Sistema Operatiu és doncs, donar una facilitat d'us del hardware al usuari i com a objectiu secundari, que el hardware s'utilitzi de forma eficient. Aquest últim objectiu, es veurà més endavant que en Sistemes Operatius en Temps Real és més important que qualsevol altre objectiu.

(iv) Un Sistema Operatiu, és el programa més important de l'ordinador. Per que funcionin altres programes, serà necessari l'existència del Sistema Operatiu. Els sistemes Operatius realitzen tasques bàsiques com reconeixement dels dispositius d'E/S, enviar imatge a la pantalla, etc..

2.8.1.2 Funcions d'un Sistema Operatiu

El Sistema Operatiu com a Software que simplifica l'ús del computador té unes funcions associades essencials per a la gestió de l'equip. Com a funcions més destacades es troben: Gestió de recursos de l'equip i donar una interfície per l'usuari.

Com a gestor de recursos, el Sistema Operatiu administra la CPU, la memòria, la E/S, l'administració de recursos, etc.. Com que són moltes les coses a controlar des del Sistema Operatiu, s'han dividit les tasques que té que realitzar un Sistema Operatiu.
Un Sistema Operatiu ha de gestionar els processos, gestionar la memòria, gestionar els fitxers, gestió E/S, seguretat i comunicació. Totes aquestes tasques estan molt ben diferenciades.

2.8.1.3 Història

1940-50: En aquest període comencen aparèixer els primers computadors, aquest s'utilitzen sense cap Sistema Operatiu ja que encara no existeixen. L'ús dels computadors és molt complicat i requereix molt de temps.

1950-60: Comencen aparèixer de forma discreta els Sistemes Operatius amb l'objectiu de facilitar les tasques als usuaris. En aquells temps el funcionament dels Sistemes Operatius era bastant simple, es carregaven els programes a través d'unes cintes o targetes perforades (monitor resident).

1960-70: En aquesta època van sorgir canvis notables, entren en joc noves tècniques com la multiprogramació en el qual la computadora pot tenir més d'un programa d'usuari en execució i aprofitar les E/S per executa altres programes. També el temps compartit el qual permet l'execució de més d'un programa i usuari, al igual que en la multiprogramació, la computadora serà compartida entre els diferents programes/usuaris amb la diferència que ara si un programa porta un temps concret executant-se, passarà la CPU a un altre programa i així és compartirà el processament entre diferents programes/usuaris.
El [1]Temps Real també és una altre concepte introduït en aquesta època, hi ha doncs Sistemes Operatius que executen les tasques en temps real i per últim existeixen el multiprocessador, Sistemes Operatius que poden gestionar varis processadors alhora.

1970-80: L'electrònica avança i apareixen grans projectes per crear Sistemes Operatius, però molts sortien molt cars o es necessitava molt més temps de l'esperat. La complexitat augmenta molt.

1980-90: Apareixen els grans Sistemes Operatius que ara són els més usats per l'usuari, Windows (MS-DOS) i Apple Macintosh. Aquest Sistemes Operatius és van crear utilitzant explícitament la tercera definició anomenada anteriorment, és a dir, l'objectiu principal d'aquests Sistemes Operatius és donar la màxima facilitat a l'usuari per utilitzar la màquina i com a objectiu secundari, que aquesta es faci de forma òptima.

[1] Concepte explicat en els punts 2.1, 2.2, 2.3, 2.4

2.8.1.4 Classificació dels Sistemes Operatius

Amb el temps, han sorgit multitud de Sistemes Operatius que han fet que es puguin classificar en diferents categories depenent de les característiques de cadascun. La classificació es fa a partir de les característiques dels Sistemes Operatius i del ús o aplicació que se li donarà.

(i) Sistemes Operatius per lots: Els Sistemes Operatius per lots, processen una gran quantitat de treball amb molt poca interacció amb l'usuari. Aquests Sistemes Operatius són dels més antics i poden obtenir uns alts temps d'execució ja que el procesador està molt més usat gràcies a la seva simplicitat. L'objectiu d'un Sistema Operatiu per lots és el processament d'una gran quantitat de processos sense o amb poca interacció amb l'usuari. L'usuari carrega una seqüència de tasques a realitzar i li dona el control al Sistema Operatiu per que les executi. Són Sistemes Operatius destinats a alts índex de processament. Aquest Sistemes Operatius seran aconsellables per a tasques de llarga durada.

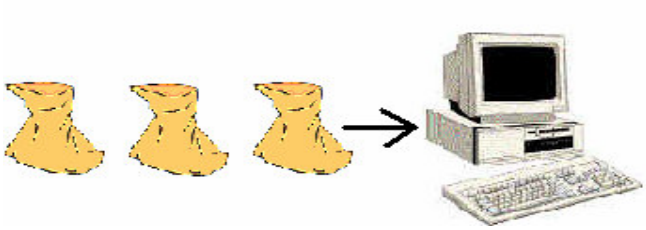


Figura 2.12: *Sistema Operatiu per lots*

(ii) Sistemes Operatius de multiprogramació: La característica principal d'un Sistema Operatiu multiprogramació és que pot executar més d'una tasca a la vegada i així el processador el processador estarà sempre en ús. Per aconseguir això, el Sistema Operatiu el que fa es posar les vàries tasques en memòria per poder-les executar totes en qualsevol moment i així tenir sempre el processador ocupat. Aquest tipus de Sistemes suporten sovint a multiusuaris, això implica afegir seguretat al Sistema Operatiu. La característica principal doncs serà que un Sistema Operatiu multiprogramació tindrà varies tasques competint pels recursos del sistema.

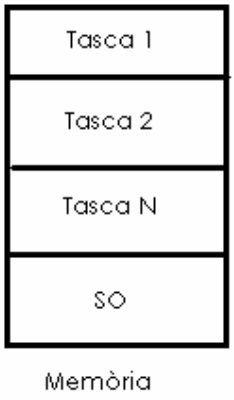


Figura 2.13: *SO Multiprogramació*

(iii) Sistemes Operatius de Temps Compartit: Els Sistemes Operatius de Temps Compartit simulen que cada usuari té tot el control dels recursos del computador. L'usuari creu que té disponibles tots els recursos en tot moment. El Sistema Operatiu el que farà es gestionar tots aquests recursos de forma eficient per a que els usuaris creguin que tenen aquests en tot moment. Tindrem doncs una gran carrega sobre el Sistema Operatiu. Un exemple clar es el SO utilitzat en les WorkStation utilitzant el SO Sun.

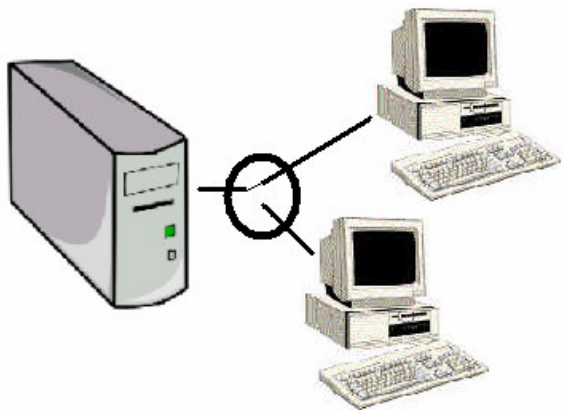


Figura 2.14: *Sistema Operatiu de Temps Compartit.*

(iv) Sistemes Operatius Distribuïts: Aquests Sistemes Operatius el que fan es distribuir les diferents tasques en diferents processador en un mateix equip o en diferents equips. Donen total transparència al usuari, un usuari llença una tasca a executar-se, ell no sap en quina màquina s'està executant, però rebrà els resultats com si l'estigués executant en el mateix PC.

Generalment aquests Sistemes Operatius proporcionen mecanismes per a la compartició global de recursos.

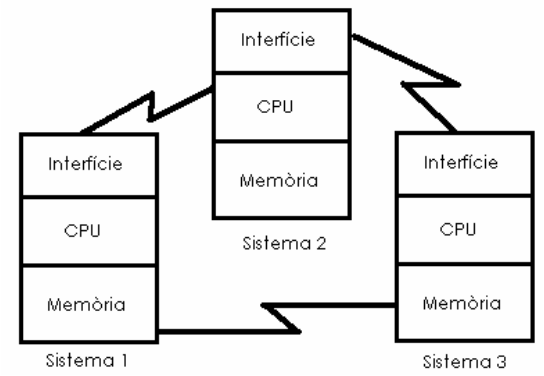


Figura 2.15: *Sistema Operatiu Distribuït*

(v) Sistemes Operatius de Xarxa: Són Sistemes Operatius que s'interconecten en una xarxa per compartir recursos i la informació. Actualment la majoria d'empreses informatitzades tenen un Sistema Operatiu de xarxa per poder interconnectar tots els components de la xarxa i poder compartir la informació.

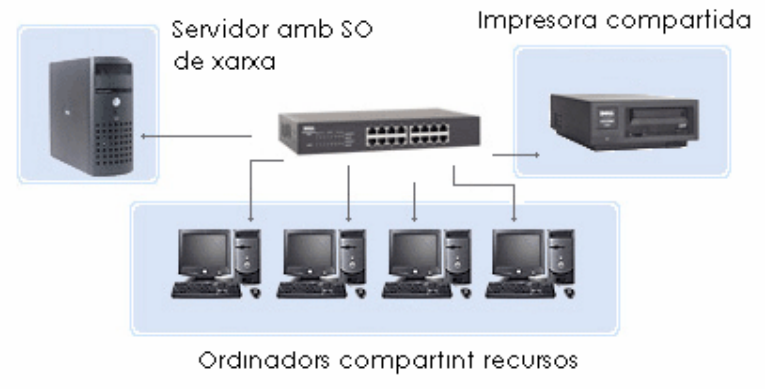


Figura 2.16: *Sistema Operatiu de Xarxa*

(vi) Sistemes Operatius en Temps Real: Aquest tipus de Sistema Operatiu és el que té protagonisme en el projecte. Serà necessari pel control de processos en temps real.
Aquests tipus de Sistemes Operatius no donen importància a l'usuari sinó que tot el treball fet serà pensant en les pròpies tasques. S'utilitzen quan hi ha una gran quantitat de successos. Normalment aquests Sistemes Operatius són molt específics com per exemple Sistemes Operatius pel control aeri, per la borsa, etc.
Han de ser capaç de tractar una gran quantitat de successos amb un temps o plaç determinat, amb temps d'execució precís. El procés d'aquestes ràfegues de successos serà de milers en un segon sense perdre'n ni un. A de ser capaç de tractar-los per sota de l'ordre dels [2]ms.

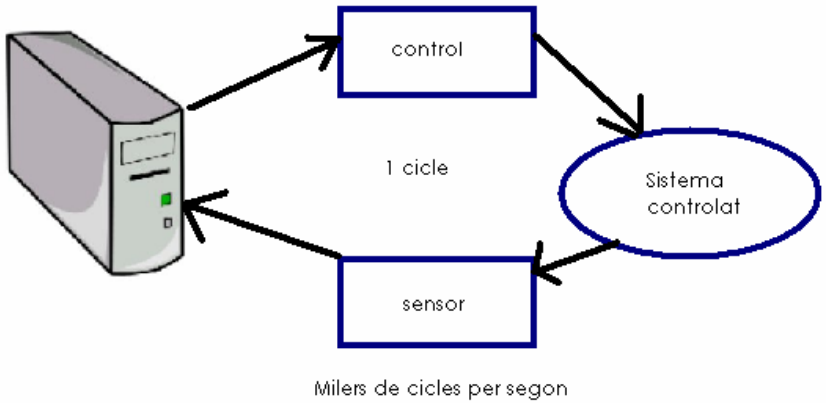


Figura 2.17: *Sistema Operatiu en Temps Real.*

[2] ms:milisegons, un segon partit en mil parts

2.8.2 Característiques

En un Sistema Operatiu de Temps Real no tant sols és important que les tasques es facin de manera correcte sinó que aquestes s'han d'executar en un temps establert.
Tenen una forta interacció amb l'exterior, reben els estímuls els tracten i retornen una sortida (control), aquests tipus de sistemes s'anomenen reactius ja que reaccionen al exterior. Normalment les tasques d'un Sistema Operatiu de Temps Real són periòdiques i es basen en rebre estímuls, tractar-los i enviar una sortida a l'exterior. Aquest llaç periòdic pot fer-se en l'ordre dels microsegons i es possible que en un segon es repeteixi milers de vegades i no pot perdre ni un sol estímul. Aquest es el fonament principal i més important d'un Sistema de Temps Real, poder processar molts estímuls sense perdre'n cap en molt poc temps.
Normalment es cau en l'error que un Sistema Operatiu de Temps Real ha de ser ràpid i això no es cert, la importància és complir els temps definits i el determinisme d'aquest.

2.8.3 Arquitectura

Per entendre l'arquitectura d'un Sistema Operatiu en Temps Real primer s'ha de veure l'arquitectura d'un Sistema Operatiu Convencional.

En un Sistema Operatiu Convencional la memòria està dividida en l'espai per l'usuari i l'espai del sistema. Els usuaris accedeixen pensant que tenen tot l'espai d'usuari per ells gràcies a la gestió del kernel multitasca. Per accedir a l'espai del sistema es farà mitjançant crides a sistemes i així donar transparència als programes d'usuari sobre els detalls físics de l'arquitectura.
Les funcionalitat d'un Sistema Operatiu Convencional serà la gestió de processos, gestió fitxers, gestió memòria, interacció amb el hardware, etc.. En canvi, en un Sistema Operatiu de Temps Real la funcionalitat principal i requerida es proveir d'uns temps de resposta necessitat per les tasques de temps real.

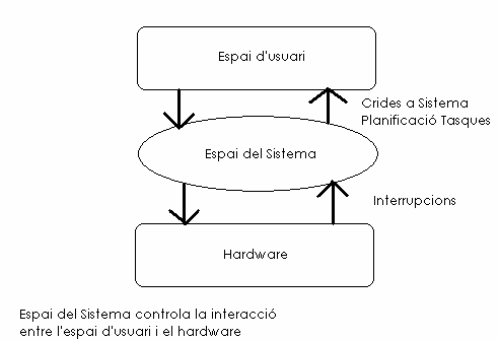


Figura 2.18: *Arquitectura Sistema Operatiu Convencional.*

Una altre diferència entre un Sistema Operatiu Convencional i un de Temps Real, es que en el primer, normalment es tenen uns valors baixos de latència i jitter, però els Sistemes Operatius de Temps Reals requereixen que aquests valors estiguin determinats i no depenguin de la càrrega del sistema.

Ara un cop s'entén com està estructurat un Sistema Operatiu Convencional i quines diferències existeixen amb un Sistema Operatiu de Temps Real, s'ha de veure com transformar aquesta arquitectura per aconseguir-ho.
Per fer-ho s'ha de reduir el jitter i la latència, per fer això s'han desenvolupat diferents arquitectures que modifiquen l'anterior.

(i) Preemptable Kernel: Aquesta tècnica el que fa es modificar l'espai del Sistema de forma que els processos de kernel i temps real s'executin amb major prioritat i puguin interrompre als processos de menor prioritat. Aquesta tècnica necessitarà modificar el gestor d'interrupcions del sistema per a no bloquejar els processos de major prioritat. Les tasques de temps real s'executaran com a tasques del sistema i aleshores tindran major prioritat que les tasques que no són de temps real i així el planificador podrà decidir correctament sempre prioritzant els processos en temps real. Aquests tipus de tècnica millora notablement el temps de jitter i la latència.

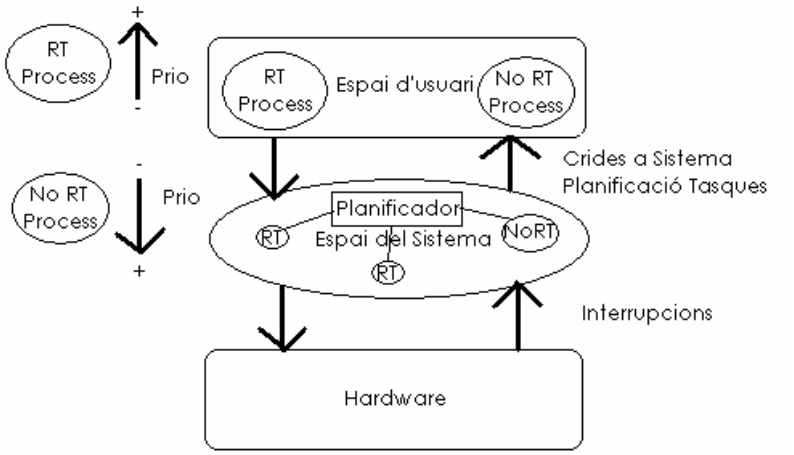


Figura 2.19: *Arquitectura Preemptable.*

Aquest tipus d'arquitectura facilita a l'usuari desenvolupar fàcilment aplicacions, té protecció de memòria i l'usuari no pot corrompre el Kernel. L'usuari també disposarà de tots els serveis disponibles en l'arquitectura convencional. Dins d'aquests avantatges hi ha uns punts molt en contra per aconseguir el paràmetres òptims.

El rendiment no es suficient per aconseguir la baixa latència de l'ordre dels microsegons, es produeix un canvi molt gran en el codi del kernel, s'està fent un kernel nou. El rendiment global del sistema baixarà notablement.

Es pot concloure que preemptable kernel no serà una arquitectura bona per obtenir un Sistema Operatiu en Temps Real.

Aquesta tècnica modifica l'espai del sistema totalment, hi ha altres tècniques que consisteixen en modificar parts de l'espai del sistema, aquestes tècniques es denominen kernel Patch.

(ii) Kernel Patch: Aquesta tècnica consisteix en modificar l'espai del sistema i afegir un segon kernel, per dotar el Sistema Operatiu de temps real. Existeixen tres tècniques:

1. Micro-Kernel: Aquesta tècnica afegeix una capa entre el hardware i l'espai del sistema. Aquesta capa s'anomena HAL (Hardware Abstraction Layer). Aquesta capa controla l'execució de les tasques de temps real i executa el kernel del sistema com una tasca en [3]background. El kernel del sistema tant sols s'executarà quan no hi hagi cap tasca executant-se. Aquesta capa controla les interrupcions i s'assegura que les tasques en temps real s'executin amb màxima prioritat. Un exemple d'aquesta arquitectura és RTLinux, el Sistema Operatiu utilitzat en el present projecte.

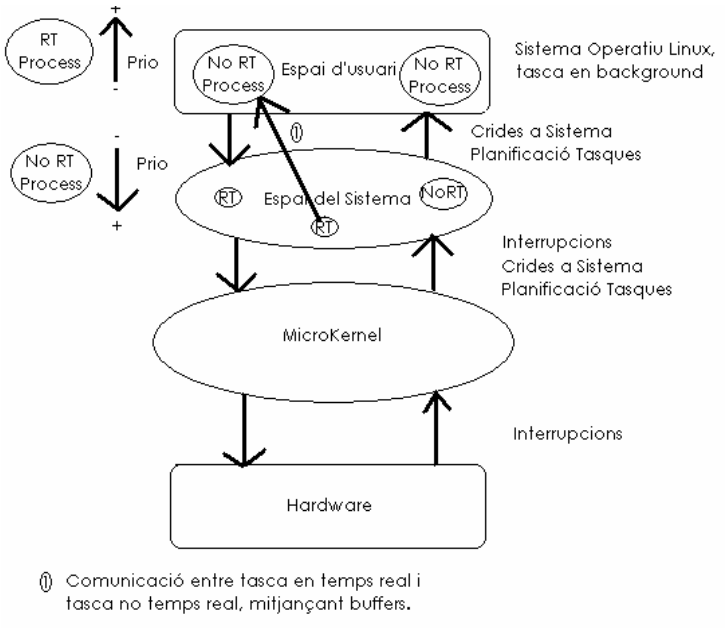


Figura 2.20: *Arquitectura Mikro-Kernel*

[3] Procés executat en segon pla.

2. Nano-Kernel: Aquesta tècnica consisteix en posar una capa que captura les interrupcions hardware i permet l'execució en paral·lel de varis sistemes. Aquesta tècnica no es directament per crear un sistema operatiu en temps real ja que per sobre el nano-kernel es pot trobar sistemes de temps real o no. Un exemple d'aquesta estructura es ADEOS.

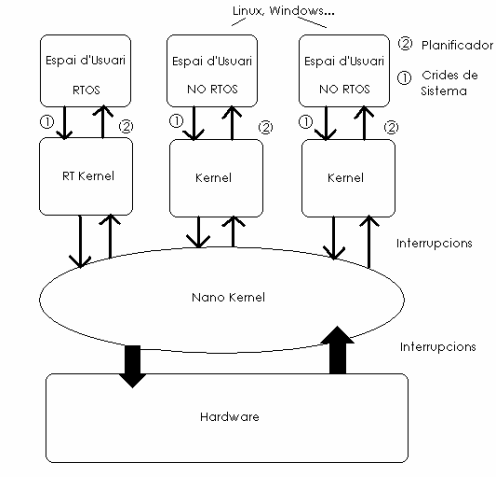


Figura 2.21:*Arquitectura Nano-kernel*

3. Recurs-Kernel: La última tècnica a tractar entre les de kernel-patch, es la que utilitza un kernel nou com a porta als recursos (sistema de fitxers, ports, etc.). Aquesta porta es per l'espai d'usuari i per l'espai de sistema. El recurs-kernel a part de capturar les interrupcions hardware, gestiona els recursos que demanaran les diferents tasques tan com les d'usuari com les de sistema.

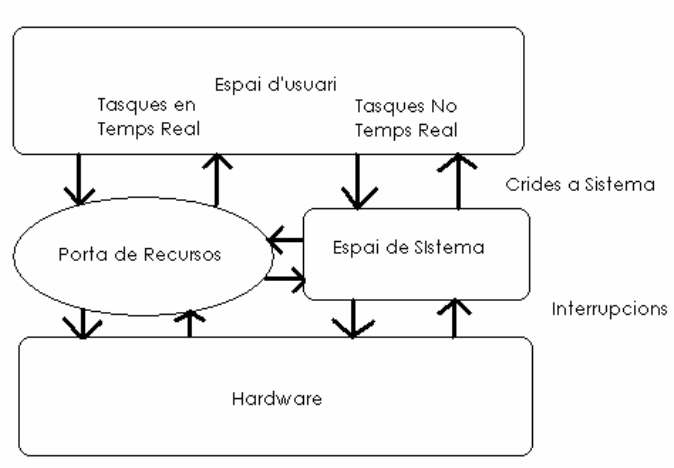


Figura 2.22: *Arquitectura Recurs-Kernel*

La tècnica de kernel patch dóna millors resultats que la de preempted kernel, obtenint latències de l'ordre dels 5 microsegons, clarament millor que l'anterior tècnica. El Kernel de linux s'executarà com una tasca en no temps real i per tant, amb la prioritat més baixa. La planificació d'aquests tipus d'arquitectures resultaran una planificació totalment determínistica, per tant es compliran els dos requisits indispensables per obtenir un Sistema Operatiu en Temps Real.

Però no tot seran punts positius, el desenvolupament de tasques en temps real serà bastant més complexa i per tant s'haurà de tenir coneixements amplis de linux. Les característiques de temps real seran implementades en mòduls com es veurà més endavant. És pot arribar a corrompre el kernel si és comet algun error.

2.8.4 Classificació dels Sistemes Operatius en Temps Real

Els Sistemes Operatius en Temps Real es poden classificar de diverses formes, es poden classificar en base a la importància dels temps límits, segons l'escala de temps, segons la integració amb el sistema , segons la forma de processament i per últim, segons la tècnica de planificació utilitzada.

(i) Segons la importància dels temps límits: Existeixen tres grups**, sistemes de temps real estricte** els quals un incompliment dels plaços establerts comportarà una falla total del sistema, sense la possibilitat de continuïtat. Per tant amb aquests sistemes es més important acabar les tasques en el temps assignat. Els **sistemes de temps real flexible**, a diferència del sistema anterior, aquests poden incomplir algunes vegades les restriccions temporals però això farà que el sistema respongui amb menor qualitat. I per últim en aquesta classificació hi ha els **sistemes operatius ferm**, aquests sistemes són sistemes operatius durs però poden tolerar algunes falles si la probabilitat que passi això és baixa.

(ii) Segons l'escala de temps: En aquesta classificació hi ha tres grups altre cop, **sistemes basats en rellotge**, s'executen a intervals repetits i molt curts, per a tasques periòdiques que s'executaran cada cert temps. Els **sistemes basats en events** les accions es realitzaran asíncronament, quan es produeixi un succés s'executarà la tasca que tractarà tal succés. Per últim en aquesta classificació, els **sistemes interactius** els quals s'inicien a intervals irregulars, quan per exemple es prem una tecla del teclat i això ha de produir una certa tasca.

(iii) Segons la forma de processament: Aquest tipus de classificació farà una divisió en dos tipus, **sistemes centralitzats** quan el processament de les tasques a executar es farà en un sol node, la comunicació es fa mitjançant la memòria, en canvi en els **sistemes distribuïts** el processament de les tasques es farà entre diferents nodes, aquests es comunicaran per la xarxa, el consum de temps en comunicació serà bastant important en aquests tipus de sistemes.

(iv) Segons la integració amb el hardware: N'hi ha de dos tipus, els **encastats i el no encastats,** en els **sistemes encastats**, el sistema operatiu de temps real estarà dins un subconjunt d'un sistema més complexa. En canvi en els **sistemes no encastats**, seran independents del hardware extern.

(v) Per últim, segons la tècnica de planificació: Altre cop es diferencien dos grups, els **sistemes estàtics**, els quals totes les seves tasques s'en saben les seves característiques i en temps de disseny es pot planificar l'execució. No s'admeten tasques noves en temps d'execució. En els **sistemes dinàmics**, es poden tenir tasques conegudes i que en temps d'execució apareguin noves tasques i poder-les planificar de forma eficient.

2.8.5 Distribucions de SO en Temps Real

Existeixen diferents distribucions de Sistemes Operatius en Temps Real, a continuació es descriuen algunes les quals abarquen tots els tipus d'arquitectures anteriorment explicades, especialment la micro-kernel que serà la utilitzada en el projecte. Per últim, s'anomenen alguns projectes que hi ha actualment però que no interessen tant ja que no són de codi lliure.

2.8.5.1 Sistemes Operatius amb llicencia GNU/GPL

1. Chimera OS (Preemptable-Kernel): Chimera és sistema operatiu real de multiprocessadors dissenyat per software reconfigurable per sistemes robotics.
   Chimera es caracteritza per la seva alta multiplicitat ens les característiques ofrenades i gràcies això pot oferir un desenvolupament ràpid de reconfiguració.
   Disposa d'un nucli de temps real el qual té programació estàtica i dinàmica, aquest nucli permet multitasca per tant podran executar-se múltiples tasques de forma concurrent.
   Pel que fa a la planificació tan pot suportar planificació per prioritats fixes (Rate Monotonic) i planificació amb prioritats dinàmiques (Earliest Deadline First).

Aquest tipus d'arquitectura una de les avantatges que té, es la fàcil implementació d'aplicacions en temps real. La distribució Chimera conté multitud de biblioteques per a ús general i biblioteques de desenvolupament tal com biblioteques de matemàtiques, de comandes d'intèrprets, etc.

Incorpora complets i complexes controladors d'errors. Es dels únics Sistemes Operatius que és de multiprocessador, a diferència de altres sistemes que fan rèpliques de la CPU i les comunica amb un protocol específic.

Chimera proporciona la majoria de característiques disponibles en Sistemes Operatius de Temps Real, més les eines necessàries pel desenvolupament de sistemes de control basat en captadors i mòduls reconfigurables.

Com ja s'ha explicat, l'arquitectura preemptable no aconsegueix bons resultats pel que fa a les latències i jitter, per tant, de ben segur que serà un bon Sistema Operatiu en Temps Real i serà complet però per l'arquitectura que s'utilitza no obtindrà els requeriments desitjats. Aquest projecte però ja no està actiu.

2. Linux/RK (Recurs-Kernel): Linux Recurs Kernel com el seu nom indica, utilitza l'arquitectura de Recurs-Kernel. Utilitzarà un Kernel per abstraure del hardware, aquest Kernel serà una porta als recursos que oferirà el hardware. Les tasques quan tenen la necessitat d'un recurs (CPU per exemple) es comuniquen amb el recurs kernel per demanar el recurs. Aquest kernel gestionarà els recursos entre les tasques depenent de les seves prioritats. Aquest és un projecte no massa conegut i on no hi ha massa documentació per implementacions.

3. ADEOS OS (Nano-Kernel): ADEOS el que busca es crear un entorn que serveixi per compartir el hardware per múltiples Sistemes Operatius o instàncies. Aquests s'anomenen dominis, aquests dominis poden viure junt amb altres dominis, independents entre ells, l'únic domini que han de conèixer tots es ADEOS. ADEOS consta d'una arquitectura Nano-Kernel. ADEOS notifica a cada mòdul les interrupcions hardware, les crides a sistema i els events. Cada domini disposarà d'una prioritat, aquesta prioritat determinarà l'ordre en que els events són tractats en els dominis. Aquests events són col·locats en una pipeline. La manera de treballar de ADEOS és la següent:

   Un domini entra en execució, aleshores pot passar dos coses, que finalitzi voluntàriament ja que acabat totes les interrupcions que tenia que tractar o un domini més prioritari a bloquejat el domini. La importància de ADEOS es la necessitat de controlar de forma determinista el flux d'interrupcions.

4. RTAI (Micro-kernel): RTAI es una implementació de linux en temps real, afegeix un petit kernel i tracta el kernel de linux com una tasca de menor prioritat. RTAI afegeix una amplia selecció de mecanismes de comunicació entre processos i altres serveis de temps real. RTAI proporciona LXRT un mòdul per facilitar el desenvolupament d'aplicacions en temps real en l'espai d'usuari. RTAI tracta el kernel de linux com una tasca més executada en background, per tant, s'executa amb prioritat més baixa. Les interrupcions externes seran tractades per RTAI Interrupt dispatcher, el qual enviarà les interrupcions al kernel de linux.

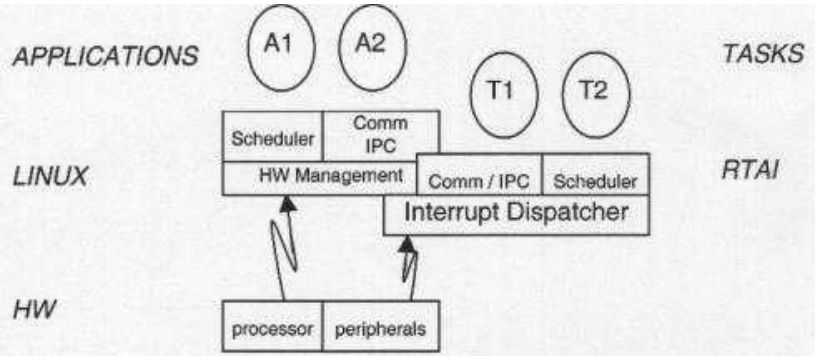


Figura 2.23: *Arquitectura RTAI*

Es pot veure com cada kernel disposa d'un planificador independent. El kernel en temps real disposa del Interrupt Dispatcher el qual tracta les interrupcions externes i les envia al kernel linux. En canvi les interrupcions del processador com els missatges d'error són enviats directament el kernel de linux. RTAI inclou HAL (Hardware Abstraction Layer) que s'utilitza per interceptar les interrupcions hardware i processar-les després. L'objectiu principal de HAL és minimitzar el número de canvis sobre el codi del kernel i per tant millorar el manteniment de RTAI.

La planificació en RTAI es implementat com un mòdul dedicat, aquest té sempre una tasca executant, el kernel de linux, com a tasca amb menor prioritat.

RTAI té tres tipus de planificadors, Uniprocessador (UP), Multiprocessador simètric (SMP) i Multi-Uniprocessador (UMP) en aquest últim es obligat assignar la CPU a cada tasca.

La comunicació entre tasques es farà mitjançant les FIFO. RTAI pot llançar interrupcions quan escriu en una FIFO, així l'espai d'usuari pot capturar-les. Una altre eina bàsica són els semàfors, RTAI proporciona un API per utilitzar semàfors, cada semàfor té associada una FIFO. Un altre mecanisme, els mailbox, qualsevol número de processos pot enviar i rebre missatges de i des de un mailbox. Un mailbox emmagatzema missatges fins a un límit establert.

RTAI proporciona actualment gestió de memòria dinàmica en temps real, reservant espai en temps d'execució.

Com a última eina important, es pot anomenar el mòdul LXRT, la interfície de l'espai d'usuari amb RTAI. Això el que permet es executar aplicacions en temps real en l'espai d'usuari. Aquest fet evita molts cops el mal funcionament del sistema operatiu ja que no s'accedeix a posicions de memòria crítiques.

Gràcies aquesta eina, es pot depurar el procés en temps real fins que no conté cap error i aleshores es pot convertir en un mòdul. En el període de depuració s'està treballant sobre procés en temps real soft, en canvi un cop depurat i sense errors es pot convertir en un procés en temps real hard. LXRT proporciona APIs per la comunicació entre processos de temps real o usuari. Quan s'està en mode soft s'executarà amb el planificador del kernel de linux, en canvi si s'està en hard-real time, es farà amb el planificador de temps real. LXRT proporciona eines per facilitar que una tasca pugui passar d'un estat a un altre (hard a soft). Encara que els programadors de RTAI veuen bé el desenvolupament d'aplicacions en temps real estricte utilitzant LXRT, el temps de resposta no es tant bo com l'execució de tasques com a mòduls del kernel.

5. RTLinux (Micro-Kernel): RTLinux és un Sistema Operatiu que executa linux com un thread de menor prioritat, igual que RTAI. RTLinux executa tasques de temps real i rutines d'interrupció en la mateix màquina que el linux estàndard. El pitjor cas de temps es entre que es detecta la interrupció i s'executa la primera instrucció de la rutina de tractament en aquesta interrupció, aquest temps es la latència i està en els 10 microsegons, en la plataforma x86. RTLinux té dos tipus de llicencies, RTLinux/Open, disponible sota la llicencia GPL però que des del 2001 no es treballa i RTLinux/Pro versió comercial.

   En el projecte es dona especial importància a la distribució Open ja que l'objectiu es aconseguir un sistema en temps real utilitzant llicencies gratuïtes.

   RTLinux segueix l'arquitectura Mikro-Kernel, per tant es bastant semblant a la de RTAI. RTLinux és un sistema operatiu petit i simple que està entre mig del kernel de linux i el hardware. Per a que RTLinux tingui el control del hardware s'han de fer unes modificacions prèvies al kernel de linux, dotar-lo del control directe de les interrupcions hardware, control del rellotge hardware e implementació d'un rellotge virtual per Linux, el control de les interrupcions per part de Linux es reemplaçat per dues funcions que permeten activar o desactivar interrupcions virtuals.

   RTLinux proporciona un entorn d'execució sota el kernel de linux, per tant es té més baix nivell i per això molts dels serveis de linux no podran ser utilitzats.

2.8.5.2 Sistemes Operatius Comercials

1. QNX OS (Mikro-Kernel): Sistema Operatiu de Temps Real basat en UNIX que compleix amb la normativa POSIX. Un Sistema Operatiu orientat a sistemes encastats. Distribució comercial de QNX Software System. Està basat en una arquitectura Mikro-Kernel. QNX incorpora l'arquitectura del model de procés universal UPM. UPM permet reduir el temps de desenvolupament.
   QNX està disponible per plataformes x86, MIPS i PPC.
   El Mikro-Kernel de QNX es altament optimitzat i de mida reduïda, 12K.
   Aquest tipus de distribució no interessa tant ja que es comercial.

2. VxWorks OS (Mikro-Kernel): Un altre Sistema Operatiu en Temps Real comercial és VxWorks, comercialitzat per Wind River System. Com QNX OS, VxWorks s'utilitza majoritàriament per a sistemes encastats. Consta de l'arquitectura Mikro-Kernel, potser la més usada actualment. Suporta PPC, Motorola, ARM, x86, SPARC, etc.
   Mikro-Kernel wind de espai reduït. El planificador és multitasca basat en prioritats.
   Al ser una estructura Mikro-Kernel serà escalable, es podran ampliar amb mòduls d'E/S, sistemes de fitxers, suport de xarxa, suport per a sistemes SMP, suport per a memòria virtual i suport per a sistemes distribuïts.

En la taula següent es mostra un petit resum del estat de cada distribució comentada en aquest apartat. Pel projecte s'ha escollit finalment RTLinux, encara que ja no segueixi activa la versió open source, s'ha cregut que és la mes adient per l'estudi que s'està fent. Com ja es veurà més endavant, els seus valors de latencia i jitter són òptim i es disposa de informació necessària per poder implementar el sistema en temps real que es necessita.

Distribució	Llicència	Estat	Arquitectura	Versió actual
Chimera OS	-	Parat	Preemptable-Kernel	3.2
Linux/RK	Open Source	Actiu	Recurs-Kernel	2.4.18
ADEOS OS	GNU/GPL	Actiu	Nano-Kernel	2.6
RTAI	GNU/GPL	Actiu	Mikro-Kernel	3.2
RTLinux/OPEN	GNU/GPL	Parat	Mikro-Kernel	2.4.21
QNX	Comercial	Actiu	Mikro-Kernel	4.24
VxWorks	Comercial	Actiu	Mikro-Kernel	6

2.9 Sistema Operatiu Real Time Linux (RTLinux)

RTLinux va ser creat per Michael Barabanov i Victor Yodaiken. Actualment està disponible la versió comercial RTLinux/Pro distribuïda per FSM Labs empresa dels creadors. Existeix una versió open source que ja no hi treballen, van deixar-la per iniciar la versió comercial però existeixen comunitats que segueixen el desenvolupament. En aquest projecte interessarà la versió open source que s'ha està en la versió 2.4.

RTLinux funciona sobre PPC i x86. RTLinux té una arquitectura Mikro-Kernel patentada per Victor Yodaiken, la qual s'ha utilitzat per crear altres distribucions com RTAI. RTLinux adopta, des de la versió 2, l'estàndard POSIX, pthreads, etc.

També existeix una versió per a plataformes multiprocessador, en la qual es pot assignar tasques als processadors. RTLinux en part es un parche que s'aplica al kernel de linux i una altre part són mòduls que es poden carregar. Cada versió de RTLinux està fet expressament per a versions de linux concretes.

2.9.1 Característiques

1- És un planificador expulsiu per prioritats fixes, per a l'execució de tasques en temps real.
2- Les tasques poden ser tan periòdiques o bé activades per una interrupció.
3- Incorpora mecanismes per comunicar-se amb tasques no crítiques, aquests mecanismes son les FIFOs.
4- Les tasques en temps real s'executen en mode supervisor, és a dir, poden accedir a la E/S, reprogramar interrupcions, etc.
5- Executa el nucli de linux com una tasca més en background.

El nucli de linux ja no activa/desactiva directament les interrupcions, quan ho vol fer li dirà a RTLinux. Si linux vol desactivar una interrupció li diu a RTLinux però aquest no la desactivarà ja que les tractarà i les retornarà a Linux quan es reactivi. El que es té doncs, és que linux perd el control del sistema, ens podem trobar en la situació en que sembla que el sistema s'ha penjat però en realitat el que passa es que tota la CPU està destinada a les tasques en temps real i no donen temps a que s'executi linux.

RTLinux es bastant eficient, i deixa a linux les tasques no crítiques com les de monitorejar els valors captats per les tasques crítiques. En un processador 486 de Intel, RTLinux pot mostrar amb precisió de fins a 1 mostra cada 30 microsegons.

RTLinux suporta l'estàndard POSIX, Portable Operating System Interface, i la X de UNIX. POSIX és una sèrie de crides al sistema operatiu definides pel IEEE. Actualment POSIX es divideix en tres grans grups:

1. POSIX.1, Core Services (implementa les crides de C)

- · Creació i control de processos.
- · Senyals.
- · Excepcions de punt flotant.
- · Excepcions per violació de segment.
- · Excepcions per instrucció il·legal.
- · Errors del bus.
- · Control de temps.
- · Operacions de fitxers i directoris.
- · Pipes
- · Biblioteca C
- · Instruccions d'E/S i control dispositius.
- · Senyals.
- · Excepcions de punt flotant.
- · Excepcions per violació de segment.
- · Excepcions per instrucció il·legal.
- · Errors del bus.
- · Control de temps.
- · Operacions de fitxers i directoris.
- · Pipes
- · Biblioteca C
- · Instruccions d'E/S i control dispositius.

2. POSIX.1b, extensions per a temps real

- · Planificació amb prioritats.
- · Senyals de temps real.
- · Control de temps.
- · Semàfors.
- · Intercanvi de missatges.
- · Memòria compartida.
- · E/S síncrona i asíncrona.
- · Bloqueig memòria.

3. POSIX.1c, extensions per threads

- · Creació, control i destrucció threads.
- · Planificació.
- · Sincronització.
- · Control de senyals.

2.9.2 Arquitectura RTLinux

RTLinux com ja s'ha comentat, té una arquitectura Mikro-Kernel. A diferència d'altres sistemes operatius, RTLinux no afegeix noves crides a sistema ni modifica cap de les existents.
RTLinux es situarà entre el hardware i el propi sistema operatiu de linux, creant com una mena de màquina virtual per a que linux segueixi funcionant.
RTLinux obtindrà tot el control del hardware i executarà linux com a tasca en background.

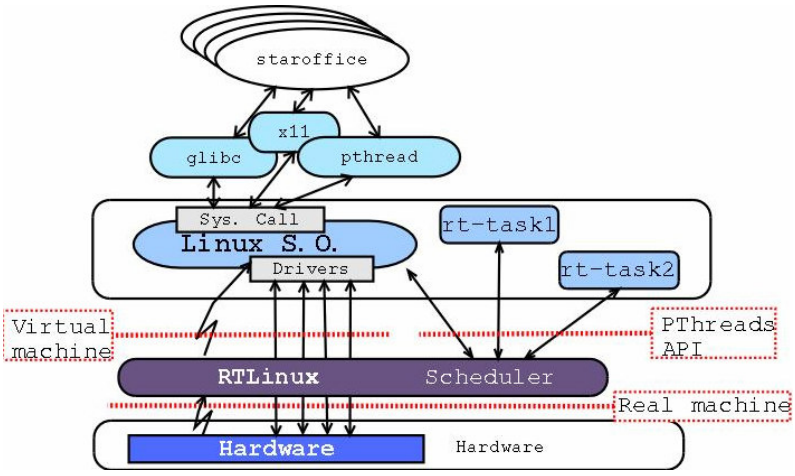


Figura 2.24: *Arquitectura RTLinux*

En la figura anterior, es pot veure més clara l'arquitectura de RTLinux. Es pot veure com RTLinux forma una capa entre mig del hardware i el sistema operatiu. RTLinux gestionarà les interrupcions hardware i controlarà tots els recursos hardware. El sistema operatiu de linux serà una tasca més a planificar amb el planificador de temps real. La creació de tasques es farà mitjançant l'API de pthreads. El sistema operatiu de linux tindrà el seu propi planificador el qual gestionarà les seves tasques. L'arquitectura doncs es de tres capes. La capa hardware, que serà la màquina real, entre mig RTLinux que serà la màquina virtual sobre la qual s'executaran les tasques en temps real i el sistema operatiu de linux que conformarà la tercera capa, la capa de més alt nivell.
Les tasques de temps real, rt-tasks, comparteixen el mateix espai de memòria que el nucli, per tant, cada rt-task pot accedir a totes les variables i funcions. Una rt-task no pot fer crides a sistema del sistema operatiu, aquestes s'executen en mode supervisor, poden executar qualsevol instrucció i poden accedir a totes les E/S. Per executar una rt-task s'utilitzaran com a mòduls que es carreguen en el sistema.

2.9.3 Mòduls del nucli

Com ja s'ha comentat, per posar en execució una tasca en temps real, RTLinux utilitza uns mòduls que es carregaran. Aquests mòduls són trossos de sistema operatiu que és poden inserir o eliminar en temps d'execució. El mòdul no es res mes que un objecte compilat, extret des d'un font en C.
Més endavant s'explicarà com s'utilitzaran aquests mòduls i com es programaran tasques en temps real en RTLinux.

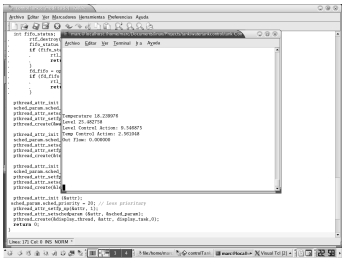
2.9.4 Aplicacions de RTLinux

1. NASA Flight Linux utilitza RTLinux, com a sistema operatiu de temps real amb sistemes encastats.

Figura 2.25: *Logo NASA FL*

2. Robot HOAP de Fujitsu, és un robot que té la capacitat de interactuar amb el medi, posseeix un sistema avançat per reconèixer imatges i té l'habilitat de comunicar-se a més de mostrar emocions. Tot això funciona sota el sistema operatiu RTLinux i es proporcionen tot una sèrie de d'eines per poder programar el robot i afegir-hi funcionalitat donant control total al usuari.

Figura 2.26: *Robot HOAP3*



3

Preparació d'un Sistema en Temps Real

---

3. Preparació d'un Sistema en Temps Real

3.1 Previ

Un cop assolits els conceptes anteriors ja s'està en disposició de crear el sistema en temps real. Serà necessari un Sistema Operatiu en Temps Real, com ja s'ha vist al capítol anterior, l'escollit serà RTLinux, es veurà el perquè s'ha elegit aquest Sistema Operatiu i com instal·lar-lo, configurar-lo i fer-lo funcionar. S'han fet alguns canvis en el codi font del Sistema Operatiu per pròpia necessitat que també es comentaran. Un cop es té el Sistema Operatiu en Temps Real llest per treballar es veurà el seu funcionament i com programar tasques en Temps Real.

També es veurà en aquest capítol alguns exemples que demostren que el nostre Sistema Operatiu es realment en Temps Real.

Una vegada es tingui el Sistema Operatiu llest i els coneixements de com funciona, quedarà elegir alguna eina per programa aplicacions en Temps Real. L'eina escollida serà el llenguatge TCL/TK.

Es veurà que és TCL/TK, perquè s'ha elegit i com s'instal·la. Un cop instal·lat tcl/tk s'instal·larà també una eina que ajudarà a programa software amb GUI avançades, aquesta eina és vtcl. Finalment es veuran alguns exemples de programació en aquest llenguatge.

Al finalitzar aquest capítol ja es disposarà del Sistema en Temps Real utilitzant totes les eines amb llicencia gratuïta, i quedarà veure que es pot dissenyar aplicacions avançades, això es veurà en el següent capítol.

3.2 Elecció de les eines

L'objectiu del projecte és veure que són els Sistemes en Temps Real i per a que serveixen. Estudiar la viabilitat de tenir un Sistema en Temps Real en un PC de sobretaula tot utilitzant llicencies gratuïtes i desenvolupar software de Temps Real seguint la mateixa línia de les llicencies gratuïtes, en poques paraules, obtenir un Sistema en Temps Real gratuït i crear software amb una interfície amigable per l'usuari.

Com ja s'ha explicat, els Sistemes Operatius en Temps Real basats en Kernel-Patch donen un millor rendiment del sistema, obtenint latències de l'ordre dels microsegons. El desavantatge que tenen en aquest sistemes és que alhora de crear tasques en Temps Real es bastant més complex que en Sistemes basats en preemptable kernel. Per millorar aquest aspecte es crearà un Software de generació de tasques en Temps Real, això es veurà en el següent capítol. Abans però, s'haurà de preparar el Sistema en Temps Real i les eines necessàries per desenvolupar Software i tenir clar com crear tasques en Temps Real.

Les eines escollides per portar a terme el desenvolupament de l'entorn seran RTLinux i TCL/TK. RTLinux com a Sistema Operatiu en Temps Real i TCL/TK com a llenguatge per desenvolupar Software de Temps Real.

3.2.1 RTLinux com a Sistema Operatiu en Temps Real

Per a crear el Sistema en Temps Real s'elegeix RTLinux. Un cop vistos en el capítol anterior alguns Sistemes Operatius en Temps Real de llicencia gratuïta s'ha acabat elegint RTLinux, tant pel seu bon rendiment com per tota la documentació disponible per desenvolupar software en temps real. RTLinux proveeix de totes les avantatges que brinda linux. Es podrà modificar tots els mòduls del sistema de RTLinux que siguin necessaris.

Se sap de l'existència d'una distribució comercial, RTLinux/Pro que de ben segur que segueix l'estructura inicial de RTLinux de codi lliure, per tant, si en un futur és vol seguir desenvolupant software per aplicacions en temps real, es tindrà l'elecció de tirar pel camí de la llicencia comercial la qual ens servirà mòduls avançats i de ben segur sense haver de modificar el software ja creat. De moment però, començarem amb l'estructura inicial que brinda el RTLinux de codi lliure que és l'objectiu del projecte. L'avantatge de treballar amb codi obert és la possibilitat de modificar el que es vulgui depenent de les necessitats. A més RTLinux conta amb una característica que poques distribucions lliures suporten, RTLinux suporta arquitectures multiprocessador, aquest és un gran avantatge ja que si es volen sistemes de temps reals complerts i que suportin aquests tipus d'arquitectures, s'haurien de pagar llicencies bastant cares. Tenim RTOS comercials molt bons, que tenen millor rendiment que els anomenats de codi lliure, però amb suport econòmic és normal que siguin eines més completes.

En la taula següent és veuen uns preus orientatius dels Sistemes Operatius Comercials. Si es volen llicencies per fer projectes de gran envergadura, seran necessàries llicencies del tipus 'team-of-developers" equip de desenvolupadors i es pot comprovar que els preus són bastant

elevats. Per tant queda clar que l'ús de Sistemes Operatius en Temps Real comercials per ús individual queda bastant limitat.

RTOS Comercial	Preu llicencia
VxWorks	23000$ (team-of-developers) 1000$ (single use)
QNX OS	1995$ (Programació amb kernel Neutrino) 2195$ (Desenvolupament sota Photon)

Per tant, no queda més remei que utilitzar un RTOS de llicencia gratuïta, no serà necessari per l'objectiu del projecte, crear un sistema de temps real molt crític en el que puguin perillar les vides de persones (exemple un controlador d'avió), sinó que el que es necessita es crear un sistema de temps real amb un bon rendiment dintre els RTOS de codi obert.

3.2.2 TCL/TK com a eina de desenvolupament de Software

TCL/TK és un llenguatge de programació interpretat i multiplataforma, aquesta última característica és molt important. Un llenguatge multiplataforma és en el que es pot programar aplicacions per vàries plataformes, per exemple, per Linux i Windows. En els punts següents es veurà tot el referent a TCL/TK, es veurà en més detall que és, com s'utilitza i de quines eines es disposa per programar.
TCL/TK s'ha elegit per desenvolupar aplicacions de temps real i donar una manera més amigable a l'usuari per interactuar amb el sistema. TCL/TK és de codi lliure, per tant aquest és un dels punts més importants alhora d'elegir TCL/TK com a llenguatge per a desenvolupar les aplicacions. Hi ha altres llenguatges de codi lliure i que servirien de igual manera que TCL/TK per a crear les aplicacions, però TCL/TK s'ha escollit per les següents raons:

- Facilitat de programació al ser un llenguatge interpretat, un llenguatge interpretat és un llenguatge que s'executa a través d'un intèrpret, al contrari que els llenguatges compilats.
- S'obtenen amb molta rapidesa aplicacions avançades.
- Comparat amb altres llenguatges interpretats, aquest és un llenguatge bastant ràpid.
- Les aplicacions són fàcil de modificar ja que separa molt clarament la part de còmput (TCL) amb la part d'interfície d'usuari (TK).

- Com ja s'ha comentat, una gran característica d'aquest llenguatge, és la propietat de multiplataforma.
- Per últim anomenar una altre gran característica, la possibilitat d'afegir noves comandes escrites en C/C++. No tant sols podem desenvolupar Software a partir del llenguatge, sinó que podem millorar el llenguatge a la manera que vulguem.

Però no tant sols són avantatges el que s'aconsegueix amb TCL/TK, hi ha també algunes desavantatges però que seran de poca importància pel que es vol aconseguir. Algun dels problemes que presenta és la lentitud respecte als llenguatges compilats, alhora d'executar una aplicació farà falta l'intèrpret i la depuració es fa difícil ja que es tracta d'un llenguatge interpretat i l'únic que fa es traduir.

3.3 Preparació de RTLinux

En aquest punt es prepara el Sistema Operatiu en Temps Real mitjançant RTLinux. Es veurà com instal·lar-lo, configurar-lo i com funciona. Es veurà com funciona l'estructura de mòduls dinàmics i la manera de programar. Tot això es veurà finalment reflectit amb exemples. Però no tot són facilitats, com a punt final a la preparació de RTLinux es descriuran alguns problemes trobats duran la preparació.

3.3.1 Instal·lació

En aquesta secció es presenta com instal·lar RTLinux en un PC per obtenir el nostre entorn en Temps Real.
S'ha de tenir clar, que no cal tenir uns coneixements molt avançats per fer el que s'explicarà a continuació, amb les explicacions següents qualsevol persona amb mínims coneixements de l'entorn de Linux podrà instal·lar-se RTLinux.
El que es necessita primer és tenir instal·lat un Sistema Operatiu Linux que serveixi per instal·lar-hi RTLinux. La instal·lació d'aquestes distribucions, actualment, són fàcil, per tant no hi ha cap problema, serà com instal·lar un Windows ja que tot es pot fer de manera visual. Un cop es té instal·lat el Sistema Operatiu Linux ja es podrà començar a fer els preparatius per instal·lar RTLinux.

1. Primer de tot es necessari obtenir el codi font d'un kernel de Linux i d'una distribució de RTLinux. Com ja s'ha explicat, RTLinux és una arquitectura de tipus Kernel Patch, s'afegirà un parche al kernel de Linux per obtenir RTLinux. També s'ha explicat que cada distribució de RTLinux té associada unes versions de kernel de Linux. S'obtenen els dos fitxers des de "www.kernel.org" (el kernel de Linux) i a fsmlabs es troba la versió de RTLinux Open.

2. El següent pas es descomprimir els dos fitxers obtinguts:

   a. La versió descarregada de Linux:
      cd /usr/src/
      tar xzf linux.2.x.y.tar.gz (x.y la subversió del fitxer)
      Un cop fet això s'obté un nou directori: /us/src/linux.2.x.y

   b. La versió descarregada de RTLinux:
      cd /usr/src/
      tar xzf rtlinux-3.*.tar.gz
      Un cop fet això s'obté un nou directori: /usr/src/rtlinux-3.*
      Per millorar l'accessibilitat per part de l'usuari es pot crear un link que apunti al directori creat i que s'anomeni RTLinux. Per fer això s'escriu:
      cd /usr/src
      ln –sf /usr/src/rtlinux-3.* ./rtlinux

   Ara per accedir al directori del codi font de Linux s'anirà a "/usr/src/linux" i per accedir al codi font de RTLinux s'anirà a "/usr/src/rtlinux".

3. El següent pas serà enllaçar RTLinux amb Linux, per fer-ho s'utilitzaran les comandes:
   cd /usr/src/rtlinux
   ln –sf /usr/src/linux ./linux
   Al fer això s'obtindrà un altre directori dins de /usr/src/rtlinux anomenat rtlinux.

4. Ja està tot preparat, ara queda mirar si les versions de RTLinux i el Kernel de Linux són compatibles. Per fer-ho s'escriu:
   cd /usr/src/rtlinux
   grep –A 2 "[^\W*VERSION" ./patches/kernel_patch*
   La sortida d'aquesta comanda donarà les versions de Kernel de Linux que suporta la distribució de RTLinux descarregada.

   També s'hauria de comprovar la versió del compilador gcc **"gcc –v"** i veure que és una versió igual o superior a la 2.7.3.2. També es pot utilitzar el compilador kgcc (gcc 2.9.1) modificant el Makefile de /usr/src/rtlinux la línia:

   CC = $ (CROSS_COMPILE) gcc

   per la línia:

   CC = kgcc

5. Ara es crearà el Kernel Patch. La instal·lació de RTLinux consisteix en aplicar el parche de RTLinux al kernel de Linux. Per fer això, s'utilitzarà la comanda "patch".

cd /usr/src/linux

patch –p1 < /usrc/src/rtlinux/kernel_patch-2.*.*

6. Ja s'ha aplicat el parche RTLinux al kernel de Linux, ara es configurarà el kernel, es compilarà i s'obtindrà una imatge del Linux en temps real. Cal saber que compilar un kernel pot portar temps i dependrà de la màquina utilitzada.
   Per configurar el kernel:
   cd /usr/src/linux
   make config
   Un cop fet això sortiran preguntes les quals donen respostes per defecte. Cal tenir en compte algunes d'importants. Cal especificar correctament el tipus de CPU de la màquina a la que s'està instal·lant RTLinux. Si la màquina tant sols té una CPU, s'haurà de desactivar l'opció SMP. Es desactivarà també el servei APM de la BIOS, aquest servei pot produir efectes imperdibles en un Sistema de Temps Real.
   Un cop configurat el kernel adequadament, es compilarà per obtenir la imatge del RTLinux, això es farà amb les següents comandes:
   make dep
   make bzImage
   make modules
   make modules_install
   Com a resultat d'això, s'obté la imatge "bzImage" que es trobarà a /usr/src/linux/arch/i386/boot. Aquesta es la imatge que s'ha d'arrencar per entrar al Sistema Operatiu en Temps Real.

7. Ara doncs s'ha de configurar el gestor d'arrencada per que carregui la imatge de RTLinux obtinguda en el pas anterior. S'haurà de configurar doncs el gestor d'arrencada LILO.
   Abans de configurar res, es copiarà la imatge bzImage al directori /boot/
   cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage
   Per configurar el gestor LILO el que es fa es crear una entrada nova per la imatge creada de RTLinux. Per fer-ho es seguiran els passos següents:
   - Editar el fitxer /etc/lilo.conf
   - Afegir l'entrada:
     image = /boot/bzImage
     label = RTLinux
     read-only
     root = /dev/hda1 (hdaX, X és el valor de la partició)
   - Escriure la comanda **/sbin/lilo** per aplicar els canvis en el gestor
   - Comprovar que la sortida ens surt
     Added linux *
     Added rtlinux

8. Ara es reinicia el Sistema Operatiu i s'observarà com surt el gestor d'arrancada el qual té com a opció el Sistema Operatiu en Temps Real RTLinux. Es seleccionarà per passar a configurar-lo.

9. El procés és semblant al de configurar el kernel de Linux.
   cd /usr/src/rtlinux
   make config
   En aquest punt, es permet configurar aspectes de RTLinux com l'activació de les senyals o rellotges POSIX, l'activació de operacions en punt flotant, la mida de les fifo, etc.

10. Com a últim pas només queda compilar RTLinux amb:
    make
    make devices
    make install

Seguint aquests 10 passos, utilitzant les distribucions anomenades, en principi no s'ha de tenir problemes en obtenir finalment RTLinux.
Per comprovar que tot a funcionat bé, es pot intentar d'activar el temps real amb la comanda:
start rtlinux
Si es carreguen una sèrie de mòduls sense cap error, significarà que ja està activat el temps real en el RTLinux i que tot funciona correctament

3.3.2 Funcionament

El funcionament de RTLinux és una mica especial. Un cop iniciada una sessió amb RTLinux es pot escollir quan es vol estar en Temps Real. Amb la comanda **>start rtlinux** s'inicia l'estat en temps real. El que fa aquesta comanda es afegir una sèrie de mòduls o trossos de sistema operatiu que habiliten el temps real. Amb la comanda **>stop rtlinux** es trauran tots els mòduls inserits deshabilitant el temps real.
RTLinux doncs, funciona a base de mòduls que es poden inserir i eliminar en temps d'execució.

3.3.3 Mòduls

Els mòduls son trossos de Sistema Operatiu que es poden inserir i extraure en temps d'execució. Quan es compila un programa que conté diferents fitxers fonts, el que es fa primer es compilar-los per separat per obtenir fitxers objecte .o i finalment s'enllacen tots per obtenir un executable.
Ara imaginem que el fitxer objecte que conté el main del programa es pugues posar en execució i que el Sistema Operatiu fos capaç d'enllaçar els altres fitxers objecte quan es necessites, en

temps d'execució. Doncs linux es capaç de fer-ho amb el propi kernel. Quan linux arranca, només ho fa l'executable vmlinuz que es l'indispensable per arrancar, després es poden carregar i descarregar en temps d'execució els mòduls que es creguin necessaris.

Es pot crear mòduls nous e inserir-los en temps d'execució sense necessitat de recompilar el kernel o reiniciar l'ordinador.

Una vegada un mòdul s'ha carregat, passarà a formar part del Sistema Operatiu, podrà doncs accedir a totes les variables i estructures de dades del nucli, s'executarà amb el màxim nivell de privilegi per tant tindrà accés a totes les E/S i el mappeig d'adreces de memòria es farà directe a memòria física, per tant no hi haurà paging.

El mòduls comencen a donar algunes característiques de Temps Real, no es pot produir fallades de pàgina (més determinisme) i donen accés a tota la E/S.

Per treballar amb els mòduls de RTLinux es disposa de les següents comandes:

Mòdul	Descripció
insmod	Instal·la un mòdul en el nucli.
rmmod	Extreu un mòdul del nucli.
modinfo	Mostra informació del mòdul
modprobe	Automatitza i facilita la gestió dels mòduls
depmod	Determina dependència entre mòduls
lsmod	Llista els mòduls carregats

Per compilar un mòdul, és a dir, un fitxer font .c per obtenir el mòdul .o s'utilitzarà la següent comanda:

gcc –c –O2 –fomit–frame–pointer –DMODULE –D__KERNEL__ nomfitxer.c

- Es posa l'opció "-c" per a que el compilador compili i ensambli el fitxer, però no enllaçar-lo, únicament general el fitxer objecte.
- L'opció "-O2 –fomit-frame-pointer" serveix per generar codi optimitzat.
- Finalment, l'opció "-DMODULE –D__KERNEL__" defineix les macros MODULE i __KERNEL__ usades pels fitxers de capçalera del nucli per generar el codi apropiat.

Amb aquesta comanda s'obtindrà el nomfitxer.o llest per carregar-lo com un mòdul més al nucli.

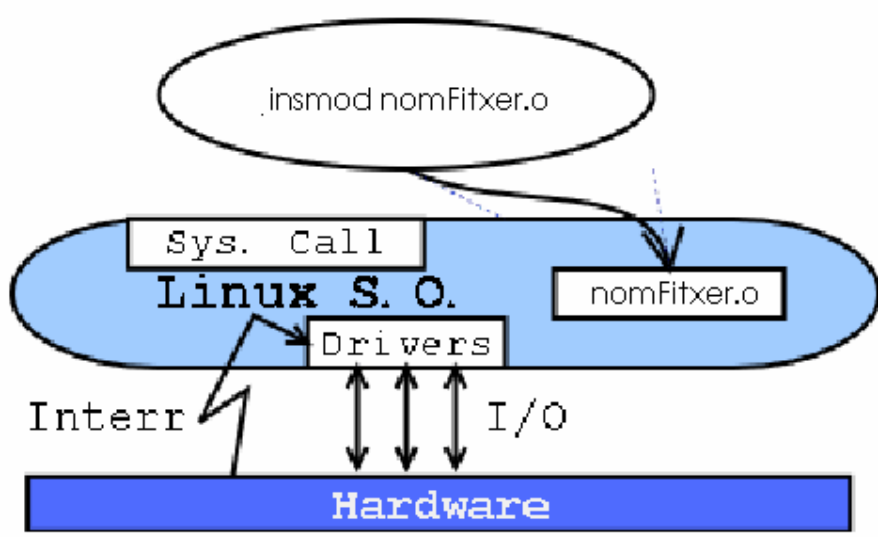


Figura 3.1: *Mòduls en RTLinux*

Un cop inserit un mòdul, es poden veure els missatges enviats pel kernel amb la comanda **dmesg** per fer tot això, serà necessari ser root.

Per inserir el mòdul s'utilitzarà **>insmod nomFitxer.o** i per extraure el fitxer **>rmmod nomFitxer,** no caldrà afegir l'extensió del fitxer al extraure un mòdul del nucli.

Una vegada un mòdul s'ha carregat al nucli, totes les seves funcions i variables públiques són accessibles des de tots els mòduls carregats o que s'hagin de carregar.

La majoria de APIs de RTLinux estan dividides en varis mòduls opcionals amb l'objectiu de poder dissenyar Sistemes de Temps Real ajustats a les necessitats de cadascú. Si una aplicació en Temps Real no necessita comunicació amb altres tasques doncs no es carregarà el mòdul de fifos.

Per tant, abans d'executar un mòdul creat , s'haurà de carregar els mòduls requerits pel mòdul implementat.

3.3.4 Rendiment

Per comprovar el rendiment del Sistema Operatiu RTLinux s'ha creat dos programes, un en temps real i un altre que no és en temps real. Aquests programes el que faran serà enviar una dada al port paral·lel amb un període de 40000 microsegons. Amb les modificacions fetes al mòdul planificador de RTLinux es podran mesurar els temps. Per obtenir resultats més precisos s'hauria de fer amb circuitería externa però amb aquest projecte s'han deixat com a bons els resultats obtinguts amb la monitorització dels temps a través del planificador.
Tot això sembla una mica complex ja que de moment no s'ha parlat de programació de tasques en Temps Real ni de modificacions al planificador de Temps Real, etc..
La idea d'ara mes que quedar-se amb el funcionament del programa, és en els resultats obtinguts que ajudaran a comprendre perquè RTLinux servirà com a Sistema Operatiu en Temps Real.

El mòdul a executar en temps real és el següent:

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#define LPT 0x367 // adreça del port paral·lel
pthread_t mytask;

void fun(int t) {
struct sched_param p;
p . sched_priority = 1;
pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
pthread_make_periodic_np (pthread_self(), gethrtime(), 40000000);
while (1) {
outb(t, LPT);
pthread_wait_np ();
}
}
int init_module(void)
{
hrtime_t now;
pthread_create (&mytask, NULL, fun, 0);
return 0;
}

void cleanup_module(void)
{
rt_task_delete(&mytask);
}
```

El mòdul a executar en no temps real és el següent:

```
// Programa a executar en no temps Real.

#include <stdio.h>
#include <fcntl.h>
#include <time.h>
#include "lpv.h" // /dev/lpv és un driver simple del port paral·lel

void main(){

int fd;
unsigned char buffer;

struct timespec t;

t.tv_sec = 0;
t.tv_nsec = 40000000;

fd = open("/dev/lpv",O_RDWR); //obrim per esciure en el port paral·lel
if (fd<0) {
perror("Could not open");
exit(-1);
}

while(1){

write(fd,"1",1);
nanosleep(&t,NULL);
write(fd,"2",1);
nanosleep(&t,NULL);

}

close(fd);

}
```

Aquests dos programes s'executaran amb i sense càrrega al sistema i es comprovarà que en Temps Real, no es nota aquesta carrega ja que li dóna màxima prioritat a la tasca en temps real per a ser executada al igual que a les interrupcions al port paral·lel i deixa en espera totes les altres tasques que no siguin en temps real.

Si primer s'executa el programa sense temps real, s'obtenen els resultats:

1) Amb càrrega al sistema (aplicacions en background):
   Temps mínim d'execució: 18.000 microsegons
   Temps màxim d'execució: 20.540 microsegons
   S'obté una mitjana d'execució de 20.000 microsegons i una desviació de **34,3 microsegons**.

2) Sense càrrega al sistema (cap procés executant-se):
   Temps mínim d'execució: 18.032 microsegons
   Temps màxim d'execució: 20.553 microsegons
   S'obté una mitjana d'execució de 20.032 microsegons i una desviació de **1,52 microsegons**.

Si ara s'executa el segon programa amb temps real, tenim les dos opcions:

1) Amb càrrega al sistema (aplicacions en background):
   Temps mínim d'execució: 19.992 microsegons
   Temps màxim d'execució: 20.018 microsegons
   S'obté una mitjana d'execució de 19.999 microsegons i una desviació de **1,35 microsegons**.

3) Sense càrrega al sistema (cap procés executant-se):
   Temps mínim d'execució: 19.992 microsegons
   Temps màxim d'execució: 20.013 microsegons
   S'obté una mitjana d'execució de 19.999 microsegons i una desviació de **0,92 microsegons**.

En el valor a fixar-se és en el de la desviació obtinguda. Es pot veure com en els dos programes, tant en temps real com no en temps real, quan no es té càrrega al sistema s'obtenen valors de la desviació pròxims a 1, això significa que la variació de temps d'execució es molt semblant en cada període. Això passa per que no hi ha cap càrrega en el sistema o casi nul·la i fa que cap altre tasca bloquegi el programa.
Ara be la part més interessant, quan s'executen els programes amb càrrega en el sistema s'obtenen resultats molt interessants e importants per l'objectiu del projecte. Quan es fa amb el programa en no temps real, la desviació creix molt, arribant a 34,3 això es causat pel bloqueig d'altres tasques de Linux a la tasca d'escriptura al port paral·lel. En canvi el programa en temps Real segueix obtenint valors pròxims a 1. això es per que RTLinux dóna en les tasques en temps real tot el control sobre el hardware. Els processos de Linux han d'esperar a que acabi la tasca en temps real, per això no es noten canvis en els temps tant si hi ha o no càrrega en el sistema.
Aquest fet assegura que es disposa d'un Sistema de Temps Real funcionant correctament, aquest exemple n'és la prova ja que al executar la tasca en temps real dona un cert determinisme.

Com més s'aproxima a 0 el valor de la desviació, més determinístic serà el sistema. Per tant, un experiment amb un Sistema Operatiu en Temps Real ideal la desviació hauria de ser 0 amb molta carrega i per tant el sistema seria totalment determinístic.

Aquest és un exemple per comprovar que s'ha elegit un bon Sistema Operatiu en Temps Real. En pròxims punts, es veuran altres exemples més pràctics i on es notarà millor la força del Temps Real. Ara per ara s'introduirà més a fons en la programació de RTLinux.

3.3.5 Configuració

Un cop instal·lat RTLinux i se sap com funciona, s'haurà de preparar pel projecte. Com que es vol mesurar els temps quan s'estan executant tasques, per saber el temps que estan en execució i quan canvien per altres tasques, s'haurà de trobar alguna manera de mostrar aquesta informació. El mòdul que fa aquesta tasca, és el planificador de Temps Real (rtl_sched.o), per tant s'ha de modificar el codi font i recompilar-lo per obtenir el planificador modificat.

El planificador utilitzat per RTLinux és de tipus FIFO i basat en prioritats estàtiques. Es basarà en les prioritats, s'executarà la tasca amb major prioritat. Per exemple, es té A=1, B=2 prioritats, i comença a executar-se A i mentre aquesta està en execució, s'activa B aleshores A es bloquejarà ja que es de menor prioritat donant pas a l'execució de la tasca B esperant a tenir la CPU lliure, un cop acabada l'execució de B, es tornarà a executar A. El planificador de RTLinux es expulsador per prioritats estàtiques.

El que es pretén es obtenir els valors de temps dels canvis de tasques en el processador, modificant el planificador. Per fer-ho, s'afegeixen unes línies de codi al rtl_sched.c que escriguin la informació necessària en una de les real time fifos. Així des de un procés no de temps real, es podrà obtenir la informació. Aquesta informació després es podrà fer servir per veure'n l'execució amb alguna eina de diagrama de temps. Més endavant es veurà una eina creada per monitorejar el pas dels processos per la CPU amb els seus temps, creada pel projecte.

Ara centrem l'atenció en modificar el planificador de rtlinux.

Analitzant el codi de rtl_sched.c es troba que en la funció **rtl_schedule** es on es troben els canvis de tasques en la CPU. Fixant-se en les següents línies de codi que hi ha en la funció:

```
if ((t->pending & ~t->blocked) && (!new_task ||
(t->sched_param.sched_priority > new_task>sched_param.sched_priority))) {
new_task = t;
now1 = gethrtime();
}
```

Notar que en aquest punt es mira si al tasca següent en la cua de tasques (t) esta pendent de ser executada i no està bloquejada per una altre tasca i que té prioritat més alta. Si es compleix, la nova tasca a ser executada serà t.

Per tant, en aquesta funció es pot obtenir la informació de canvi de tasques en la CPU. Amb la funció **`rtl_switch_to(&sched->rtl_current, new_task);`** es fa el canvi de tasca en la CPU.

El que s'ha d'afegir en aquesta funció són les següents línies al principi de la funció:

```
if ((sched->rtl_current != NULL))
                {
                if (first == 0)
                {
                        idOld = (unsigned long)sched->rtl_current;
                        taskLinux = idOld;
                                printk("Primera tasca a executar-se %ul \n\n",idOld);
                        first = 1;
                        time1 = 0;
                }
                else
                {
                        if (idOld != (unsigned long)sched->rtl_current)
                        {
                                time1 = time1 + ((gethrtime()-now1));
                                _pthread =(unsigned long)sched->rtl_current;
                                if ((unsigned long)sched->rtl_current == taskLinux) {
                                printk("LINUX: %ul amb prioritat %d - %ld \n",(unsigned
long)sched->rtl_current,sched->rtl_current->sched_param.sched_priority,time1);
                                }
                                else
                                {
                                printk("OTHER: %ul amb prioritat %d - %ld \n",(unsigned
long)sched->rtl_current,sched->rtl_current->sched_param.sched_priority,time1);
                                }
                                write (fd_fifo,&_pthread, sizeof(unsigned long));
                                write (fd_fifo,&sched->rtl_current-
>sched_param.sched_priority, sizeof(int));
                                write (fd_fifo,&time1, sizeof(long int));
                                idOld = (unsigned long)sched->rtl_current;
                                time1 = 0;
                        }

                }
        }
```

En el mòdul inicial s'inicialitza un temporitzador que en cada canvi de tasca s'inicialitzarà, la variable de temporitzador és now1.

La variable first serveix per saber quan és la primera vegada que s'entra al mòdul. La primera vegada es posa la variable first = 1 per no tornar a entrar en aquesta secció i time1 també a 0 per inicalitzar el temporitzador. Una altre variable a inicialitzar es idOld i taskLinux que serà la tasca actual en la CPU.

En les pròximes entrades a la funció, si la tasca actual és la mateixa que la tasca anterior no es farà res i el temporitzador seguirà corrent, en canvi, si la tasca actual es diferent a la tasca antiga s'obtindrà la diferencia entre el temps des de que es va iniciar la tasca i el temps actual obtenint el temps total de la tasca en la CPU. Aquesta informació s'escriurà a una real time fifo que proporciona RTLinux. En aquesta fifo s'escriu la següent informació.

- Identificador de la tasca.
- Prioritat de la tasca.
- Temps de la tasca en la CPU.

En el main del planificador, en aquest cas, en el **init_module** s'hauran d'afegir les pertinents inicialitzacions de la fifo i de les variables de temps i control que s'han utilitzat per monitoritzar els temps.

```
time1 = 0;
time2 = 0;
first = 0;
now1 = gethrtime();
contadorTime=0;

rtf_destroy(0);
        fifo_status = rtf_create(0,4000);
        if (fifo_status)
                {
                        rtl_printf("RTLinux measurement test fail.
fifo_status=%d\n",fifo_status);
                        return -1;
                }
        rtl_printf("RTLinux measurement module on CPU %d\n",rtl_getcpuid());
fd_fifo = open("/dev/rtf0", O_NONBLOCK);
                        if (fd_fifo < 0) {
                                rtl_printf("/dev/rtf0 open returned %d\n", fd_fifo);
                                return (void *) -1;
                                }
```

Per últim, queda definir les variables globals:

```
int fd_fifo;
long int idOld;
long int taskLinux;
int first;
long int time1;
```

L'explicació es potser una mica difícil d'entendre ja que es a més detall d'implementació, serveix per guiar a qui vol modificar el fitxer de planificació del RTLinux. Amb la següent figura s'il·lustra el funcionament del nou planificador obtingut després de compilar-lo.

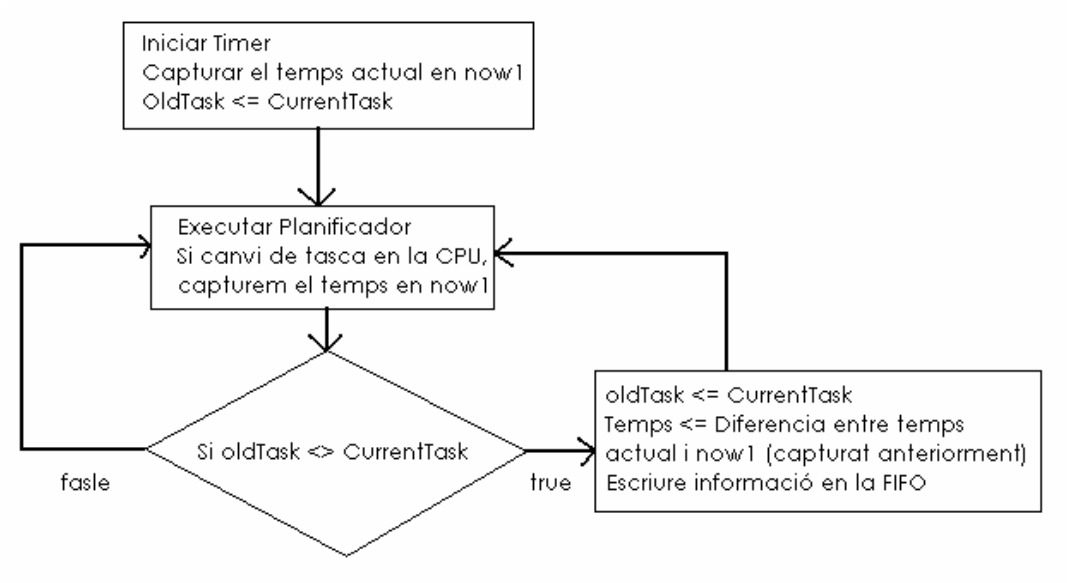


Figura 3.2: *Diagrama de flux del Planificador.*

El que s'aconsegueix amb la modificació, es emmagatzemar la informació a una real time fifo el que està passant al planificador. Així després des de un programa en no temps real es pot obtenir les dades llegint-les de la fifo i escriure-les en un fitxer per que altres aplicacions en facin l'ús que vulguin L'esquema seria el següent:

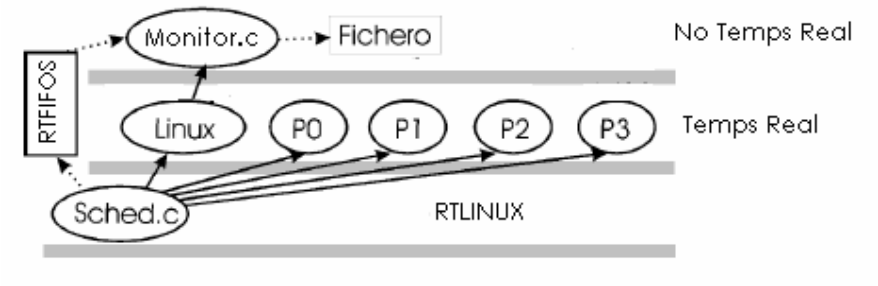


Figura 3.3: *Obtenció dades del Planificador*

3.3.6 Programació

La programació de tasques en Temps Real és fa usant el llenguatge C. Com ja s'ha explicat, quan es crea una tasca en temps real, el que es fa serà compilar un fitxer font C per obtenir-ne un objecte .o que serà un mòdul que es carregarà al nucli de RTLinux quan es necessiti.
Per comprendre com programar aquests mòduls, primer s'explica l'estructura bàsica d'un mòdul i seguidament s'introdueix a una explicació ràpida dels temes més importants en programació de tasques en temps real.

3.3.6.1 Estructura bàsica d'un mòdul

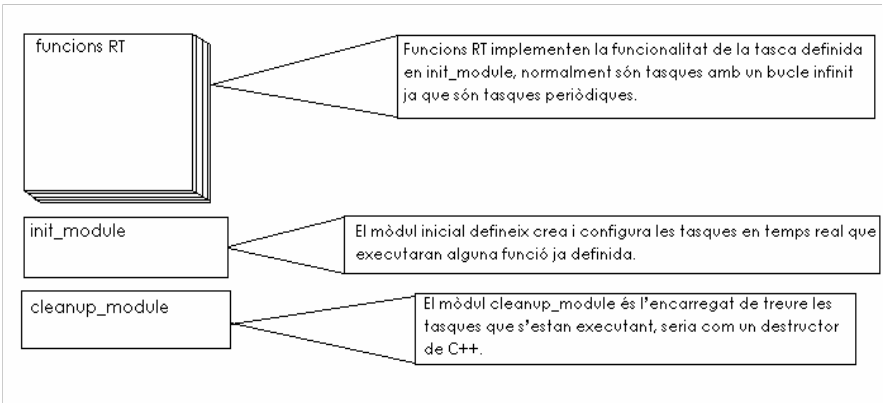


Figura 3.4: *Estructura bàsica d'un mòdul RT*

En la figura anterior, es mostra l'estructura bàsica d'un mòdul en de Temps Real, aquesta és l'estructura bàsica, a partir d'aquí es pot anar complicant i augmentant la mida però sempre seguint aquesta estructura. Seguidament es pot veure el primer mòdul en Temps Real.

```
pthread_t thread;
void * thread_code(void)
{
        pthread_make_periodic_np(pthread_self(), gethrtime(), 1000000000);
        while (1)
        {
                pthread_wait_np ();
                rtl_printf("Hello World\n");
        }

        return 0;
}
int init_module(void)
{
 return pthread_create(&thread, NULL, thread_code, NULL);
}
void cleanup_module(void)
{
   pthread_delete_np(thread);
}
```

Aquest mòdul no fa res , simplement crea una tasca que executarà el procediment **thread_code**, aquesta execució serà periòdica de període 1*10^9 microsegons. En cada període imprimirà el missatge "Hello World" en la sortida del nucli del RTLinux. Com es pot veure, aquest exemple segueix l'estructura bàsica definida anteriorment. Té el cos del codi que es la funció **init_module** que conté la creació i configuració de les tasques, després té les funcions que executaran les tasques creades i per últim té la funció **cleanup_module** per destruït les tasques.
Un cop estudiat com és l'estructura d'un mòdul, s'han de veure una sèrie de temes importants sobre programació de tasques en Temps Real.

El més important és saber com funciona la creació i gestió de tasques en temps real i l'ús de FIFOS per la comunicació entre tasques, això ajudarà a entendre el codi modificat del planificador de Temps Real. Aquesta memòria no te la pretensió de fer un manual complet sobre programació RTLinux però si que vol deixar clar al lector com iniciar-se a la programació de tasques en Temps Real i per això es interessant tractar aquests dos temes.

3.3.6.2 Creació i gestió de tasques en Temps Real

pthread_create(*thread, *attr, *start, *arg): Inicia l'execució d'una tasca, els atributs es veuran després, l'atribut per defecte = NULL.
pthread_exit(retval): Finalitza l'execució de la tasca que el crida.
pthread_join(thread,retval): Suspèn l'execució de la tasca que l'invoca fins que acaba l'execució de la tasca = thread.

Els atributs d'una tasca definiran com aquesta tasca s'executarà i amb quines propietats. Les crides més importants són:
pthread_attr_init(*attr), pthread_attr_init_destroy(*attr): Inicialitza els atributs en una tasca, els atributs són una estructura especial.
pthread_attr_[get|set]schedparam(*attr, *param): Estableix o obté la prioritat d'una tasca.

Per definir la periodicitat d'una tasca s'utilitzen les següents funcions:
pthread_make_periodic_np(thread, start_time, period): Permet despertar una tasca cada cert temps periòdic, el període s'especifica en nanosegons.
pthread_wait_np(): Suspèn l'execució de la tasca que l'invoca, és necessari que la tasca sigui periòdica per a que es torni a activar en un cert temps.

Per eliminar una tasca de temps real s'utilitza la crida:
pthread_delete_np(pthread_t thread): Aquesta crida es fa en el mòdul cleanup_module i destrueix la tasca.

Ara ja es té una mica més clar com funciona el mòdul hello.c escrit anteriorment. Ara es passa a veure com funcionen una mica per sobre les FIFO per entendre el funcionament de les modificacions que s'han fet al planificador.

3.3.6.3 Comunicació entre Tasques (FIFOS)

Les FIFO són un mecanisme basat en la implementacio de UNIX. Aquestes s'utilitzen per comunicació entre tasques de temps real o entre tasques de temps real i tasques de Linux. Per utilitzar les FIFO, s'haurà de carregar el mòdul opcional de FIFOs en temps real. La comunicació és unidireccional, cada lectura elimina les dades llegides del buffer.
Des de linux es veuen les fifo com dispositius especials de caràcters.

Per crear i destruir fifos s'utilitzarà:
rtf_create(fifo_nr,size): Aquesta funció només es pot invocar des del mòdul inicial i no des de les funcions de les tasques en temps real. Aquesta funció crea una fifo número "fifo_nr" amb una mida "size".

rtf_destroy(fifo_nr): Marca com a lliure la fifo "fifo_nr" i allibera la memòria utilitzada, normalment aquesta funció s'invoca des del mòdul cleanup_module.

Per llegir i escriure en fifos s'utilitzarà:
rtf_get(fifo_nr,&buf,nBytes): Llegeix de la fifo número "fifo_nr" el contingut del buffer "buf" amb mida "nBytes".

rtf_set(fifo_nr,&buf,nBytes): Insereix a la fifo número "fifo_nr" el contingut del buffer "buf" amb mida "nBytes".

Des de linux es poden llegir aquestes fifos com si de fitxers és tractessin amb les típiques funcions de C (read/write).

Ara ja se sap més com funciona la programació de tasques en Temps Real, com crear-les i com comunicar-les entre elles i amb programes de linux. En aquest moment ja es pot comprendre bé com s'han fet les modificacions al planificador i també s'està preparat per comprendre l'exemple

posat, per veure el rendiment de RTLinux, ja no només servirà per quedar-se amb els resultats de rendiments sinó que ara ja s'haurà de comprendre com s'han implementat les tasques.

Per acabar d'entendre millor la programació de RTLinux en el següent apartat es veuran més exemples de tasques en temps real. Aquests exemples també ajudaran a comprendre per que serveix un Sistema Operatiu en Temps Real .

Pel que fa a la programació de tasques en temps real es podria parlar de molts més aspectes però simplement s'ha fet un petit incís als coneixements bàsics d'aquest tipus de programació. Es podria parlar de senyals, interrupcions, semàfors, memòria dinàmica, etc.

3.3.7 Exemples

En aquest apartat es mostren dos exemples per comprendre una mica millor la programació de tasques en temps real i per veure el funcionament de RTLinux.

Exemple1

En el primer exemple es crea una tasca en temps real periòdica que executa un bucle. La tasca que executarà el bucle serà periòdica de 50000000 ns. Aquest fitxer es compilarà i un cop obtingut el fitxer objecte s'inserirà al kernel amb **insmod exemple1.o**.

Un cop inserit el mòdul es pot notar que el sistema va més lent, això és per que la tasca està demanant casi sempre la CPU i com que és en temps real, sempre li dona, en canvi linux no tindrà tant temps de CPU i per això es notarà la lentitud. Ara es pot modificar el número d'iteracions del bucle per augmentar el temps de còmput. Com més temps de còmput, més alentiment es notarà. Si s'augmenta el bucle fins a un temps de còmput de 50000000 ns (el mateix temps de període de la tasca en temps real), semblarà que el Sistema Operatiu s'ha penjat, però no és així, la tasca en Temps Real s'està executant amb un ús de la CPU del 100% ja que quan acaba un període, de immediat ha de tornar-se a executar i no pot deixar la CPU, per tant Linux no tindrà temps de CPU i no es podrà processar. Això produirà que sembli que el sistema s'ha bloquejat.

```
#include <rtl.h>
#include <rtl_sched.h>

pthread_t _thread;

  void fun() {
    int i,x,j,n;
    n=5000;
    j = 10;
    while(1){
      for (i=0; i<n; i++)
        for (x=1; x<j; x++);
  pthread_wait_np();
     }
  }
```

```
int init_module(void) {
  struct sched_param sched_param;

  pthread_attr_t attr;
  pthread_attr_init (&attr);
  sched_param.sched_priority = 1;
  pthread_attr_setschedparam (&attr, &sched_param);
  pthread_attr_setfp_np(&attr, 1);
  pthread_create(&_tank_thread, &attr,fun,0);
  return 0;
  }

  void rt_cleanup_module(void){
              pthread_delete_np (_thread);
  }
```

Exemple2

En l'exemple anterior s'ha vist la creació i gestió de tasques en temps real i com és té control total sobre els recursos hardware. Ara en aquest exemple es veurà com funciona la comunicació entre tasques i els mecanismes de fifos. Aquest exemple es bastant interessant ja que es comprova el que ofereixen els Sistemes Operatius de Temps Real, es comprovarà de manera auditiva. S'han implementat dos programes de lectura d'un fitxer de só (.au) un en temps real i un altre des de linux en no temps real. Un cop executats es comparen els resultats observats amb carrega al sistema i es veurà clarament un aspecte important de RTLinux.

El que fa l'aplicació es llegir dades de la fifo que s'escriuen des d'un procés linux, aquestes dades són el contingut d'un fitxer .au. El període de llegir de la FIFO i emetre el só és de 8192Hz. Per executar l'exemple es compilarà e inserirà el mòdul al kernel, aquest esperarà a que hi hagi dades a la fifo, des de una aplicació linux s'escriuran les dades a la fifo, per exemple amb la comanda cat.

El programa que reprodueix és el següent:

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#include <rtl_fifo.h>
#include <asm/io.h>
pthread_t tasca;
/*Filtre del só*/
int filtre(int x)
{
   static int oldx;
   int ret;
   if(x & 0x80) { x = 382 - x;  }
   ret = x > oldx;
   oldx = x;
   return ret;
}
```

```
void * sound(void *arg)
{
  char dat;
  char t;
  struct sched_param p;
  p.sched_priority=1;
  pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
  pthread_make_periodic_np (pthread_self(), gethrtime(),1000000000/8192LL); /* freq =
                                                                              8192 Hz */
  while(1)
  {
   pthread_wait_np ();
   if(rtf_get(0, &dat, 1) > 0)
   {
          dat = filter(dat);
          t = inb(0x61);
          t &= 0xfd;
          t |= (dat & 1) << 1;
          outb(t,0x61);
   }
  }
}

int init_module(void)
{
 rtf_create(0, 4000); /* crear fifo */
 /* preparar el speaker */
 outb_p(inb_p(0x61)|3, 0x61);
 outb_p(0xb0, 0x43);
 outb_p(3, 0x42);
 outb_p(00, 0x42);
 return pthread_create(&tasca, NULL,sound, 0);
}

void cleanup_module(void)
{
 pthread_delete_np(tasca);
 rtf_destroy(0);

}
```

Per comparar el temps real amb el que no és temps real, s'ha creat un exemple que fa exactament el mateix però des de una aplicació d'usuari (en no temps real.)

En aquest exemple s'haurà de demanar permís al Sistema Operatiu per accedir als ports del speaker ja que no s'està en zona de temps real i per tant no es té total accés lliure als recursos hardware. Per fer-ho s'utilitzarà la crida ioprem(). Una altre qüestió a veure és com generar la lectura i emissió del só a una freqüència de 8192Hz, la crida nanodelay() tant sols té una resolució del ordre dels milisegons pel que s'haurà de fer alguna cosa per obtenir 0,12 milisegons. Per fer-ho es crea una funció que gasti temps en còmput.

```
#include <unistd.h>
#include <asm/io.h>
#include <time.h>

static int filtre(int x)
{
        static int oldx;
        int ret;
        if(x & 0x80) x = 382 - x;
        ret = x > oldx;
        oldx = x;
        return ret;
}

void wait(int x)
{
        int i;
        for (i=0; i<x; i++);
}

void sound()
{
        char dat;
        char t;
        while (1)
        {
         if (read(0, &dat, 1) > 0)
         {
                dat = filter(data);
                t = inb(0x61);
                t &= 0xfd;
                t |= (dat & 1) << 1;
                outb(t,0x61);
         }
         wait(40000);
        }
}


int main(void)
{
        unsigned char dummy,x;
        ioperm(0x42, 0x3,1);
        ioperm(0x61, 0x1,1);
        dummy= inb(0x61);
        wait(10);
        outb(dummy|3, 0x61);
        outb(0xb0, 0x43);
        wait(10);
        outb(3, 0x42);
        wait(10);
        outb(00, 0x42);
        sound();
}
```

Si s'executen els dos programes es podrà comprovar com el primer (en temps real) pot complir els requeriments temporals amb càrrega al sistema ja que la reproducció del fitxer .au anirà bé, en canvi en el segon exemple si s'afegeix carrega al sistema la reproducció es veurà tallada (bloqueig de la tasca). Ni que el só sigui bastant dolent ja que s'està treballant sobre el speaker del PC que no aporta gens de qualitat, aquest exemple serveix per percebre una de les propietats importants d'aquests tipus de sistemes, els compliments dels terminis temporals imposats.

3.4 Preparació TCL/TK

3.4.1 Que és TCL/TK?

TCL/TK és un llenguatge de programació interpretat i multiplataforma. Llenguatge que va se creat per Jhon K. Ousterhout i un equip de la Universitat de California. Actualment els desenvolupadros del llenguatge són Sun Microsystems Laboratories. Aquest llenguatge és d'ús gratuït, per tant serà adient pels requisits del projecte.
Aquest llenguatge de programació es divideix en dos grups:

- TCL: Llenguatge de comandes, l'intèrpret d'aquest llenguatge de comandes és tclsh. Una de les grans avantatges d'aquest llenguatge és la facilitat amb que es poden crear noves extensions del llenguatge implementant funcions en C++ que passaran a ser noves instruccions de l'intèrpret. Alguna d'aquestes extensions són: BLT (representacions gràfiques en 2D), Itcl (Incremental tcl, TCL orientat a objecte), OraTcl (per manipular ORACLE). Una de les extensions més importants i populars de TCL és TK (Tool Kit) creada pel creador de TCL.

- TK: Proporciona un intèrpret denominat wish i que permet la creació d'interfícies gràfiques. TK doncs permet crear elements d'interfície gràfica anomenats widgets (botons, scrolls, llistes, etc.).

TK es distribueix juntament amb TCL en un paquet anomenat TCL/TK.
Tcl/Tk pot arribar a ser un llenguatge molt potent ja que com ja se sap, es gratuït i el programador pot estendre el número de funcions. D'aquesta manera amb l'ajuda de tota la comunitat de programadors es pot anar augmentant la complexitat del llenguatge. Aquesta és una de les avantatges dels llenguatges de codi lliure. Amb aquest llenguatge es poden crear diferents funcions/procediments per obtenir funcionalitats personalitzades i també es poden obtenir de altres programadors que formen part d'una comunitat pel desenvolupament de noves eines, funcionalitats, etc. pel llenguatge descrit.

3.4.2 Per que utilitzar TCL/TK?

Existeixen diferents raons que fan elegir TCL/TK per programar les aplicacions que es volen implementar pel projecte. El primer aspecte i el més important de tots es que el llenguatge és de codi lliure, en l'apartat anterior ja s'han comentat els avantatges que això comporta.

Aquest llenguatge permet crear interfícies gràfiques amigables per l'usuari per representar les dades del programa de forma senzilla. TCL/TK és un llenguatge bastant fàcil d'aprendre i entendre. Amb poques línies de codi es pot crear una aplicació amb una interfície gràfica complexa. Per tant, es disposa d'un llenguatge fàcil d'utilitzar i amb una gran potència.

Una altre raó és la facilitat en que es pot modificar les aplicacions creades en TCL/TK en posteriors versions. La seva alta facilitat de separació del codi computable i la interfície gràfica fa que es puguin fer modificacions de la GUI sense modificar l'estructura del flux de dades de l'aplicació.

Un avantatge important alhora d'utilitzar Tcl/Tk es que aquest serveix per mostrar les dades que genera un programa en C. Imaginar una aplicació en Tcl/Tk que genera la gràfica dels resultats de la temperatura d'un controlador donat per un programa en C. Si es canvia el programa en C per generar els valors d'un controlador, ara però de nivell, l'aplicació Tcl/Tk seguirà servint, simplement s'hauran de canviar alguns aspectes de presentació.

Per últim comentar que TCL/TK és un llenguatge multiplataforma, per tant es poden programar les aplicacions ja sigui des de Linux, Windows o altres Sistemes Operatius. El llenguatge actua com una màquina virtual on es poden executar les aplicacions sobre qualsevol plataforma. Això dóna gran avantatge davant altres llenguatges.

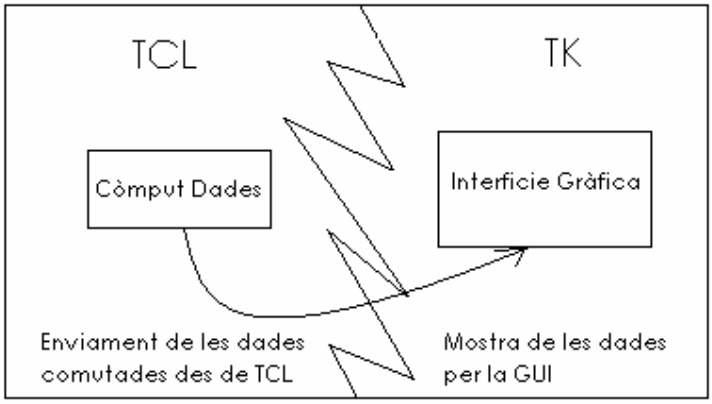


Figura 3.5: *Arquitectura TCL/TK*

3.4.3 Instal·lació

Per instal·lar TCL/TK simplement s'ha d'obtenir la distribució actualitzada de Tcl/Tk, ja sigui per Windows, Linux, Unix o qualsevol sistema compatible per Tcl/Tk. Es pot buscar la distribució per Internet. En el cas del projecte, com que s'està realitzant amb RTLinux, es

baixarà la versió per Linux, serà un fitxer d'extensió .rpm. Els fitxers RPM són paquets que contenen l'aplicació, els documents d'aquesta i les fonts de l'aplicació.
Un cop obtingut el fitxer RPM s'instal·larà amb la comanda **rpm –i fitxer.rpm.**
Un cop instal·lat es podrà accedir als intèrprets de tcl i tk. Executant la comanda **tclsh** s'accedeix al intèrpret de tcl i executant la comanda **wish** s'accedeix al intèrpret de tk.

3.4.4 VTCL eina de suport

VTCL és una eina de desenvolupament d'aplicacions visuals d'alta qualitat. Aquesta eina és totalment lliure i es pot trobar per diferents plataformes tal com Windows, MAC, UNIX, etc.
Si la programació de TCL/TK ja es senzilla, aquesta eina encara facilita més el treball i dóna més complexitat i potència a les aplicacions que es vulguin crear amb el mínim temps possible.
Un aspecte molt important de VTCL és que dóna la possibilitat de provar l'aplicació abans d'utilitzar-la amb un botó de test. Un cop conforme amb els resultats només s'ha de guardar i ja estarà llest per ser usat. L'únic requeriment que té VTCL és que com és normal, necessita tenir instal·lat el llenguatge TCL/TK. Aquesta eina està en continu desenvolupament, per tant, cada cop surten versions més avançades i millorades.
Per tant, el llenguatge TCL/TK juntament amb l'eina VTCL és una gran aposta per desenvolupar les aplicacions necessàries de RTLinux ja que permet fer-ho de forma gratuïta, fàcil i obtenint software de qualitat i amb gran potència.

3.4.5 Exemple d'aplicació: Diagrama de temps

Per posar a prova les possibilitats de Tcl/Tk es crearà una aplicació, aquesta sense utilitzar l'eina de programació visual VTCL. Aquesta aplicació consistirà en representar de forma visual la planificació de les tasques en temps real de RTLinux. Aquesta aplicació mostrarà l'execució de les tasques en temps real més la tasca de linux a través del temps. L'aplicació mostrarà els canvis de tasques en la CPU. Aquesta aplicació és la primera que s'ha fet utilitzant Tcl/Tk en el projecte. A estat una manera de comprovar la viabilitat d'utilitzar el llenguatge i veure'n la potència de programació que ofereix.

3.4.5.1 Concepte

Un cop se saben les possibilitats de Tcl/Tk i s'ha escollit com eina de programació de les aplicacions ara es passa a comprovar la seva potència i veure de forma pràctica si l'elecció a estat correcta. L'aplicació que es crea és un generador de diagrama de temps el qual mostra l'execució de les tasques en Temps Real de RTLinux. Aquesta aplicació

serveix per dues coses, la primera ja comentada per veure'n la potència del llenguatge, i la segona raó per poder entendre millor que està passant quan s'executen tasques en temps real.

Aquesta eina es de gran utilitat i s'ha implementat en dos períodes. En el primer període s'ha fet la funcionalitat bàsica, mostrar les tasques com s'executen en el temps, en aquesta primera versió no s'ha fet a escala ni s'han afegit cap mena d'informació al diagrama, simplement surt el flux d'execució. Aquesta primera versió serveix per comprovar la viabilitat de Tcl/Tk en crear aplicacions amigables per l'usuari. En el segon període es millora l'aplicació afegint informació de tot tipus de les tasques e implementant una funció de zoom per millorar-ne la interacció amb l'usuari alhora de veure l'execució de les tasques. Aquesta última versió s'afegirà a l'aplicació que s'explicarà en el següent capítol, el generador de tasques.

3.4.5.2 Disseny

El disseny de l'aplicació es senzill. El que fa l'aplicació és llegir les dades del fitxer monitor.txt. Les dades que conté aquest fitxer són les dades de la planificació de les tasques en RTLinux.

Com ja s'ha explicat anteriorment, es va modificar el fitxer de planificació de tasques en temps real (rtl_sched.c) per tal d'obtenir les dades de planificació de les tasques. Aquestes dades s'escrivien a una rt_fifo i una aplicació en no temps real en recollia les dades i les emmagatzemava en un fitxer, aquest fitxer es el monitor.txt. L'aplicació que es passa a dissenyar també s'executarà en entorn de no temps real. En la següent il·lustració es mostra l'arquitectura de l'aplicació.

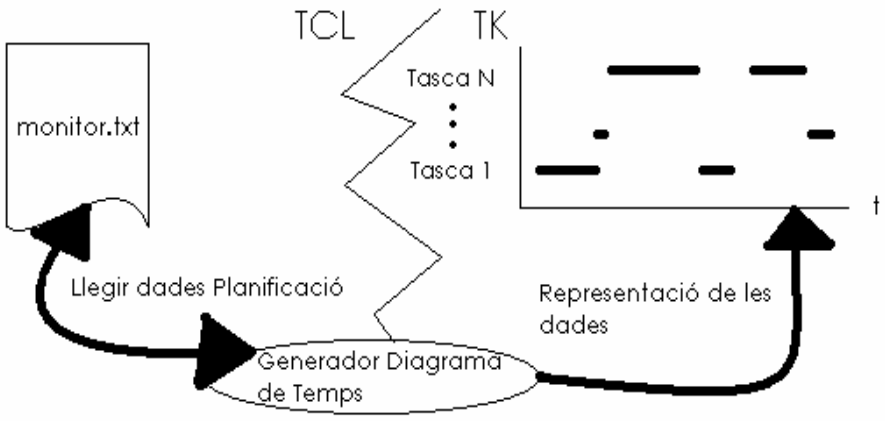


Figura 3.6: *Arquitectura aplicació Diagrama de Temps*

Notar que l'arquitectura segueix la base de TCL/TK, hi ha dos parts ben diferenciades, la part de TCL (còmput i manipulació dades) i TK (representació visual de les dades).

Es pot treure per exemple la part de TK i modificar-la totalment sense haver de modificar cap aspecte de TCL. Aquesta característica ja s'ha comentat però ara es veu l'ús pràctic. Per entendre'n l'arquitectura de l'aplicació s'explica dividint ens els dos grups, TCL i TK.

- Part TCL: Llegeix les dades del fitxer monitor.txt i introdueix les dades en unes estructures de TCL per la posterior visualització. Aquestes estructures són manipulades per obtenir dades que podrà utilitzar la part de TK, aquestes dades són del tipus, temps mínim d'execució, temps màxim d'execució, número de tasques, etc.. Notar doncs que aquest mòdul es típic d'obtenció i manipulació de dades.
- Part TK: Aquest mòdul el que fa es representar les dades que a generat el mòdul TCL. Aquest crea la interfície de l'aplicació i en gràfica el diagrama de temps.

3.4.5.3 Implementació

Un cop es té clar el disseny de l'aplicació, s'entrarà en més detall en cada part de l'aplicació. Seguidament s'expliquen cadascun dels dos mòduls (Tcl i TK) de l'aplicació a detall d'implementació.

- Part TCL: Aquest mòdul el que fa es llegir del fitxer monitor.txt les dades escrites pel planificador en temps real. Aquestes dades les posa en una variable. El format del fitxer monitor.txt és del tipus:

```
<ID Tasca1> - <Temps en execució>
<ID Tasca2> - <Temps en execució>
……..
```

El que farà l'aplicació serà crear una estructura en forma de llista de llistes de dos posicions cada subllista. Cada posició de la llista contindrà la ID de la tasca i el seu temps d'execució, [[ID1,T1],[ID2,T2],ID1,T3]]. Aquest mòdul també obtindrà unes dades importants pel mòdul TK. Obtindrà una altre llista on contindrà les ID de les tasques que entren en execució per saber quines són i quantes [ID1,ID2,ID3]. També obtindrà una variable amb el temps total de la simulació, aquesta variable s'obtindrà a partir de la suma dels temps de la llista de llistes. Per últim, obtindrà una dada important per l'escalat de la gràfica, la variable $AUX. Aquesta variable inicialment serà la divisió entre el valor màxim de temps i el valor mínim de temps. Aquesta variable des del mòdul TK es podrà modificar

mitjançant un botó, això es veurà a continuació. Seguidament es mostra el codi de la part TCL comentat.

```
## Llegim dades del fitxer
set fp [open "monitor2.txt" r]
set data [read $fp]
close $fp

## llista de tasques
set lista ()
## data conté la ID i el Temps de cada línia
set data [split $data "\n"]
## Variable que contindrà el valor total de temps de simulació
## totalTime = Suma(Temps d'execució)
set totalTime 0
set aux 0
set pt 0
## Variables per obtenir el màxim i mínim de temps per calcular
## la variable AUX d'escalat
set min 999999999
set max 0

## Per cada linea del fitxer
 foreach line $data {
        ## Separem el ID i temps de cada línia
        set data1 [split $line " - "]
        ## Mirem si ja existeix la ID en la llista
        set var [lsearch $lista [lindex $data1 0]]
        if { $var == -1 } {
                ## Afegim la ID a la llista de IDs
                set lista [linsert $lista [llength $lista] [lindex $data1 0]]
                set b [lindex $data1 3]
                set totalTime [expr $b + $totalTime]
        }
        ## Obtenim Temps i l'acomularem a la variable de Temps total
        set b [lindex $data1 3]
        set totalTime [expr $b + $totalTime]
        puts $totalTime

        ## En aquesta part s'obté el valor mínim i màxim de Temps
        if { $min > $b } {
                if { $b > 0 } {
                        set min $b
                }
        }
        if { $max < $b } {
                set max $b
        }
        puts "-->$min"
        puts "-->$max"

## Calculem el valor d'escalat
set AUX [expr $max/$min]
```

- Part TK: La part TK el que farà es agafar les dades generades per TCL i mostrar-les a través d'una interfície. Crearà una finestra que contindrà un Canvas amb possibilitat de moure'l a partir d'una barra de desplaçament. Un Canvas és un control que està limitat per una regió concreta i que s'hi pot dibuixar el que es vulgui. Dins d'aquest Canvas hi haurà el diagrama amb l'execució de les tasques. En la part inferior del Canvas s'hi trobarà un boto el qual permetrà allunyar o apropar el canvas, es a dir, canviarà el valor d'escalat. Seguidament es mostra el codi de la part Tk.

```
wm title . "Simulació de les tasques creades"

## Crearem el Canvas amb el scrollbar amb la funció ScrollTexto
ScrollTexto .p $totalTime $AUX -width 900 -height 600
        .p.texto xview moveto 0.0

## Afegim el botons de Zoom apropar i allunyar
button .in -text ZomIn -width 30 -command {set AUX [expr $AUX + $min];
button .out -text ZomOut -width 30 -command {set AUX [expr $AUX - $min];
## Al premer un dels botons, cridarem la funció Zoom que redibuixarà
## el diagrama amb l'escalat modificat

## ho afegim a la finestra
pack  .in .out -side top

## Cirdem la funció Zoom per graficar tasques amb les dades obtingudes
## des del mòdul de TCL

Zoom $AUX $data $line $lista $totalTime $pt 520 $aux
```

El codi font complet de l'aplicació es pot trobar a [4]l'annex on esta totalment comentat. Aquí tant sols es posen les parts importants per entendre l'aplicació i per entendre les facilitats que aportarà TCL/TK a les aplicacions.

Un cop vist el disseny i la implementació de l'aplicació es pot veure com realment aquest llenguatge es bastant comprensiu ja que llegint-lo es pot entendre bastant el que s'està fent. El llenguatge és bastant potent perquè permet fer coses amb un grau de complexitat alta amb poques línies de codi i aporta abstracció entre la interfície gràfica i la manipulació de dades. L'únic aspecte que no s'ha comprovat és el de multiplataforma però en el cas d'aquest projecte no es de massa importància ja que el que es necessita es desenvolupar eines pel Sistema Operatiu de Temps Real RTLinux.

[4] El codi font es pot trobar en el punt 7.1 de l'annex.

3.4.5.4 Exemple d'execució

Per finalitzar el tema de TCL/TK tant sols queda veure algun exemple d'execució de l'aplicació de generació de diagrama de temps de les tasques en temps real. En aquest punt es mostrarà també l'aspecte de les aplicacions creades amb Tcl/Tk.

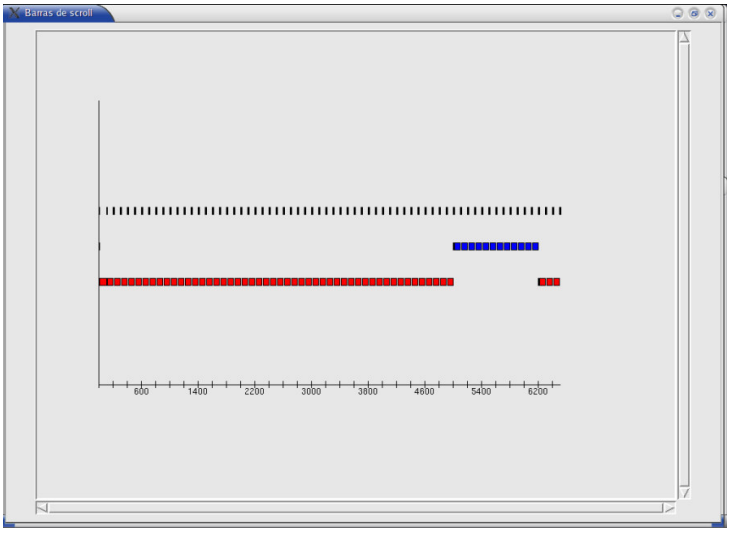


Figura 3.7: *Exemple execució versió 1*

En la il·lustració anterior es mostra una imatge de la primera versió que es comentava anteriorment. Notar que no es dona massa informació i l'escalat s'ha fet a ma modificant el codi. S'ha buscat el valor òptim per veure el resultat de la millor manera. Notar com està mostrant l'execució de tres tasques. Las tasca inferior es pot observar que es la de Linux ja que és la que s'està executant en background, és a dir, quan no hi ha cap tasca en execució. Després hi ha dues tasques en temps real. La mes superior es pot concloure que s'executa amb un període mes curt per això la seva freqüència es major i es pot veure que de ben segur que és més prioritaria que la tasca en temps real que es troba al mig ja que quan aquesta del mig intenta executar-se es va bloquejant per causa de la tasca superior que demana més freqüentment l'ús de la CPU.

En la següent il·lustració es mostra un exemple d'execució de la segona versió de l'aplicació, en aquesta es pot veure com es mostren més dades i s'han definit els eixos. La barra de desplaçament s'ha modificat en aquesta versió per fer més ràpid l'accés a tota la gràfica ja que en l'anterior versió no es podia desplaçar arrestant-la. A l'inferior de la finestra es pot veure dos botons que s'han afegit per fer de manera manual l'escalat del diagrama i no haver de modificar el codi. Són dos els botons que hi ha, un per allunyar i l'altre per apropar. Ara l'usuari es pot moure tant en el temps com en

l'escalat i així poder analitzar de forma bastant còmoda l'execució de les tasques a través del planificador en temps real de RTLinux.

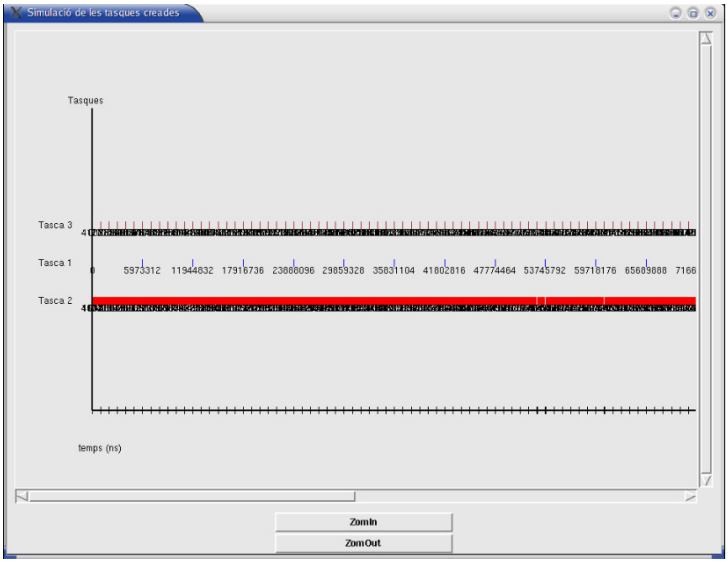
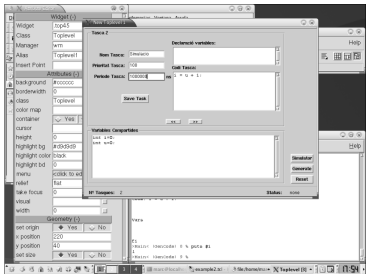


Figura 3.8: *Exemple execució versió 2*

En l'exemple es pot veure com s'ha allunyat molt per veure'n una vista general de l'execució. Les línies negres que es veuen són la informació de temps. Aquesta informació de temps es mostra per a cada tasca. En cada tasca el temps es mostra quan hi ha un canvi d'execució a no execució. Al allunyar molt la vista, els resultats es solapen entre ells, en futures versions es podria millorar i mostrar menys quantitat de dades quan mes s'allunyi la vista.

Notar doncs que es possible de forma relativament fàcil crear aplicacions sobre RTLinux per donar un entorn més fàcil d'entendre i flexible per l'usuari. Des de que s'ha instal·lat el RTLinux i no es sabia que passava amb l'execució de les tasques fins en aquest moment el salt ja es prou gran. Quan es tenia modificat el planificador i s'obtenia un fitxer de text, aquest era casi incomprensible per l'usuari però ara es disposa d'una interfície que ajuda a entendre l'execució de les tasques. No s'han necessitat masses línies de codi ni aprendre a programa perfectament amb TCL/TK. Amb més experiència amb programació TCL/TK és poden arribar a fer aplicacions bastant interessants i completes.



4

Generador de Tasques en Temps Real

4.1 Previ

En aquest capítol es posa en marxa els conceptes obtinguts durant el projecte a través de les eines instal·lades, configurades i apreses a manipular. En aquest punt l'objectiu serà cercar un problema real que faci referència al tema del projecte. Algun problema que ajudi a veure si es pot acomplir l'objectiu proposat durant tot el projecte, obtenir un Sistema Operatiu en Temps Real per a un PC i poder-ne crear aplicacions per facilitar la feina de l'usuari final, tot utilitzant software lliure.

Un cop arribat aquí ja s'han assimilat els conceptes de temps real, sistemes operatius en temps real, etc. assimilats en capítols anteriors. També s'han escollit les eines a utilitzar, s'han instal·lat i estan llestes per utilitzar-les. L'últim pas ha estat introduir una mica en l'ús d'aquestes eines, Sistemes Operatius, llenguatges de programació. Ara queda proposar un problema real.

Aquest problema s'haurà d'analitzar i veure com és pot afrontar utilitzant tot el que s'ha après. Es dissenyarà l'aplicació que solucionarà el problema plantejat i un cop llest el disseny es passarà a implementar l'aplicació amb el llenguatge elegit en el capítol anterior. Si tot ha funcionat correctament es passarà a fer proves per veure el funcionament i els resultats.

En aquest tema s'està posant en marxa tots els coneixements assimilats a base d'estudiar teoria, fer proves, fer tests, analitzar exemples, etc. I en el següent capítol es veuran els resultats i es podrà començar a treure conclusions sobre l'objectiu principal del projecte.

4.2 Plantejament del Problema i Solució

Ara es té un Sistema Operatiu en Temps Real de codi lliure, es pot modificar, és més, s'ha modificat com s'ha volgut, gran avantatge del codi lliure. Es disposa d'eines de codi lliure que també es poden modificar i ampliar comes vulgui. Es té tot preparat per posar-ho en marxa. Aquí s'ha arribat a complir mig objectiu, ara toca posar-ho a prova. Per fer això s'ha d'analitzar un problema real que és podria trobar l'usuari final i solucionar-lo creant una aplicació. Fent això es podrà acabar de complir l'objectiu del projecte. Si s'acompleix és podrien arribar a pensar multitud d'eines per l'usuari del Sistema Operatiu en Temps Real i fins i tot modificacions a aquest Sistema Operatiu tot gràcies a utilitzar software lliure.
Des del principi de tot el projecte es podria haver escollit el camí d'utilitzar software privat i pagar totes les llicencies i de ben segur que s'obtindria un Sistema Operatiu de Temps Real amb un rendiment òptim com ja s'ha vist en la descripció de Sistemes Operatiu comercials i a mes a mes també es disposaria de tota classe de software potent. No tot són avantatges però, el preu a pagar de ben segur que seria molt alt i no es podrien fer les modificacions que es volguessin ja que seria un sistema protegit i tancat.

Tornant al punt actual, s'ha de pensar una problema real que pot tenir un usuari final del Sistema Operatiu en Temps Real i com es podrà solucionar amb les eines que es disposa. En el següent subapartat es planteja el problema alhora de crear les tasques amb temps real.

4.2.1 Problema

Potser un dels problemes més importants i recents que es pot trobar un usuari del Sistema Operatiu en Temps Real serà alhora de crear les tasques en temps real. Aquest és el primer problema que és poden trobar amb el primer contacte amb el sistema. L'objectiu principal del Sistema Operatiu serà poder crear algunes tasques, cadascuna amb una finalitat, i que aquestes siguin en temps real. De moment no es pot pretendre que l'usuari no tingui cap mena de noció en programació, potser en treballs futurs si. El perfil de l'usuari del sistema hauria de ser d'un programador amb coneixements de C. No necessàriament necessitaria coneixements de Sistemes Operatius ni cap de les coses estudiades anteriorment.

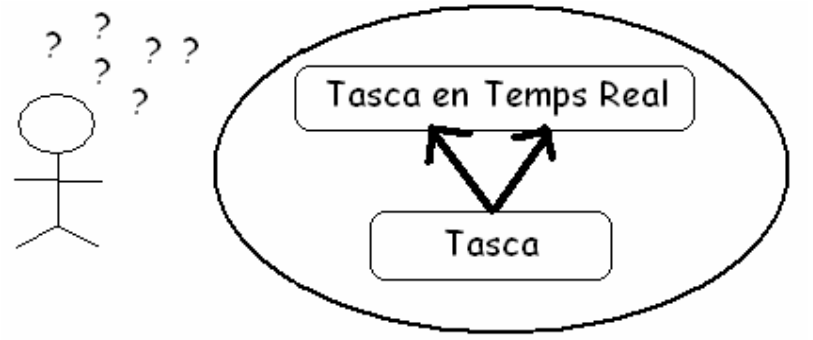


Figura 4.1: *Problema: creació tasques en TR*

Imaginar la situació, un programador es trobar amb un Sistema com aquest i sap que és de temps real i també sap que necessita aquesta característica.
Aquest programador no té nocions de programació de tasques en temps real i no pot perdre el temps en estudiar com funciona. Necessitem crear una eina de suport per a que el programador pugui crear tasques en temps real. Ell sap que vol que facin les tasques i la funcionalitat d'aquestes les té implementades però les vol inserir en el temps real però no sap com fer-ho. L'aplicació que s'ha de crear ha de fer exactament això, implementar la funcionalitat de les tasques i posar-les en marxa.

4.2.2 Solució

La solució al problema plantejat serà la creació d'una aplicació que generi tasques en temps real a partir del codi funcional de cada tasca que aportarà l'usuari. Aquesta aplicació generarà el codi font de les tasques en temps real. L'usuari tindrà l'opció d'anar afegint tasques adjuntant el codi de la funcionalitat de la tasca i finalment podrà demanar de generar-les i posar-les en funcionament. Aprofitant eines ja implementades, l'aplicació permetrà veure'n la simulació de l'execució de les tasques i així aquest podrà analitzar si es el que necessita el seu sistema.

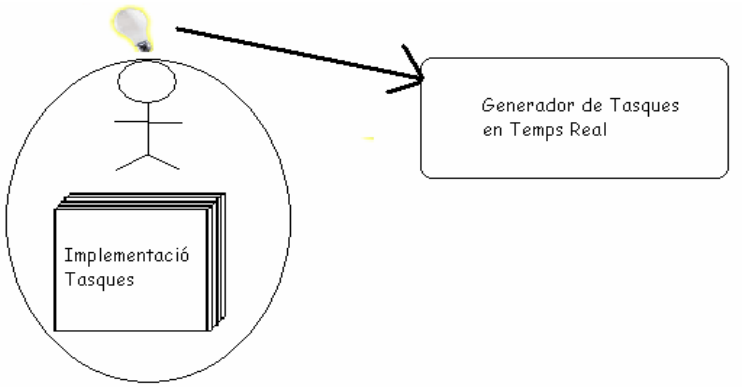


Figura 4.2: *Solució: creació tasques en TR*

Ara ja se sap que la solució al problema serà crear una utilitat per generar automàticament les tasques en temps real, fer això no serà trivial. Una proposta de projecte podria ser aquesta, dissenyar una aplicació complerta i potent de generació de tasques en temps real sobre el Sistema Operatiu RTLinux. Seguint tots els passos de disseny d'una aplicació utilitzant totes les tècniques apresses d'enginyeria del software. Aquest projecte seria l'anterior al projecte comentat. Primer es necessitaria fer un estudi de la viabilitat d'obtenir un Sistema Opreatiu en Temps Real i crear-ne software complex i potent. Per tant aquest projecte pot servir a algun lector que es vulgui aventurar a seguir en aquest tema després de llegir-ne aquesta memòria.

Bé, doncs l'aplicació haurà de ser capaç de generar tasques en temps real amb una cert grau d'abstracció. Aquesta eina es desenvoluparà amb el llenguatge ja utilitzat Tcl/Tk i amb l'ajuda de comandes del propi Sistema Operatiu RTLinux. Haurà de proporcionar una interfície gràfica fàcil d'utilitzar i amigable per l'usuari i amb un cert toc de professionalitat.

Notar doncs que amb el desenvolupament d'aquesta aplicació i els exemples executats posteriorment s'abarcaran tots els temes explicats al principi d'aquesta memòria.

Seguidament es presenta el disseny de l'aplicació per la seva posterior implementació.

4.3 Disseny de l'aplicació

En aquest punt es proposa la discussió d'alguns aspectes del disseny de l'aplicació a realitzar. Recordar que l'aplicació generarà tasques en temps real a partir de les funcionalitats de cada tasca.

- L'usuari introduirà la informació de cada tasca i aquesta haurà de generar-ne el codi i seguidament compilar i executar.
- Se li permetrà a l'usuari especificar la funcionalitat de les tasques, les prioritats i amb el període que es vol executar.
- L'usuari podrà definir variables compartides entre les diferents tasques i a més podrà decidir si vol que l'aplicació creï real time fifos.

Notar com l'aplicació serà ja bastant complerta i permetrà fer bastants coses, en el pròxim capítol es veuran una sèrie d'exemples de control amb temps real, aquests exemples s'han fet utilitzant aquesta aplicació, alguns dels exemples generats amb aquesta aplicació són de la complexitat del control del nivell i temperatura d'un tanc d'aigua, un control que segur que si s'ha d'implementar sense cap ajuda, tant sols programant, es tardaria bastant més en fer-ho per no dir que passaria si el programador no sabes programa tasques en temps real. Serà una aplicació de gran ajuda i per tant s'ha de fer un bon disseny.

El disseny es pot definir en dues etapes, la primera etapa serà el disseny de la interfície gràfica, com es vol presentar a l'usuari l'eina. Com l'usuari introduirà les tasques i les seves propietats a l'aplicació. Com es presentaran les dades introduïdes, com podrà analitzar l'execució final, quins aspectes es deixaran a l'usuari definir (variables compartides, real time fifos, etc).

La segona etapa serà la problemàtica de com generar el codi a partir de les dades introduïdes per l'usuari i l'execució d'aquestes tasques.

4.3.1 Disseny GUI

Disseny de la interfície d'usuari serà la primera etapa de disseny. En aquesta etapa es crearà una interfície simple i coherent, una interfície fàcil d'utilitzar per l'usuari i a la vegada potent. Per desenvolupar aquesta etapa, es farà pensant en el llenguatge Tcl/Tk més concretament amb l'eina de programació visual TCL. S'haurà de donar la possibilitat a l'usuari d'introduir les tasques en temps real de forma fàcil i consistent.
L'aplicació consistirà amb una finestra en la qual hi haurà un formulari on es podrà introduir informació per cada tasca, la informació a introduir per tasca serà:

- Prioritat de la tasca: serà una caixa de text on es podrà introduir la prioritat amb que la tasca s'haurà d'executar, el tipus de dada a introduir serà un número. Com més gran sigui la prioritat més possibilitats tindrà d'executar-se davant d'altres.
- Nom de la tasca: igual que l'anterior, serà una caixa de text on s'introduirà el nom de la tasca, així es tindrà una manera d'identificar cada tasca.
- Període de la tasca: caixa de text on s'introduirà el temps amb que la tasca es repetirà l'execució, aquest temps s'especifica amb nanosegons.
- Declaració variables: una caixa de text multilínia on s'introduirà la declaració de les variables del codi que implementarà la funcionalitat de la tasca.
- Codi tasca: per últim hi haurà una altre caixa de text multilínia on s'introduirà el codi de la implementació de la funcionalitat de la tasca.

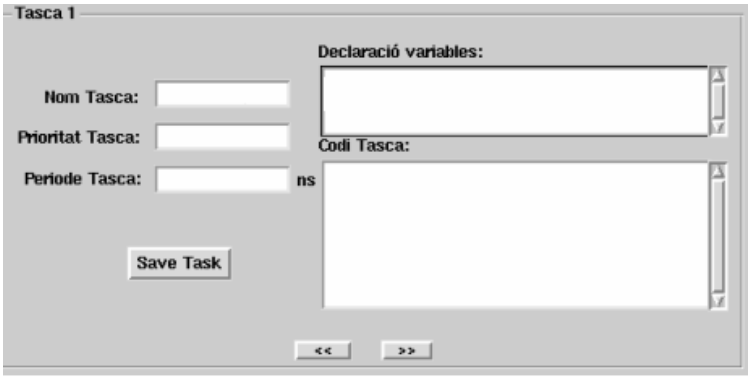


Figura 4.3: *GUI entrada dades de les tasques*

A part de donar la possibilitat d'introduir les dades de cada tasca, aquestes potser tindran la necessitat d'usar variables compartides entre elles.

Per donar aquesta possibilitat, s'introdueix una altre caixa de text multilínia en un altre requadre, general per a totes les tasques, en aquest requadre es podran declarar totes aquelles variables que l'usuari vol que siguin accessibles per més d'una tasca.

Una altre opció afegida en la interfície, és la possibilitat d'escollir si es vol crear una real time fifo o dues [5](N). Això es representa mitjançant uns *"check buttons"*. Es dóna la possibilitat d'escollir una, dos o cap real time fifo. Aquesta opció es pot necessitar quan es volen comunicar alguna de les tasques introduïdes amb alguna altre tasca.

Figura 4.4: *GUI entrada variables compartides i Real Time FIFOS*

Per últim, una altre dada a introduir mitjançant la GUI serà els includes del programa que seran necessaris. Com ja se sap, RTLinux no sempre necessita tenir tots els mòduls carregats, si en unes tasques no es necessari l'ús de real time fifos, no s'afegirà la definició dels includes per aquest mòdul. En versions futures es podria fer que el propi programa reconeixes automàticament la necessitat dels includes, això ja es comentarà en el capítol de propostes de futur. Per introduir els includes disposem d'un botó, quan premem aquest s'obrirà una finestra amb una caixa de text on es podran editar els includes que siguin necessaris.

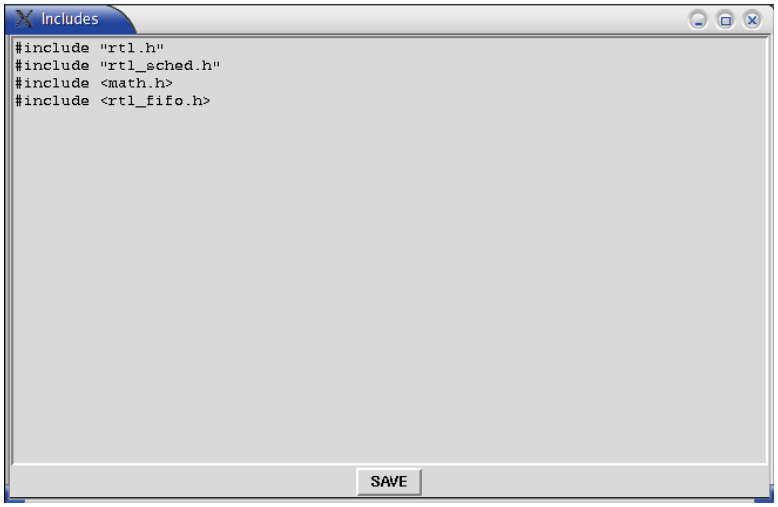


Figura 4.5: *GUI per definir includes*

[5] En l'aplicació s'ha fet per a dos rt-fifos ja que serà suficient, però podria fer-se per a N rt-fifios.

4.3.2 Generació i Execució del codi

Un cop introduïdes les dades en l'aplicació mitjançant la GUI dissenyada, el que fa falta és generar i executar-ne el codi. Un mòdul de temps real està organitzat d'una certa forma, el generador de tasques aprofita aquesta característica per crear el fitxer de codi font que posteriorment en generarà l'objecte. L'estructura del mòdul en temps real consta de una funció d'inicialització, una funció d'acabament i les funcions d'implementació de les tasques en temps real.

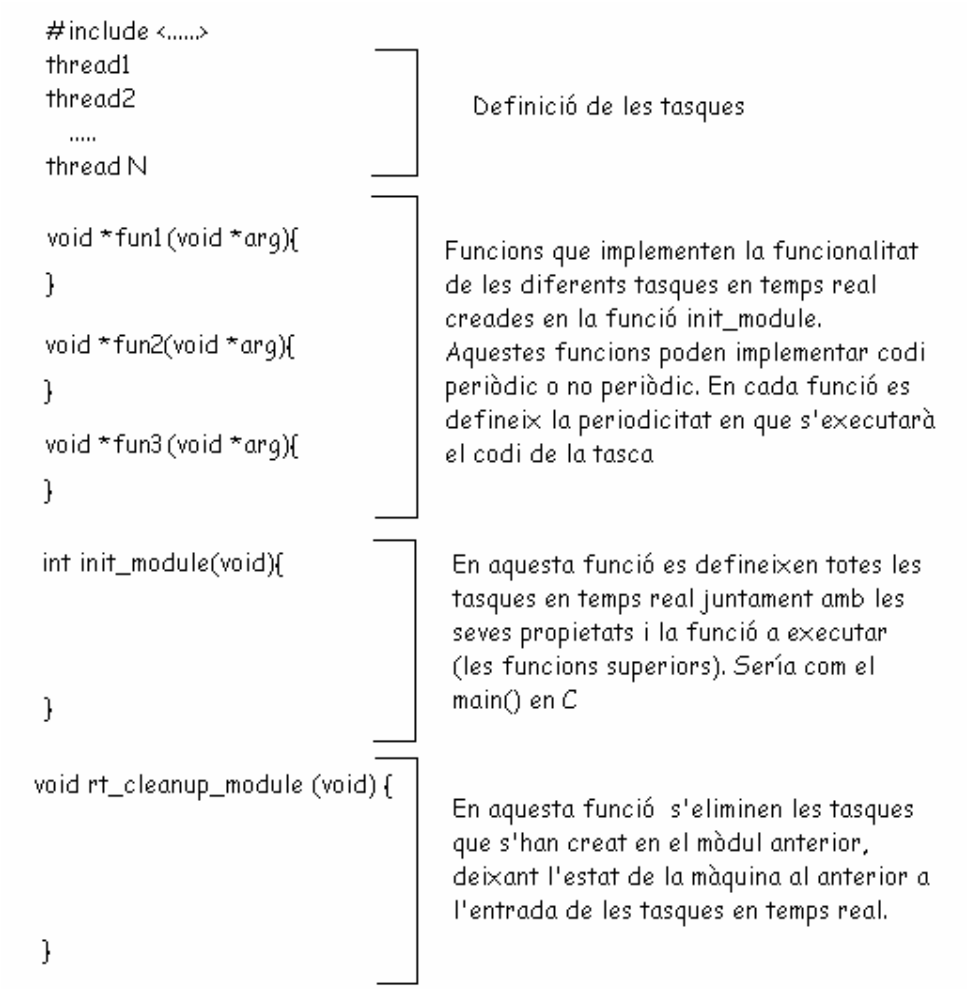


Figura 4.6: *Estructura del codi d'un mòdul TR*

Inicialment en el codi es troba la definició dels diferents includes que seran necessaris, seguidament hi ha la definició de les tasques en temps real que s'executaran.

Després ve la sèrie de funcions que són la implementació de les tasques en temps real, cada funció en defineix la funcionalitat de la tasca. Si la funció es periòdica es defineix la prioritat en que s'executarà aquesta funció. La definició de la periodicitat es fa en nanosegons. Després de

totes les funcions implementades queda la creació dels dos mòduls generals, el d'inicialització i el de finalització.

La funció d'inicialització crea les tasques en temps real, associa la tasca a una certa funcionalitat i en defineix les característiques de cada tasca. En canvi, la funció de finalització es finalitzarà l'execució de les tasques en temps real retornant al estat inicial.

Per generar tot aquest codi, el programa utilitza totes les dades introduïdes per la GUI, com s'explica en l'apartat anterior, aquestes dades són les que varien, en canvi l'estructura del codi es invariant i segueix un patró definit en unes variables definides. Per veure més detall, veure l'apartat següent d'implementació del Generador de Tasques.

Un cop generat el codi, es compila i executa durant un cert temps. Per fer això s'utilitza un fitxer Makefile com el mostrat seguidament:

```
all: code.o

include /usr/src/rtlinux/rtl.mk
clean:
	rm -f *.o

test: all
	#-rmmod sound
	#-rmmod rt_process
	#-rmmod frank_module
	(cd /usr/src/rtlinux/; scripts/rmrtl)
	(cd /usr/src/rtlinux/; scripts/insrtl)
	@insmod code.o
	sleep 1
	@rmmod code
	./monitor&
	sleep 2
	sh killmonitor.sh
#include $(RTL_DIR)/Rules.make
```

En el Makefile, es compila el codi del fitxer generat, s'obté l'objecte .o. Aleshores s'insereix en el kernel durant un temps, en aquest cas un segon. Un cop passat el segon, es treu el mòdul i s'executa un programa de monitorització de la planificació de les tasques que escriu la simulació de l'execució de les tasques en un fitxer de text per després mostrar la simulació.

Per tant, en l'aplicació es disposa d'un boto per generar i executar el codi i un altre boto per simular l'execució, aquest últim no es pot prémer fins que no s'ha generat i executat el codi.

4.4 Implementació de l'aplicació

En aquest punt s'explica a detall d'implementació l'aplicació de generació de tasques. La implementació també es divideix en els dos grans grups, la implementació de la GUI i la implementació de la generació de les tasques.

4.4.1 Implementació GUI

Per implementar la part de la interfície gràfica s'utilitza el software de codi lliure visual TCL per facilitar la tasca de creació d'interfícies. Les dades necessàries per la generació automàtica de les tasques en temps real són les següents:

- Variables compartides: són les variables globals del fitxer, que s'utilitzen en les diferents funcions.
- Includes: són els fitxers de capçaleres necessaris per la correcta compilació del codi generat.
- RT-fifos: Creació de fifos en temps real per utilitzar-les en alguna de les funcions de les tasques.
- Nom Tasca (per cada tasca): nom de la tasca a generar.
- Prioritat Tasca (per cada tasca): prioritat en que s'executa la tasca en el planificador de temps real.
- Període de la Tasca (per cada tasca): període en que s'executa repetidament la funcionalitat de la tasca.
- Variables de la Tasca (per cada tasca): definició de les variables utilitzades en la implementació de la funcionalitat de la tasca
- Codi de la Tasca (per cada tasca): codi d'implementació funcionalitat de la tasca.

Les variables referents a cada tasca, nom, prioritat, període, variables i codi s'emmagatzemen en una estructura. Aquesta s'insereix en una llista la qual contindrà tota la informació de totes les tasques inserides. Cada cop que es prem el botó guardar s'emmagatzema la informació de la tasca en la llista de tasques. Seguidament es mostra un petit esbós de codi on es mostra aquesta estructura.

```
##Codi source de la tasca
set code [Text1 get 0.0 end]
#Declaració variables
set vars [Text2 get 0.0 end]
#prioritat de la tasca
set prioTasc $txtPrio
#periode de la tasca
set periodTasc $txtPeriode
#Nom de la tasca
set nameTasc $txtName
```

```
##############################################
#llista on tenim els 4 paràmetres anteriors
#llista :     -periode tasca
#            -id tasca
#            -prioritat tasca
#            -code
#            -Nom
##############################################
set dada "$prioTasc:$periodTasc:$code:$vars:$nameTasc"
set llista [split $dada :]

#Definim un array on contindrem totes les dades de les diferents tasques
set ArrayTasc($i) $llista
```

L'altre informació, variables compartides, includes, rt-fifos, s'emmagatzemen en variables individuals. Aquesta informació es guarda en les variables quan es prem el botó de generar codi. La GUI implementada facilita la inserció d'aquesta informació en l'aplicació i la seva manipulació.

Per facilitar la comprensió de com s'ha implementat la interfície gràfica s'il·lustra la interfície amb la seva funcionalitat.

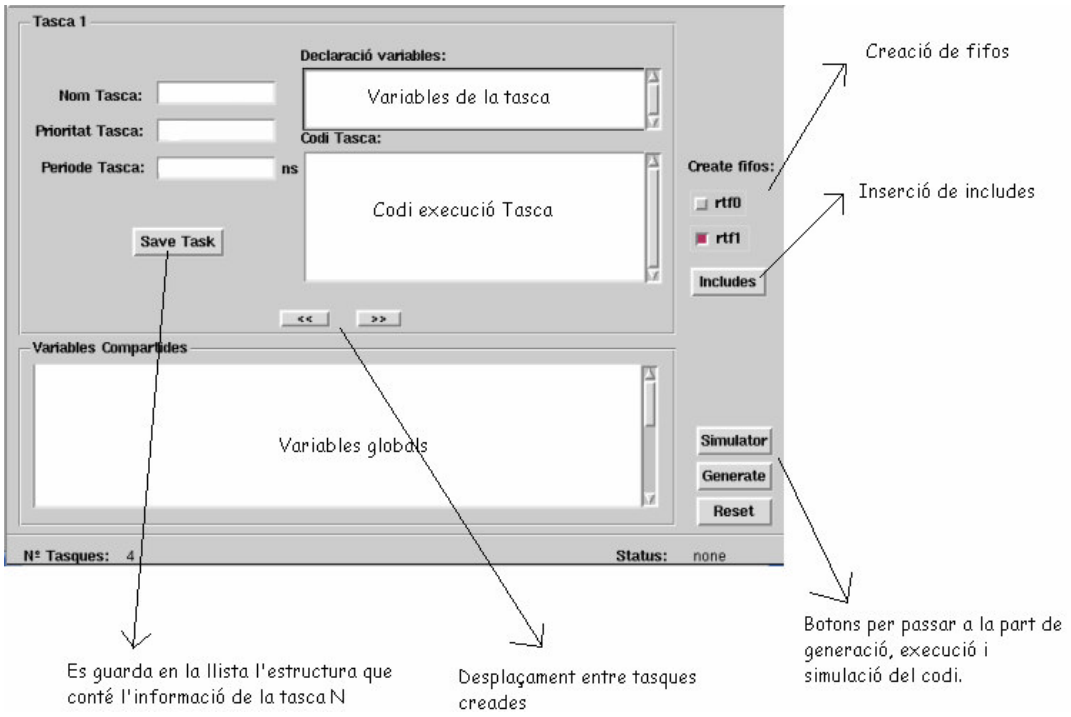


Figura 4.7: *Implementació de la GUI*

- Botó "Save Task": Aquest botó insereix la informació de la tasca en una estructura que anirà a parar a una llista de tasques. Si ja està creada la tasca el que fa es guardar els canvis en l'estructura de la tasca.

- Botons desplaçament: Aquests dos botons permet desplaçar-se en la llista de tasques. Dóna la possibilitat de moure's d'una tasca cap a una altre i poder modificar qualsevol tasca després d'haver-la inserit.
- Botons elecció rt-fifos: Aquests dos botons de selecció donen la possibilitat de decidir si es vol crear alguna real time fifo per la comunicació amb altres tasques. Aquesta informació es guarda en una variable individual.
- Botó "Includes": Aquest botó obre una finestra amb la definició dels inludes necessaris per l'execució del codi. Permet borrar/modificar/inserir aquesta informació. Els includes es guarden en una variable concreta.
- Botó "Reset": Elimina tots els elements de la llista de tasques, posa el comptador de tasques a 0 i buida les estructures omplertes al inserir tasques.
- Botó "Generate" : Genera el codi font a partir de tota la informació recol·lectada.

4.4.2 Implementació Generació i Execució

Aquesta es potser la part més important de l'aplicació. Aquesta part fa referència a tota l'implementació que hi ha darrera de l'execució del botó "Generate" explicat anteriorment.
La generació del codi consisteix en transformar tota la informació obtinguda en codi executable, per fer-ho s'aprofita l'estructura estàtica que contenen els mòduls de temps real. Es tenen unes variables estàtiques i unes altres dinàmiques (les obtingudes per la GUI de l'aplicació) i amb totes aquestes es genera el codi. La forma de generar el codi és simple. Es va construint un fitxer .c escrivint la informació de les variables estàtiques i dinàmiques de forma ordenada. Aquesta informació s'escriu creant un fitxer nou anomenat "code.c" en el qual s'escriu tot el codi generat a partir de la informació que es disposa. Per escriure en el fitxer es fa el següent:

```
set Ntasc [array size ArrayTasc]
set out [open "code.c" w]+
puts $out "Informació de generació de codi"
```

La primera línia informa del nombre de tasques a generar, les dos següents serveixen per escriure en el fitxer el codi generat. Per tant en els pròxim punts on s'explica la generació per parts, quan s'utilitzi la funció "puts" servirà per escriure el codi al fitxer a generar. El codi a generar es pot dividir en el codi de capçalera, el codi de definició de tasques, el codi de les funcions, el codi de la funció d'inicialització i el codi de la funció de finalització.

- Definició includes i variables globals: El primer a generar es la informació de capçalera del fitxer, això simplement serà copiar el contingut de la variable dels includes inserits. Aquesta informació serà el que primer s'escriu en el fitxer .c generat. Després es copien totes les variables globals inserides per la GUI. Si s'ha seleccionat la creació de rt-fifos també es definiran en aquest punt. Tot això es fa amb el següent procediment:

```
proc ::GenerateHeader {out n rtf0 rtf1} {
#Includes i capçaleres
global includes
global returns
global declareThread
global pointandcoma
#set includes "#include <rtl.h> \n#include <time.h> \n#include <pthread.h>"
set returns "\n\n"
set pointandcoma ";"
puts $out "$includes"
puts $out ""
puts $out "//Declaracions variables compartides"
puts $out [Text3 get 0.0 end]
#Declarem rtfs..
if {$rtf0 == 1} {
   puts $out "int fifo0;"
   }
if {$rtf1 == 1} {
   puts $out "int fifo1;"
   }
}
```

- Definició tasques: Seguidament es passa a definir les tasques que s'executaran. Per fer-ho serà necessari saber el número de tasques definides i els seus noms, això s'extreu de la llista de tasques creada al inserir tasques mitjançant la GUI implementada. Per a cada tasca es defineix la seva variable amb el nom inserit. En aquest tros de codi es veu com es fa ús de variables "string" per generar codi, algunes variables estàtiques com "nameThread" i altres variables dinàmiques com la variable "tasca".

```
set nameThread "pthread_t thread"
set pointandcoma ";"
         for {set i 1} {$i <= $Ntasc} {incr i 1} {
           set tasca $ArrayTasc($i)
           set name [lindex $tasca 4]
           puts $out "$nameThread$name$pointandcoma"
            }
```

- Definició funcions: En aquest apartat es defineix el contingut de les funcions que donen funcionalitat a la tasca. Aquestes funcions tenen la part de declaració de variables locals, la part de definició periodicitat i la part del codi que dona funcionalitat a la tasca. Per fer això es cridarà al procediment "GenerateTasc" per a cada tasca inserida. El codi d'aquest procediment és:

```
proc ::GenerateTasc {out tasca i} {
#Per a cada tasca
global nameRoutine
global argsRoutine
global int
global structSchedParam
global structSchedParam_priority
global hrtime_t
global setSchedParam1
global makePeriodic
global period
global ret
global closee
global opeen
global whilee
global npPeriodic
global pointandcoma

set pointandcoma ";"
set nameRoutine "void *start_routine"
set argsRoutine "(void *arg)"
set int "int"
set structSchedParam "struct sched_param p"
set structSchedParam_priority "p.sched_priority="
set hrtime_t "hrtime_t now"
set setSchedParam1 "pthread_setschedparam(pthread_self(),SCHED_FIFO,&p)"
set period "unsigned long period = "
set makePeriodic "pthread_make_periodic_np(pthread_self(),gethrtime(),"
set makePeriodic1 ")"
set ret "return 0"
set closee "\}"
set opeen "\{"
set whilee "while(1)"
set npPeriodic "pthread_wait_np()"


      set prio [lindex [lindex $tasca 0] 0]
      set period [lindex [lindex $tasca 0] 1]
      set code [lindex [lindex $tasca 0] 2]
      set vars [lindex [lindex $tasca 0] 3]
      set name [lindex [lindex $tasca 0] 4]


      puts $out "$nameRoutine$name$argsRoutine"
      puts $out "$opeen"
      puts $out "$vars"
```

```
        puts $out "$makePeriodic$period$makePeriodic1$pointandcoma"
        puts $out "$whilee$opeen"
        puts $out "$npPeriodic$pointandcoma"
        puts $out "$code"
        puts $out "$closee"
        #puts $out "$ret$pointandcoma"
        puts $out "$closee"
}
```

Primer es defineixen les cadenes de caràcters estàtiques. Després s'obté la informació de la tasca i s'escriu tota l'estructura de la funció al fitxer.
Cada tasca té associada una funció, per tant, aquest procediment es cridarà per a cada tasca inserida.

- Definició funció inicialització: La funció d'inicialització defineix les tasques i les seves propietats. Això es fa mitjançant la informació de la tasca i les cadenes de caràcters definides i que són estàtiques. Depenen de si s'escull la opció de crear rt-fifo, aquestes seran creades en aquest mòdul.

- Definició funció finalització: La funció de finalització destrueix les tasques creades en el mòdul anterior i les fifos creades.

En el codi següent es mostra la generació dels dos últims punts ja que van associats. Com sempre, primer es defineixen les cadenes de caràcters estàtiques i seguidament es recupera la informació obtinguda a través de la GUI per generar codi segons el que l'usuari vol.

```
set ret "return 0"
set initt "int init_module(void)\{"
set pthreadCreate1 "pthread_create("
set pthreadCreate2 ",&attr,"
set pthreadCreate3 ",0)"
set thread "&thread"
set routine "start_routine"
set pointandcoma ";"
set closee "\}"
set clean "void cleanup_module(void)\{"
set delete1 "pthread_delete_np("
set delete2 ")"
set thread1 "thread"
set schedParam "struct sched_param sched_param;"
set attr "pthread_attr_t attr;"
set attrInit "pthread_attr_init (&attr);"
set prio "sched_param.sched_priority ="
set sched "pthread_attr_setschedparam (&attr, &sched_param);"
set setfp "pthread_attr_setfp_np(&attr, 1);"
```

```
#Generate Init
puts $out "$initt"
set entra 0
if {$rtf0 == 1} {
   puts $out "int fifo_status;"
   set entra 1
}
if {$rtf1 == 1} {
   if {$entra == 0} {
      puts $out "int fifo_status;"
      }
   }

puts $out $schedParam
puts $out $attr

if {$rtf0 == 1} {
  puts $out "rtf_destroy(0);"
  puts $out "fifo_status = rtf_create(0, 4000);"
  puts $out "if (fifo_status) {"
  puts $out "rtl_printf(\"RTLinux measurement test fail. fifo_status=%d\\n\",fifo_status);"
  puts $out "return -1;"
  puts $out "}"
  puts $out "fifo0 = open(\"/dev/rtf0\", O_NONBLOCK);"
  puts $out "if (fifo0 < 0) {"
  puts $out "rtl_printf(\"/dev/rtf0 open returned %d\\n\", fifo0);"
  puts $out "return (void *) -1;"
  puts $out "}"
}

if {$rtf1 == 1} {
  puts $out "rtf_destroy(1);"
  puts $out "fifo_status = rtf_create(1, 4000);"
  puts $out "if (fifo_status) {"
  puts $out "rtl_printf(\"RTLinux measurement test fail. fifo_status=%d\\n\",fifo_status);"
  puts $out "return -1;"
  puts $out "}"
  puts $out "fifo1 = open(\"/dev/rtf1\", O_NONBLOCK);"
  puts $out "if (fifo1 < 0) {"
  puts $out "rtl_printf(\"/dev/rtf1 open returned %d\\n\", fifo1);"
  puts $out "return (void *) -1;"
  puts $out "}"
}

for {set i 1} {$i <= $Ntasc} {incr i 1} {
                    set tasca $ArrayTasc($i)
                    set name [lindex $tasca 4]
                    set prioritat [lindex $tasca 0]
                    puts $out $attrInit
                    puts $out $prio$prioritat$pointandcoma
                    puts $out $setfp
                    puts $out $sched
                              puts $out
"$pthreadCreate1$thread$name$pthreadCreate2$routine$name$pthreadCreate3$pointand
coma"
                    }
puts $out "$ret$pointandcoma"
puts $out "$closee"
```

```
#Generate Cleanup
puts $out "$clean"

if {$rtf0 == 1} {
   puts $out "close(fifo0);"
   puts $out "rtf_destroy(0);"
}

if {$rtf1 == 1} {
   puts $out "close(fifo1);"
   puts $out "rtf_destroy(1);"
}

for {set i 1} {$i <= $Ntasc} {incr i 1} {
                    set tasca $ArrayTasc($i)
                    set name [lindex $tasca 4]
                                puts $out "$delete1$thread1$name$delete2$pointandcoma"
                       }
puts $out "$closee"

close $out
```

Vist tot el codi implementat es pot veure com la manera de generar el codi és simple, s'utilitza la informació obtinguda a partir de la GUI i variables pre-definides. Per exemple, quan es necessita definir el codi que declara les tasques, el codi a generar serà:

```
pthread_t threadTasca1;
pthread_t threadTasca2;
```

En aquest codi generat, les variables estàtiques són:

```
set thread "pthread_t thread"
set pointandComa ";"
```

I les variables dinàmiques són els noms de les tasques inserides per l'usuari a través de la GUI.

```
set tasca $ArrayTasc($i)
set name [lindex $tasca 4]
# on $i es la variable que indica sobre quina tasca estem treballant ara mateix.
# $ArrayTasc es l'array que conté les estructures de les tasques
# $tasca es l'estructura de la tasca $i i en la posició 4 hi ha el nom de la tasca
```

Per tant, la forma de generar el codi és concatenant les diferents variables tan dinàmiques com estàtiques obtingudes.

```
sortida = "$thread$name$pointandComa"
```

Un cop generat íntegrament el codi aquest s'haurà de compilar i posteriorment executar. Per fer això simplement s'executa una comanda.

```
exec make test
```

Aquesta comanda executa un Makefile en el mateix directori ja creat. Aquest Makefile compila per generar el fitxer objecte, l'insereix durant un segon i posteriorment l'extreu. Un cop fet això s'executa un programa monitor per obtenir les dades de planificació per si es vol simular l'execució de les tasques amb el botó "simulator". El programa monitor s'executa en background durant un temps i després s'elimina amb un script "killmonitor.sh".

Codi Makefile:

```
all: code.o

include /usr/src/rtlinux/rtl.mk
clean:
	rm -f *.o

test: all
	#-rmmod sound
	#-rmmod rt_process
	#-rmmod frank_module
	(cd /usr/src/rtlinux/; scripts/rmrtl)
	(cd /usr/src/rtlinux/; scripts/insrtl)
	@insmod code.o
	sleep 1
	@rmmod code
	./monitor&
	sleep 2
	sh killmonitor.sh
#include $(RTL_DIR)/Rules.make
```

Codi killmonitor.sh

```
#!/bin/bash

kill  `ps uxc | grep -i "monitor" | awk '{print $2}'`
```

Codi monitor.c

```
#include <stdio.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include "rtl_fifo.h"

int main()
{
        FILE *f1;
        int fd0;
        int n;
        unsigned long id;
        long int time;
        int prio;
        char t;

        if ((fd0 = open("/dev/rtf0", O_RDONLY)) < 0) {
                fprintf(stderr, "Error opening /dev/rtf0\n");
                exit(1);
        }

        f1 = fopen("monitor2.txt","w+");
        fclose(f1);
        n = read(fd0, &t, sizeof(char));
        printf("%c \n",t);
        while (1) {
        f1 = fopen("monitor2.txt","a+");
        n = read(fd0, &id, sizeof(unsigned long));
        n = read(fd0, &prio, sizeof(int));
        n = read(fd0, &time, sizeof(long int));
        printf("ID: %ul  TIME: %ld PRIO: %d\n", (unsigned long)id, (long int)time,
        (int)prio);
        fprintf(f1,"%ul - %ld\n",(unsigned long)id, (long int)time);
        fflush(stdout);
        fclose (f1);
        }

        return 0;
}
```

Un cop revisats els detalls d'implementació es pot comprovar que s'ha generat una aplicació amb una complexitat del disseny un tant alta. Tot això utilitzant Tcl/Tk per tant s'ha corroborat que l'ús d'aquest llenguatge es possible per el desenvolupament de tal aplicacions. Ara queda veure com adjuntar aquests dos grups d'implementació, la GUI i la generació del codi, per obtenir el resultat final de l'aplicació.

En la següent figura es mostra un resum del disseny/implementació de l'aplicació. En aquesta figura es mostren tots els passos que segueix l'aplicació per a obtenir els resultats finals. L'aplicació comença amb una interfície que demana informació a l'usuari de les tasques, les seves propietats i altre informació general sobre el mòdul a generar. Aquesta interfície dona facilitat a l'usuari per inserir i/o modificar la informació. Amb aquesta informació emmagatzemada en variables i estructures de dades i les cadenes de caràcter pre-definides, es genera el codi font. Aquest codi es compila per obtenir el fitxer objecte el qual és inserit durant un segon al kernel. Aquest segon que el mòdul ha estat inserit es monitoreja amb l'aplicació monitor el qual generà un fitxer de text que serveix per simular l'execució de les tasques.
En tot aquest procediment, entren en joc varies aplicacions creades, el Generador de Tasques per generar el codi i englobar-ho tot, l'aplicació monitor per llegir les dades del planificador en temps real, l'script killmonitor.sh per parar l'execució de l'aplicació monitor i l'aplicació per mostrar la simulació de l'execució de les tasques llegint del fitxer de text monitor.txt

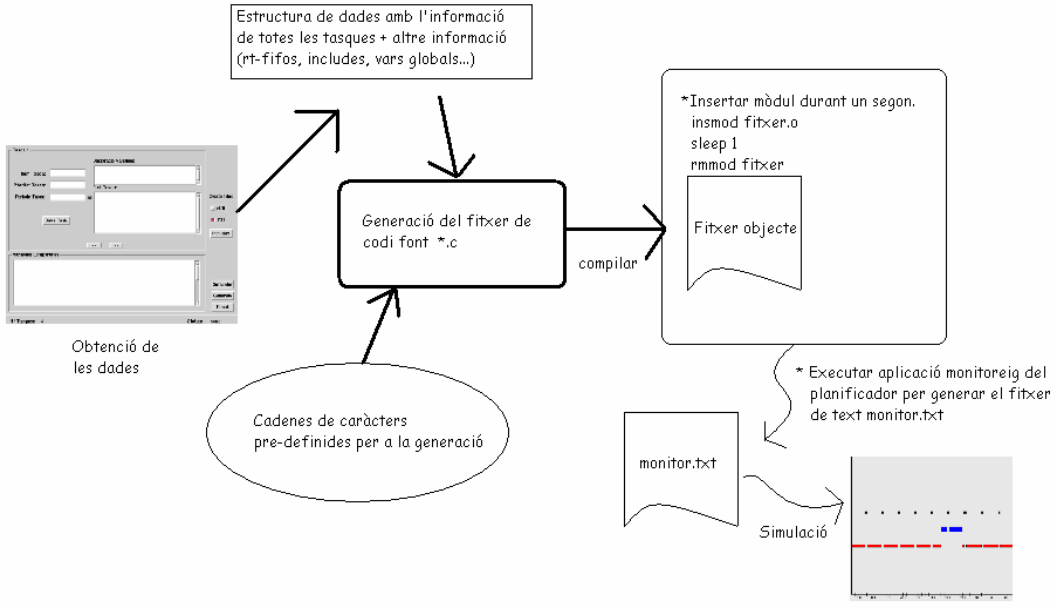


Figura 4.8: Disseny/Implementació Generador de Tasques

En l'aplicació creada s'ha utilitzat tots els conceptes analitzats anteriorment, la utilització de RTLinux, Tcl/Tk, la simulació de l'execució de les tasques, etc. En el següent apartat s'explica amb un exemple com funciona l'aplicació i els resultats obtinguts.

4.5 Funcionament de l'aplicació (Exemple d'execució)

En aquest punt es detallarà el funcionament de l'aplicació creada, per fer-ho es simularà una situació real d'execució. L'exemple que es crea consisteix en dues tasques amb les següent propietats:

Tasca 1

- Nom: Tasca1
- Prioritat: 2
- Període: 500000000 ns
- Codi a executar: un bucle amb un nombre d'iteracions que vindrà definit per una variable global

Tasca 2

- Nom: Tasca2
- Prioritat: 2
- Període: 10000000 ns
- Codi a executar: incrementa la variable global en cada execució

Aquestes dues tasques compartiran una variable global. La Tasca1 simplement realitza una operació N vegades, on N ve definit pel nombre d'iteracions que farà en cada execució. Aquest nombre d'iteracions ve definit per una variable compartida entre les dues tasques. La Tasca2 en cada execució incrementa la variable compartida entre tasques així per cada execució de la Tasca2 es veurà incrementat el temps d'execució de la Tasca1. Aquestes dues tasques no fan res d'especial, aquest exemple simplement es per comprovar la facilitat en que es poden crear tasques en temps real amb el generador de tasques i el posterior anàlisis de l'execució amb la simulació d'aquesta.

El primer que s'ha de fer, és obrir l'aplicació de Generació de Codi. Per obrir l'aplicació s'ha de cridar a vTcl i al menú obrir fitxer existent per carregar l'aplicació. Per posar en marxa l'aplicació s'ha de prémer la combinació de tecles "ALT+e" per passar el mode test. Tot això s'ha de fer amb l'usuari root del sistema per poder executar comandes de rtlinux.

Un cop oberta l'aplicació és seguiran els següents passos:

1.- S'afegeix la informació de la Tasca1, nom de la tasca, prioritat, període, codi d'execució, variables del codi.

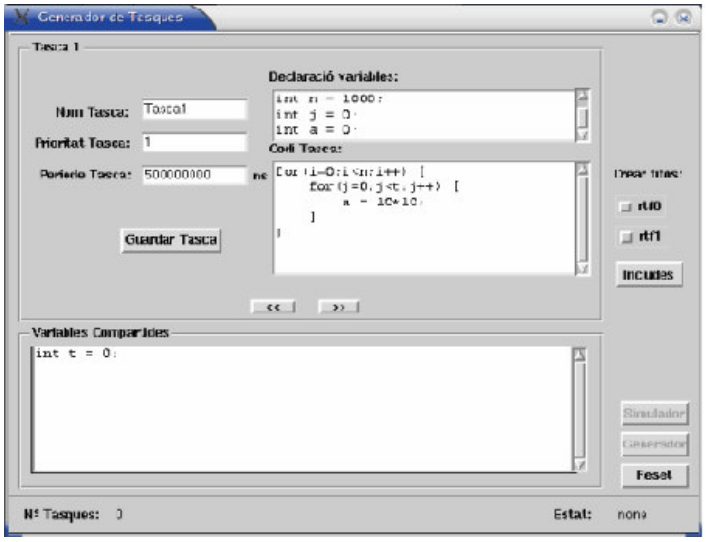


Figura 4.9: *Entrada de la Tasca1*

2.- Un cop afegida la informació de la Tasca1 es prem el botó "Guardar Tasca" per guardar els canvis i automàticament l'aplicació passa a preguntar per les dades de la següent tasca.

3.- En el següent pas s'afegeix la informació de la Tasca2 de la mateixa manera que en el punt 1 i es prem de nou "Guardar Tasca" per guardar els canvis d'aquesta última tasca. Si en aquest punt s'ha de modificar alguna de les tasques inserides s'utilitzaran els botons de desplaçament de les tasques per anar d'una tasca a una altre i modificar-ne la informació. Un cop modificada la informació d'una tasca es prem "Guardar Tasca" per a que s'apliquin els canvis.

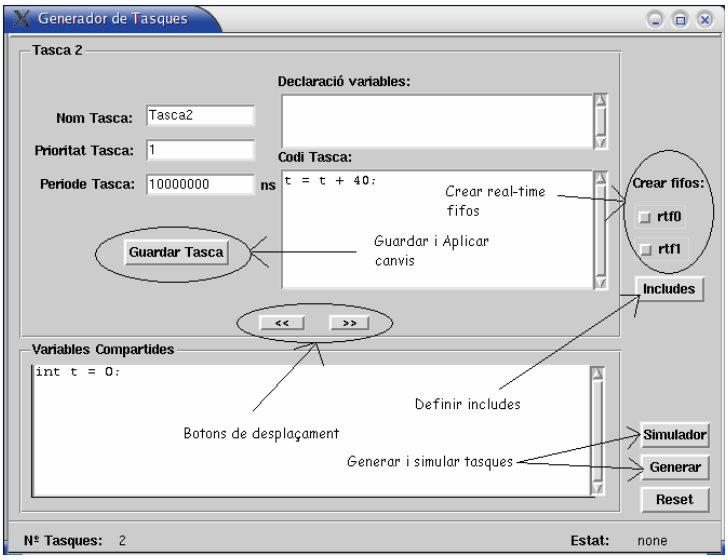


Figura 4.10*: Entrada de la Tasca2*

4.- Un cop inserides les tasques, s'haurà de comprovar si s'han d'afegir variables compartides entre tasques, si es així, s'afegiran al requadre de la part inferior de l'aplicació.

5.- Si hi ha la necessitat d'afegir algun include s'haurà de prémer el boto "Includes" on s'obre una finestra per introduir els includes necessaris per les tasques.

6.- Si hi ha la necessitat d'utilitzar real time fifos, es selecciona l'opció de crear una o dos real time fifos per utilitzar-les en alguna de les tasques inserides.

7.- Un cop inserida tota la informació es prem el boto "Generate", aquest boto genera el codi, el compila i l'insereix durant un segon al kernel de RTLinux i posteriorment es monitoritza l'execució de les tasques que incorpora el mòdul inserit.

8.- Tant sols queda veure la simulació de l'execució de les tasques per si es vol analitzar si tot a funcionat correctament. Per fer això es prem el boto "Simulació".

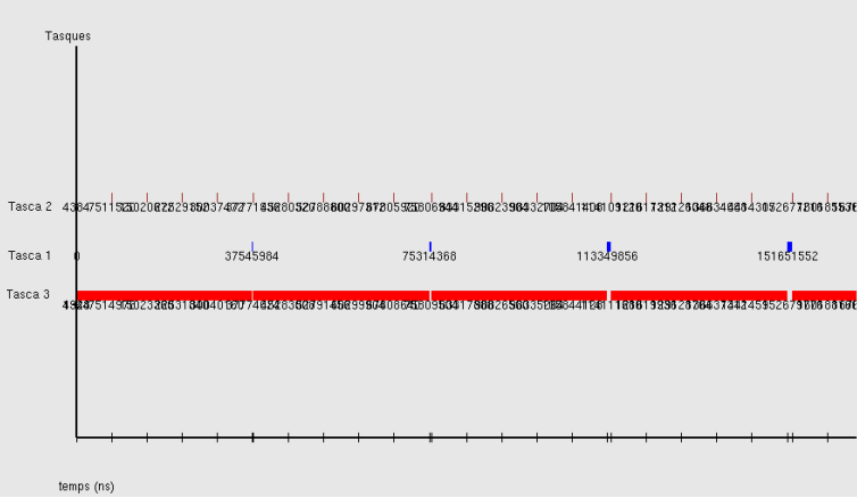


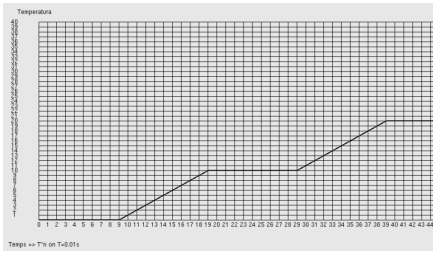
Figura 4.11: *Resultat de la simulació*

Un cop feta la simulació es comprova que tot a funcionat correctament, en la simulació es veu l'execució de tres tasques, la tasca de color vermell es la tasca de Linux, aquesta tasca sempre esta en execució sinó hi ha cap altre tasca en la CPU. Les altres dues tasques són les tasques creades automàticament amb el generador de tasques. La Tasca1 és la que executa uns bucles, el número de bucles a executar dependrà de la Tasca2 que augmentarà aquest número de bucles. Es veu com en cada execució de la Tasca1 augmenta el temps de còmput d'aquesta ja que s'està incrementant el nombre de bucles d'aquesta tasca mitjançant la Tasca2. Per cada execució de la Tasca2 s'incrementa en 40 iteracions el bucle de la Tasca1.
Per cada execució de la Tasca1 s'executa 5 vegades la Tasca2, per tant en cada execució de la Tasca1 es veu incrementat el nombre de bucles en 40*5 = 200 iteracions. En la imatge anterior es veu fàcilment com la barra de la Tasca1 augmenta en cada execució.

En tant sols 8 passos s'han creat dues tasques en temps real que comparteixen informació. S'ha aconseguit desenvolupar un primera versió d'una eina que pot ser de molta utilitat. Qualsevol programador amb coneixements de programació en C podrà programar tasques en temps real amb aquesta aplicació. També dona la possibilitat d'analitzar l'execució per entendre millor el temps real. S'ha creat una eina bàsica i fàcil d'utilitzar, en pròximes versions es podria anar completant i eliminant les restriccions que pot tenir aquesta aplicació.

El codi de l'aplicació es pot trobar en l'annex on es troba el codi de les aplicacions desenvolupades.

El codi de l'exemple generat es pot trobar en l'annex on es troba el codi dels exemples generats.



5

Exemples i Resultats

---

5. Exemples i Resultats

5.1 Previ

En aquest capítol es mostren una sèrie d'exemples al lector per a que quedin clars els conceptes i els objectius del projecte. Recordar que l'objectiu del projecte és analitzar la viabilitat de construir un Sistema Operatiu en Temps Real i desenvolupar-ne aplicacions. Tot això s'ha de aconseguir amb ús de software lliure. En aquest punt ja s'han assolit els objectius i ara tant sols queda veure'n exemples per provar que s'han assolit. Aquest capítol es dedicarà a descriure una sèrie d'exemples per comprovar alguns conceptes assolits, algun d'aquests exemples seran referents a interrupcions, a generació de tasques minimitzant el període , a reproduir un fitxer de só o en controlar algun procés. Aquests exemples és generen de forma manual o automàticament amb el Generador de Tasques, posteriorment s'analitzen els resultats detalladament i l'execució de les tasques en el planificador de temps real.

Els exemples es divideixen en dos grups, els generats automàticament i els generats manualment. El exemples generats automàticament es fan utilitzant el Generador de Tasques desenvolupat en el capítol anterior. El tipus d'exemples generats en aquest grup són de control de processos, en concret, el control d'un procés integral i el control d'un procés mes complexa, control de la temperatura i nivell d'un tanc d'aigua. En aquest grup també es fa un anàlisis de la periodicitat de les tasques canviant el valor del període d'una tasca i analitzant el resultat obtingut. En els exemples generats manualment es fa programant ja que amb el Generador de Tasques no es poden fer aquests tipus d'exemples, ja s'explicarà per que no es poden generar aquests exemples per les restriccions de l'aplicació. Els exemples generats ens aquest grup són dos, el primer exemple es proven les interrupcions del teclat i s'analitza profundament amb l'execució de la interrupció en el planificador de temps real, per últim, el segon exemple serà l'execució d'una tasca que reprodueix un fitxer de só amb una cert període.

S'han elegit cinc exemples amb els quals es cobreixen els aspectes importants estudiats en el llarg del projecte.

5.2 Exemples generats manualment

En aquest punt es passen a explicar els exemples creats manualment, ens aquests exemples es veuen alguns dels aspectes importants de RTLinux i els sistemes en Temps Real. Concretament es detallen dos exemples, el primer exemple s'utilitzen les interrupcions, concretament la interrupció del teclat, i com a segon exemple d'aquest apartat es mostra com una tasca reprodueix un fitxer de só, la reproducció es farà en una certa periodicitat.

5.2.1 Exemple 1: Interrupció del teclat

En aquest exemple es mostra el funcionament bàsic de les interrupcions del Sistema Operatiu i com aquestes interrupcions es tracten com a més prioritaries ja que RTLinux té accés directe al hardware. L'exemple simplement el que fa es esperar per una interrupció del teclat, quan és prem una tecla del teclat es desperta una tasca la qual executa una instrucció, imprimir qualsevol caràcter. Aquesta tasca podria ser una rutina de tractament de la interrupció. Per exemple, aquesta tasca podria enviar unes certes dades algun lloc. Si aquesta tasca es posa amb màxima prioritat quan es premés una tecla, el sistema operatiu deixaria de fer tot el que estava fent per enviar les dades (execució de la tasca). Aquesta tasca s'executa sempre amb el mateix temps ni que la carrega de Linux sigui alta ja que és té dos punts clau: S'està utilitzant interrupcions RTLinux, per tant, és té accés directe al hardware, es tracta la interrupció per sobre d'altres interrupcions de Linux, el segon punt important es que la interrupció executa una tasca en temps real i aquesta ni que el sistema estigui carregat, s'executarà deixant de banda l'execució de Linux.
El codi de l'exemple es pot trobar en l'annex on es troba el codi de tots els exemples.

```
OTHER: 36680499201 amb prioritat 0 - 3488
+960
LINUX: 37499247401 amb prioritat -1 - 4547040
OTHER: 36680499201 amb prioritat 0 - 3008
+640
LINUX: 37499247401 amb prioritat -1 - 4546624
OTHER: 36680499201 amb prioritat 0 - 3200
+736
LINUX: 37499247401 amb prioritat -1 - 182592
OTHER: 36680499201 amb prioritat 0 - 3584
+864
LINUX: 37499247401 amb prioritat -1 - 4661792
OTHER: 36680499201 amb prioritat 0 - 3136
+576
LINUX: 37499247401 amb prioritat -1 - 184960
OTHER: 36680499201 amb prioritat 0 - 2880
+576
LINUX: 37499247401 amb prioritat -1 - 304032
OTHER: 36680499201 amb prioritat 0 - 2528
+576
LINUX: 37499247401 amb prioritat -1 - 4765216
OTHER: 36680499201 amb prioritat 0 - 3424
+640
```

Figura 5.1: *Resultat exemple1*

Si es compila e insereix el mòdul de temps real es comprova (amb la comanda dmesg) que al prémer qualsevol tecla s'executa la tasca (imprimir el caràcter "+" al buffer del kernel).
En la il·lustració anterior és veu el resultat generat de prémer tecles quan el mòdul es inserit. Es veu com s'executa la tasca de Linux amb prioritat negativa, quan es prem una tecla treu de la CPU la tasca de Linux per passar a executar la tasca que desperta la interrupció. El temps de còmput que apareix al final de cada línia és la suma del temps de tractar la interrupció més el temps de planificació més el temps de còmput de la tasca. A sota de cada línia surt el temps tant sols del còmput de la tasca. Durant tot el projecte s'ha analitzat majoritàriament tasques periòdiques, tasques que cada cert temps executen una sèrie d'instruccions i aquestes han de complir uns terminis d'execució. En aquest exemple, s'analitza una tasca asíncrona, no té cap mena de periodicitat, simplement es passa a executar quan s'activa una interrupció, concretament la del teclat. Exemples reals d'aquests tipus de tasques poden ser tasques que esperen a la interrupció d'un port del PC per a la lectura de dades. Aquest tipus de dades han de respondre a interrupcions en un temps mínim, han de tenir la propietat de poder tractar milers d'interrupcions en un segon sense perdre'n cap. En la figura anterior és veu com el temps de tractament d'interrupcions està ajustat a l'ordre dels nanosegons. La següent figura mostra tot el que s'explica de forma més clara amb l'execució de la tasca interrupció en el planificador de temps real.

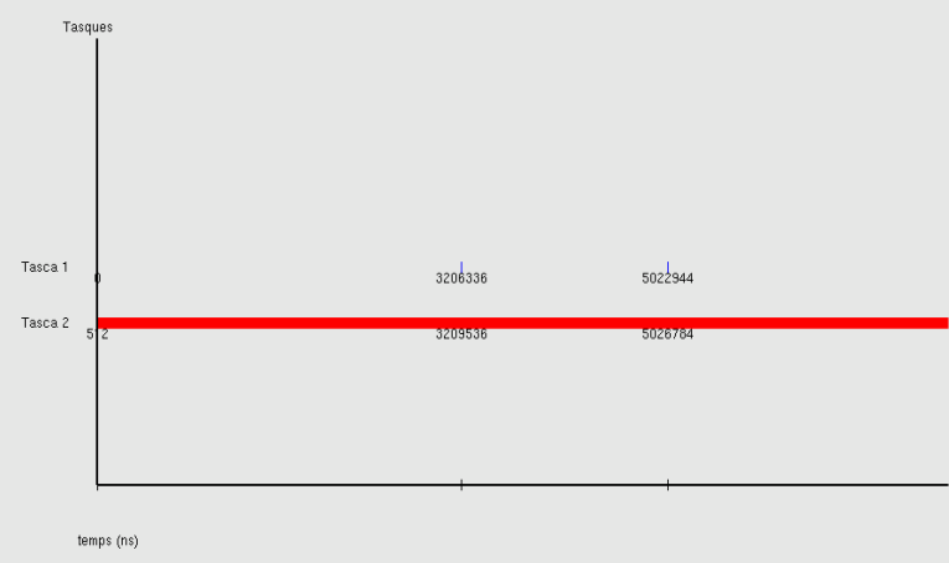


Figura 5.2: *Planificació exemple1*

En aquesta il·lustració es veu com la Tasca1, la tasca que es despertada per la interrupció del teclat, no segueix cap mena de patró de periodicitat com les altres tasques analitzades en els altres exemples. En la imatge podem analitzar que s'ha premut la tecla dues vegades amb una diferència de 2 milisegons, casi no s'ha deixat de prémer la tecla. Es pot arribar a obtenir una resolució molt més baixa d'interrupcions però manualment no es pot tant ràpid. Amb l'ús d'un port paral·lel es podria comprovar que es poden tractar interrupcions

hardware en un temps de l'ordre dels nanosegons. En un segon sense anar molt lluny, es podrien arribar a tractar al voltant d'unes 100.000 interrupcions de teclat. Si una persona pugues prémer 100.000 tecles per segon, el Sistema Operatiu en Temps Real podria captar-les totes i emmagatzemar-ne el valor.

Queda clar doncs, que es disposa d'un sistema en temps real prou eficient i que compleix els requeriments del projecte pel que fa al marc de les interrupcions.

Gràcies a un Sistema Operatiu d'aquestes característiques es pot arribar a fer infinitat de processos de monitorització de dades, control, etc. amb una precisió sobre les interrupcions de l'ordre dels nanosegons.

5.2.2 Exemple 2: Prova de só

Aquest exemple és el segon del grup de generació manualment. Aquest exemple ja s'ha comentat en el capítol 3, però aquí s'analitza més profundament. És un dels exemples que deixa més clar la potència d'un sistema operatiu en temps real. Recordar que aquest exemple consisteix en reproduir un fitxer de só a través d'una tasca en temps real. Aquesta tasca reprodueix el fitxer de só amb una freqüència de 8192Mz.

Ja s'ha comentat en el capítol 3 la diferència d'executar aquest exemple i el mateix però amb una tasca en l'àrea d'usuari i que la diferència recau en la continuïtat de la reproducció del só. Si es reprodueix el só amb la tasca de no temps real i amb una carrega alta en el sistema operatiu es nota com la reproducció es va tallant per causa d'aquesta sobrecarrega, l'efecte seria el mateix que passa quan es reprodueix un MP3 en Windows i s'està carregant el sistema amb molts processos, es talla la reproducció a vegades. Ara bé, amb una tasca en temps real això no passarà, si el sistema operatiu linux està sobrecarregat, aquest haurà d'esperar a que finalitzi la reproducció del fitxer de só per part de la tasca en temps real per executar els processos que té pendents ell mateix. El resultat serà una execució nítida encara que hi hagi sobrecàrrega en el sistema.

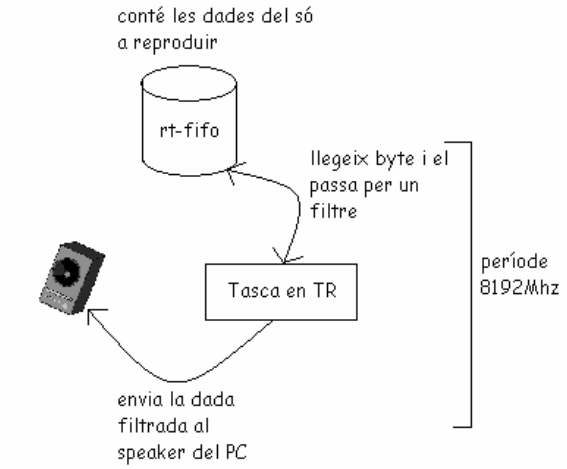


Figura 5.3: *Exemple2: Prova de só*

La tasca en temps real que implementa el codi generat el que fa és llegir d'una real time fifo el fitxer de só. Amb una freqüència de 8192Mhz llegeix una dada de la fifo, la filtra i l'envia al port del speaker del PC. Per tant ara s'està tractant amb una tasca periòdica, amb període 122070,3125 nanosegons (8192Mhz). Està clar el procediment en la figura anterior.

Es de vital importància que la reproducció segueixi estrictament el període definit per la correcta reproducció del fitxer de só, per tant, és una bona solució aplicar temps real.

Aquesta tasca s'ha provat i s'ha vist com la reproducció del só no es talla ni que es carregui el sistema. S'ha analitzat l'execució en el planificador i els resultat es molt semblant amb el sistema carregat o no carregat. En la següent figura es mostra la simulació.

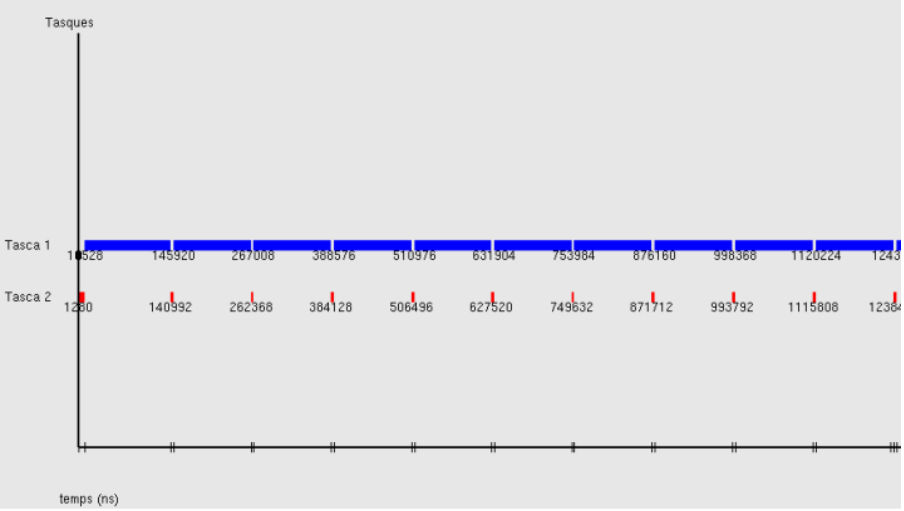


Figura 5.4: *Planificació exemple2*

En la simulació de la planificació es veu com la Tasca 2 (tasca en temps real) s'executa cada 130000 nanosegons (122070,315 + el temps de planificació), aquesta es amb la freqüència amb que es reprodueix el fitxer de só. Aquesta simulació serveix tant pel sistema sobrecarregat com no, s'assembla molt la simulació, es casi idèntica. Sempre que arriba l'instant de llegir i enviar la dada al speaker, la tasca en temps real agafa la CPU per executar-se i la Tasca1 (Linux) serà la que haurà d'esperar a executar-se. Si es fes això amb una tasca en no temps real, els rectangles vermells, els que mostren l'execució de la tasca de reproducció de só, és veurien tallats si el sistema estigués sobrecarregat i d'aquí vindrien els talls alhora d'escoltar la reproducció.

En aquest exemple es mostra la importància d'un sistema en temps real i les seves avantatges. S'ha demostrat d'una manera pràctica i molt perceptible.

5.3 Exemples generats amb l'aplicació (Generador de Tasques)

Els següents exemples es creen amb l'aplicació desenvolupada en el capítol anterior, el Generador de Tasques. Serà més fàcil ara crear els exemples que en el punt anterior en que es feia de forma manual. En el punt anterior no es podien generar els exemples amb l'aplicació per que no es prou general per poder generar qualsevol tipus de tasca en temps real, l'aplicació té restriccions. Per exemple, l'aplicació desenvolupada tant sols genera tasques periòdiques, per tant l'exemple de les interrupcions de teclat és impossible de moment generar-la amb l'aplicació. L'altre exemple, prova de só, potser no seria del tot impossible generar-lo amb l'aplicació si s'integrés la funció filtre al codi de la tasca en temps real però per més facilitat s'ha optat per la implementació manual.

Ara bé, pels exemples generats en aquest apartat, és de gran utilitat el generador de tasques, no tant sols per la seva capacitat de generar automàticament i de forma molt fàcil les tasques en temps real sinó per la facilitat i rapidesa en que es poden modificar i tornar a executar les tasques.
En aquest apartat es comenten tres exemples. El primer exemple s'analitza l'execució d'una tasca modificant el seu període d'execució fins al seu límit. Per fer les proves i modificacions és fa fàcilment amb l'aplicació de Generador de Tasques variant el valor del període de la tasca.
Els altres dos exemples fan referència al control de processos en temps real, es veu com es controlen processos amb temps real i que es possible. Es comprova la facilitat en que es pot muntar un sistema simulat amb el Generador de Tasques per la seva posterior implementació en un entorn real.

5.3.1 Exemple 3: Prova de la periodicitat

La prova de la periodicitat consisteix en crear una tasca en temps real qualsevol i modificar-ne la periodicitat. Per cada modificació veure'n el resultat de la planificació i comprovar fins quin valor mínim es pot baixar el valor del període i veure que passa si es baixa més.

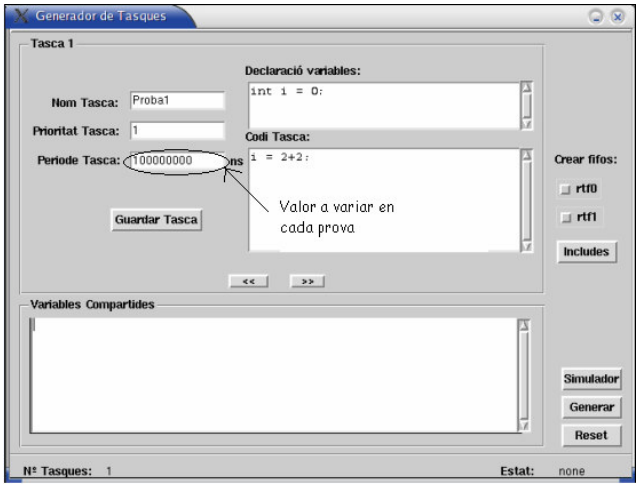


Figura 5.5: *Creació Exemple3*

La tasca en temps real que s'ha creat és una tasca que cada cert període fa una operació de suma simple, aquest període serà variat. En cada test s'insereix el mòdul en temps real obtingut durant 1 milisegon o el que és el mateix, 100000000 nanosegons.

La tasca és creada inicialment amb un període de 100000000 nanosegons. La forma de generar aquest exemple es mostra en la figura anterior. Amb aquest període durant un microsegons s'obté l'execució de la tasca dues vegades.

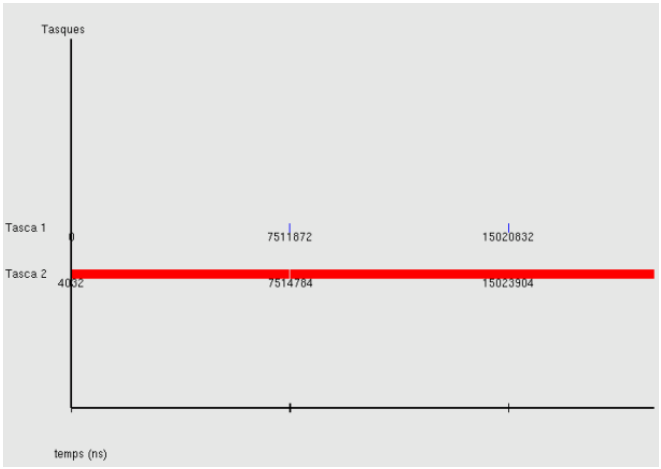


Figura 5.6: *Resultat amb P=100000000ns*

En la imatge anterior es veu com la tasca s'activa en l'instant t=7511872ns i la següent activació en t=15020832, 100000000 nanosegons després. Tant sols s'activa la tasca dos vegades en un microsegon.

Només mirant la simulació de la tasca es veu com aquesta pot tenir una periodicitat bastant més petita.

Les següents proves es fan per valors de periode=10000000ns, 1000000ns, 100000ns, 10000ns i 2500ns.

En cada prova tant sols cal modificar el valor de la caixa de període de las tasca en el Generador de Tasca i prémer el boto "Generar" i posteriorment "Simular" i ja s'obtenen els resultats. És una forma bastant ràpida d'obtenir resultats.

La simulació que s'obté per a un període de 2500ns és la següent:

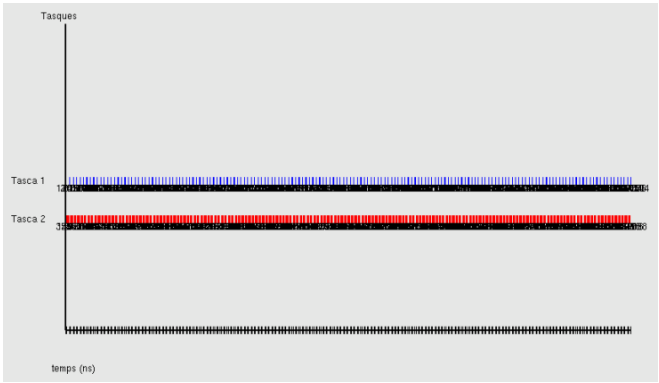


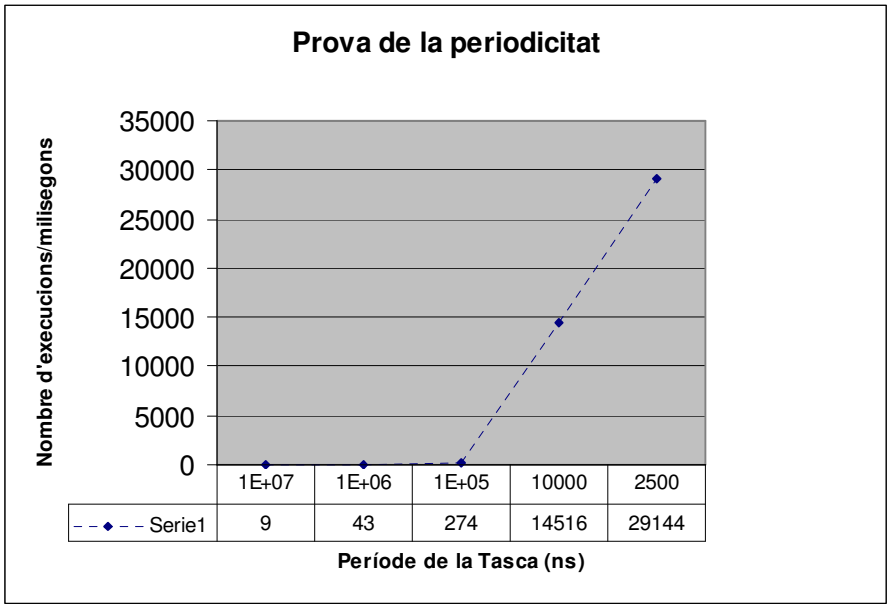
Figura 5.7: *Resultat amb P=2500ns*

Si es comparen les dues imatges anteriors es pot veure com en la segona imatge per l'exemple amb període de 2500ns la tasca es repeteix molt freqüentment agafant molt temps de CPU a la tasca Linux. La Tasca 2 és la de Linux i la Tasca 1 la que s'ha creat per l'exemple, es veu com les dos tasques tenen talls, aquests talls són casi igual de freqüents en les dos tasques. Cal dir que aquesta és la última prova que ha funcionat. Al provar un període de 2000ns el sistema s'ha penjat. Mes que penjat el que possiblement ha passat és que la tasca en temps real s'executa amb tanta freqüència que no dona temps a l'execució de Linux i dóna la sensació a l'usuari de que el sistema està penjat però en realitat la tasca en temps real es possible que s'estigui executant amb normalitat.

En la següent taula es mostra la relació entre nombre d'execucions per milisegons de la tasca i el període de la tasca:

Període de la Tasca	Nombre d'execucions/milisegon
10000000ns	9
100000ns	43
100000ns	274
10000ns	14516
2500ns	29144
2000ns	-

En la següent gràfica es mostra el resultat de modificar la periodicitat de la tasca, obtenint el nombre d'execucions per milisegon obtingut. S'obtenen 9 execucions per un període de 10000000ns i per l'altre cantó amb una periodicitat de 2500ns s'obtenen 29144 execucions. Aquest últim valor frega el límit del valor mínim de període per a la tasca analitzada ja que per un període de 2000ns el sistema ja no ho suporta.



5.3.2 Exemple 4: Control d'un procés simple

El següent exemple s'inicia la generació de tasques per al control en temps real. En aquest exemples no es farà cap control, simplement es mostrarà com seria el funcionament per després passar a un exemple més complex.

Un controlador és aquell sistema que controla el valor d'una variable dins d'uns rangs proposats.

Tot sistema de control consisteix en una planta i un control d'aquesta planta, hi ha diferents tipus de sistemes de control ja siguin de llaç obert o llaç tancat.

Els controls de llaç obert són aquells que la sortida del sistema no té efecte en l'entrada, és necessari una calibració molt bona i no reacciona a pertorbacions externes. En la següent imatge s'il·lustra l'esquema d'un control en llaç obert.

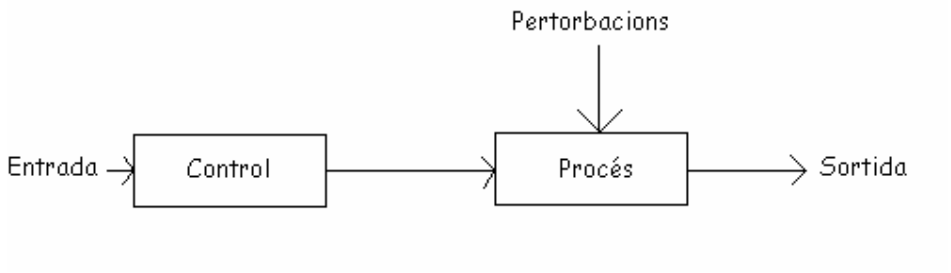


Figura 5.8: *Control en llaç obert*

També tenim els controladors en llaç tancat, s'aplicarà doncs als processos que tenen pertorbacions. L'objectiu d'aquests controladors és mantenir una variable en uns nivells desitjats davant de les pertorbacions externes.

El primer exemple de control en temps real és un sistema de llaç obert, en canvi el segon exemple del control és un sistema del tipus llaç tancat.

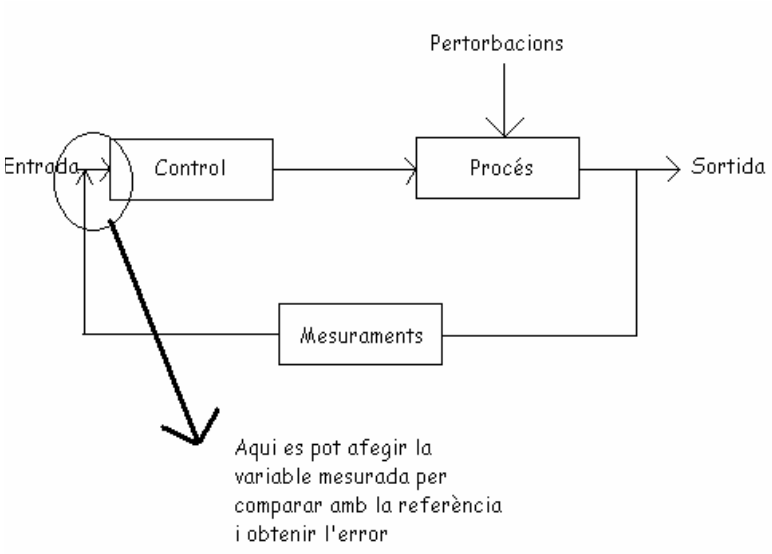


Figura 5.9: *Control en llaç tancat*

En aquest primer exemple simplement s'implementarà un sistema d'aquest tipus de control en llaç obert, però en realitat no farà cap mena de control. El sistema a simular serà una funció integradora. La funció que la descriu en el discret seria:

Y(n) = Y(n-1) + U(n), on
Y(n): és la sortida del sistema en l'instant n
Y(n-1): és la sortida del sistema en l'instant anterior
U(n): és l'acció de control en l'instant actual

El sistema el que fa es sumar la sortida de l'estat anterior amb el valor del control.
El control que s'implementa en aquest és simplement que cada 10 intervals de temps la sortida del control serà 1 i cada 10 altres intervals de temps la sortida serà 0. La resposta del sistema serà una resposta esglaó. El que fa el sistema es integrar i com que l'acció de control dona com entrada una constant la sortida del sistema tendirà a ser una recta amb un pendent de 45º. La integral d'una constant és una recta.
Aquest exemple s'implementa per entendre com es pot implementar aquest tipus de sistema amb l'aplicació de generació de tasques en temps real.
Seguidament es mostra com s'ha creat aquest exemples:

Pas1: S'implementa el procés a controlar (l'integrador), i s'escriu en cada moment la variable de la sortida per poder-ne veure el resultat.

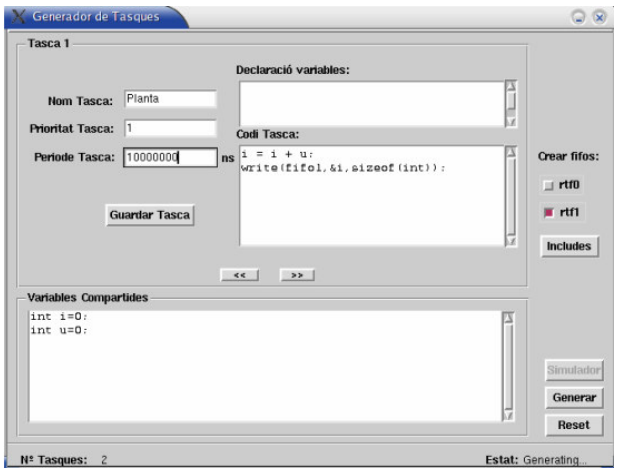


Figura 5.10: *Pas1*

Pas2: S'implementa la tasca de control que la sortida de la qual repercutirà en el procés inicial. Cal tenir les variables de sortida i control com a compartides entre mòduls i es crearà la rt-fifo1 per escriure les dades de sortida.

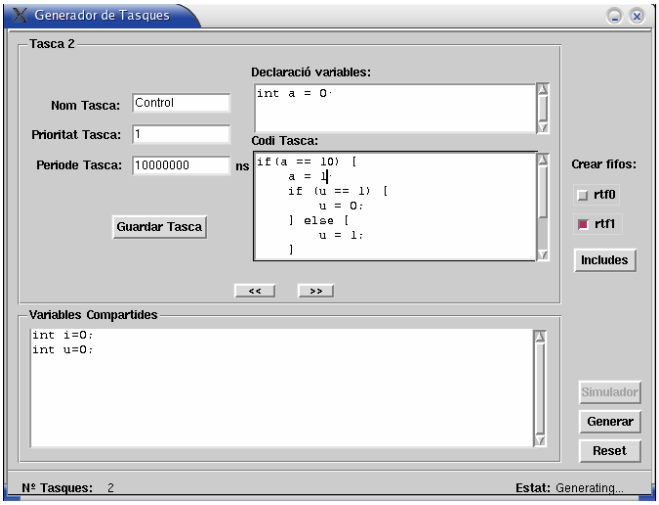


Figura 5.11*: Pas2*

Pas3: Es genera el codi i es monitoritza el resultat obtingut. En la figura següent es mostra el resultat obtingut de la sortida del sistema. Es pot veure com és una recta ja que s'està integrant una constant. Per poder veure'n el resultat, s'ha creat una aplicació que mostra gràficament els resultats obtinguts de la sortida del sistema. Per veure el codi de l'aplicació anar al Annex al apartat de codi font de les aplicacions.

S'ha comprovat que es bastant senzill crear sistemes de control amb l'aplicació de generació de tasques en temps real, i també es fàcil la modificació. Si es volguessin fer proves amb el controlador dissenyat i canviar paràmetres d'aquests, simplement s'hauria de fer en la tasca corresponent, desar els canvis i tornar a generar per veure'n els resultats.

Per tant tenim la facilitat de crear i modificar un sistema de control per poder-lo analitzar.

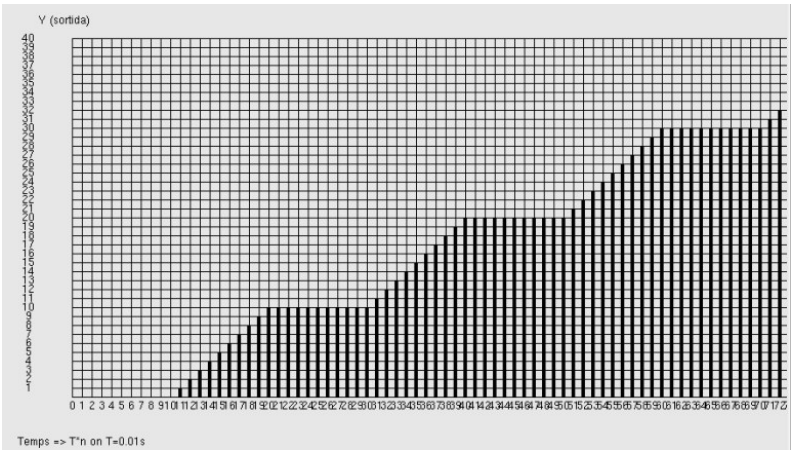


Figura 5.12*: Resultat Integrador*

5.3.3 Exemple 5: Control d'un tanc d'aigua (Temperatura i Nivell)

Per aprofitar les característiques de l'aplicació de generació de tasques desenvolupada, el que s'ha proposat fer és un sistema més complex. El sistema tractat és el del control de la temperatura i nivell d'un tanc d'aigua. Aquest sistema s'ha trobat en uns exemples de la Universitat Politècnica de València. S'han agafat les equacions de la simulació del sistema i s'ha afegit el control d'aquest sistema. El sistema controla per una part la temperatura de l'aigua mitjançant un sensor que mesura la temperatura de l'aigua per comparar-la amb una temperatura de referència i veure si s'ha d'introduir aigua calenta o no i deixar que l'aigua freda que cau contínuament refredi l'aigua del tanc. Per l'altre banda hi ha el control del nivell de l'aigua, si aquest nivell supera la referència s'haurà d'obrir la vàlvula per disminuir el nivell.

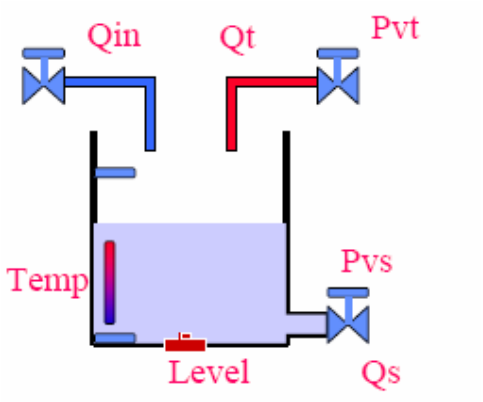


Figura 5.13: *Control d'un tanc d'aigua*

Aquest sistema de control té una complexitat prou alta per posar a prova les possibilitats que ofereix una aplicació com la de generar tasques.

Aquest tipus de control és bastant complex de controlar ja que si per exemple tenim que el nivell de l'aigua es baix s'haurà d'omplir però mirant si serà necessari tant sols amb l'aigua que cau freda o s'haurà d'activar l'aigua calenta per augmentar la temperatura d'aquest. S'ha de tenir en compte que els dos factors poden repercutir en el altre factor (temperatura, nivell). A més, les equacions de la simulació incorporen pertorbacions al sistema per veure que aquest reacciona ja que es un control en llaç tancat.

```
Qin = Qinput + (random - 0.5);
Qt = Qtm * Pvt;
Qs = cte2 * sqrt(Level) * Pvs;
F1 = (Qin * Te) + (Qt * Tc) + ((Qtotal - Qs) * Temp);
F2 = (Qin + Qt + Qtotal - Qs);
Temp = F1 / F2;
Level = Level + ((Qin - Qs + Qt) * cte1);
Qtotal = Qtotal - Qs + Qin + Qt;
```

Figura 5.14: *Simulació del sistema*

En la figura anterior es mostren les equacions del sistema, en la primera línia s'introdueix la pertorbació al sistema amb el valor random.

Per controlar aquest tipus de sistema s'utilitzarà el controlador. Un controlador PID (proporcional, integral, derivatiu) es un sistema de control, que mitjançant un actuador, es capaç de mantenir una variable o procés en un punt desitjat. Aquest tipus de controlador és dels mes usats en aquests tipus de control i per això s'ha escollit per controlar tal sistema.

Un controlador PID està dividida en la part diferencia, proporcional e integrador:

1. Part proporcional: Consisteix en el producte entre l'error del sistema mesurat i la constant proporcional. Aquest té importància quan l'error es gran. En la majoria dels casos aquest tipus de control no arriba a obtenir un error nul.

   $P = Kp * e_k$, on

   ek és l'error del sistema (la diferència entre la variable mesurada i la de referència)

2. Part integral: El mode de control integral permet eliminar l'error en l'estat estacionari. En aquesta acció de control l'error es integrat. El qual suma a un període de temps determinat i després multiplicat per una constant integral.

   $I_k = I_{k-1} + e_k * Ki$ (és fa l'acumulació del control anterior mes el producte de l'error per la constant del control integral)

3. Part derivativa: Corregeix l'error en el transitori, amb la funció de la derivada del error. Es deriva respecte el temps i multiplica per una constant derivativa i és sumada a la l'error anterior.

   $D = (e_k - e_{k-1}) * Kd$ (és fa la diferència entre l'error de l'estat actual i l'anterior per una constant derivativa)

Notar doncs que es necessari la sintonització de les constants de cada control per formar el PID òptim. Seguidament es veu com s'ha creat aquest sistema pas per pas amb el generador de tasques i posteriorment es veuran les proves fetes amb diferents sintonitzacions escollides per les constants PID i es veuran els resultats comentats.

Pas1: S'insereix la primera tasca, la tasca de simulació del sistema, per fer això s'afegeix la següent informació al apartat destinat a la informació de tasques.

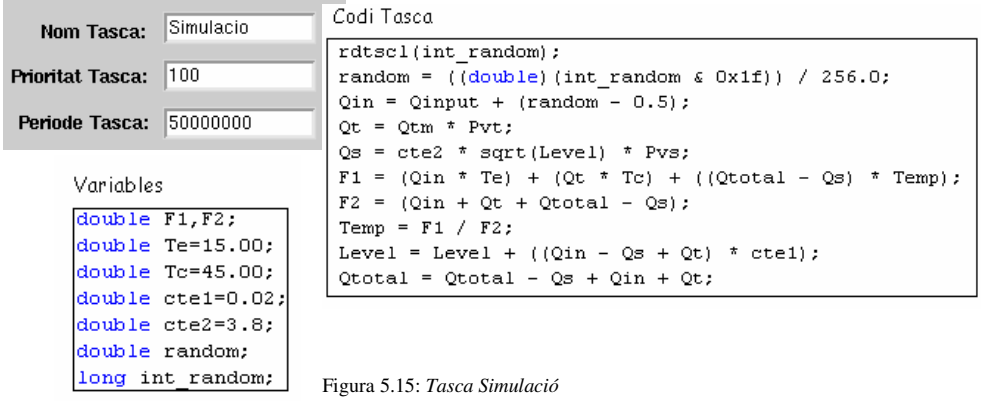


Figura 5.15: *Tasca Simulació*

Pas2: Un cop guarda la primera tasca es passa a inserir la tasca de visualització de la informació del sistema per poder monitoritzar la sortida.

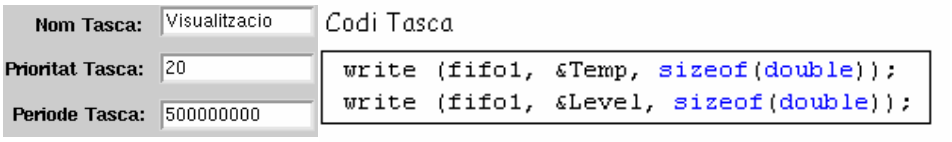


Figura 5.16: *Tasca Visualització*

Pas3: Després es passen a afegir les dos tasques de control, la de la temperatura i la del nivell. Aquests controls implementen el controlador PID.

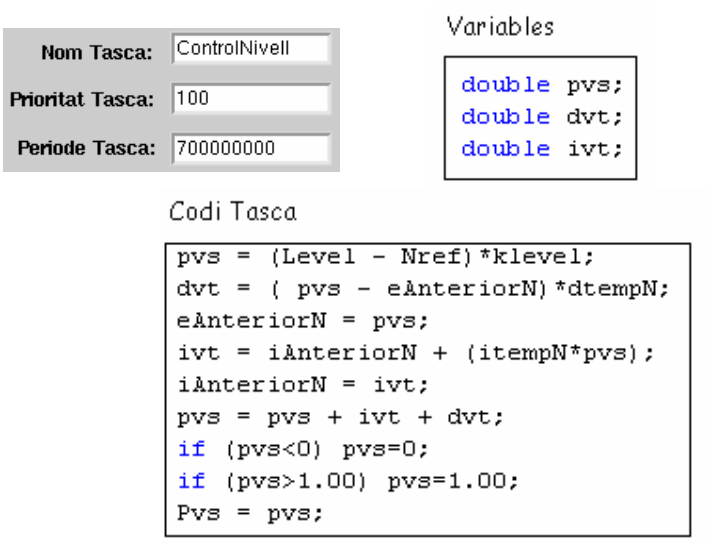


Figura 5.17: *Tasca Control del nivell.*

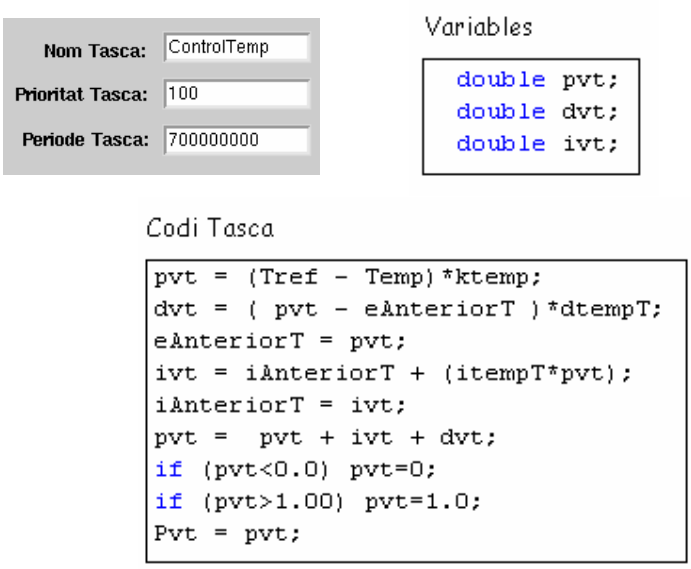


Figura 5.18: *Tasca Control de la temperatura*

Pas4: Un cop afegides totes les tasques tant sols quedarà crear la rt-fifo amb l'opció habilitada per crear rt-fifos, inserir les capçaleres necessàries i finalment les variables compartides.

Variables globals

```
//Declaracions variables compartides
extern double sqrt(double x);
static double Level= 0.00;
static double Temp = 15.00;
static double Qt ,Qtm = 16.00;
static double Qin, Qinput = 10.0;
static double Qs = 30.00;
static double Qtotal = 0.00;
static double Pvt = 0.0, Pvs = 0.0;

// Variables de control
static double klevel = 0.9;
static double ktemp = 0.5;
static double Nref=25.00;
static double Tref=18.50;

//Variables Controlador D
static double dtempT = 0.4;
static double dtempN = 0.1;
static double eAnteriorT = 0;
static double eAnteriorN = 0;

//Variables Controlador I
static double iAnteriorT = 0;
static double iAnteriorN = 0;
static double itempT = 0.7;
static double itempN = 1;
```

Figura 5.19: *Variables globals del sistema*

S'observa en la figura anterior que es defineixen les constants del control PID (constants P,I,D). Aquestes són les constants a sintonitzar per el correcte funcionament del control PID. Això es s'explica en els següents passos.

Per entendre una mica millor el codi inserit en el control aquí s'expliquen una mica les equacions utilitzades del PID:

Pas5: Un cop inserida tota la informació tant sols 'ha de generar i posar en marxa el sistema de temps real creat, això és fa premen el botó "Generar". Ara ja es podrà obtenir la sortida del sistema. Per veure'n la sortida s'han creat dues aplicacions per veure gràficament la resposta del sistema. El codi font de l'aplicació es troba en l'annex a l'apartat de codi font de les aplicacions. Un cop es té tot preparat ja es pot començar l'anàlisi de les respostes del sistema de control creat. El que s'ha fet, es donar diferents valors a les constants de control i veure'n el resultat i finalment escollir el que s'ha cregut millor. Tant sols s'ha anat canviant

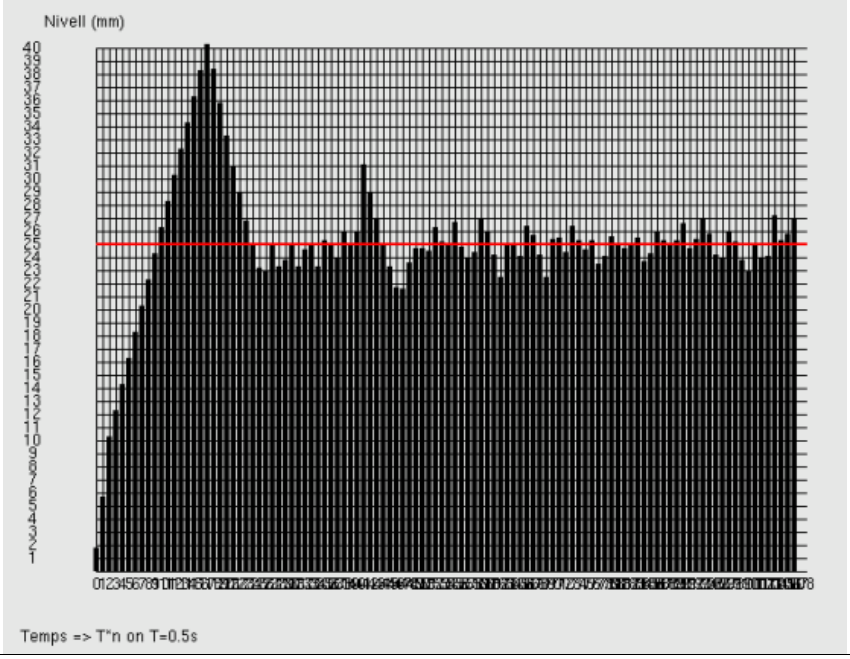
el valors de les constants de control que es troben al requadre de variables compartides i s'ha tornat a generar el codi per veure'n la resposta.

Les següents proves s'han fet amb un valor constant de la constant del control proporcional amb un valor de **Kp = 0.5**.

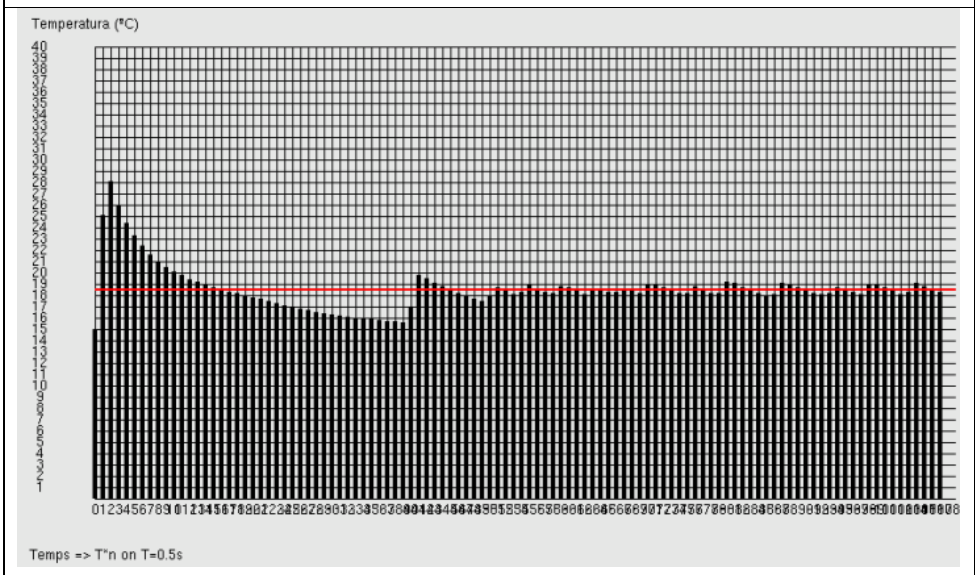
Sintonització 1

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0.9
Constant Derivativa (Kd) Temperatura	0.5
Constant Integral (Ki) Nivell	0.1
Constant Derivativa (Kd) Nivell	0.1

Resultat (Nivell)



Resultat (Temperatura)

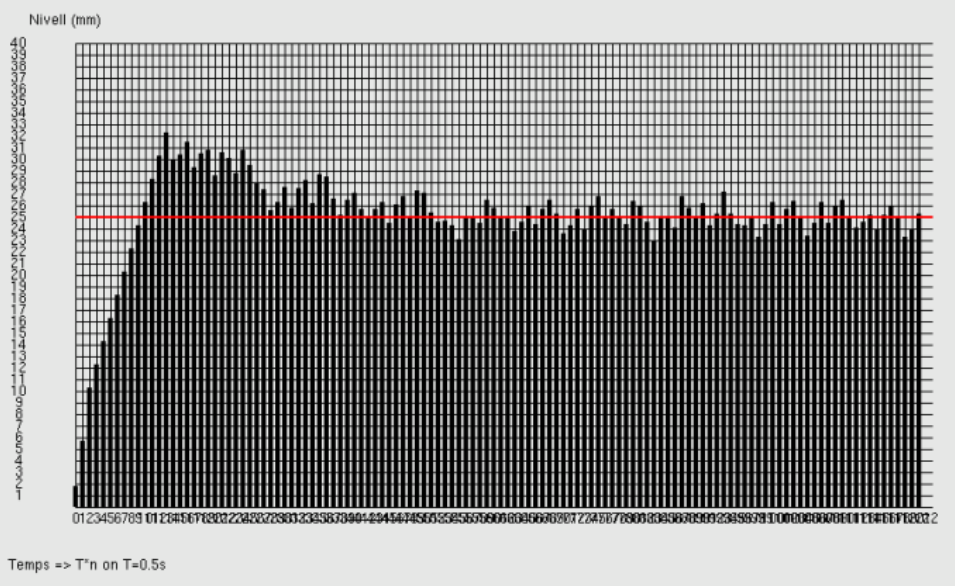


En aquesta prova els valors de la sintonització del derivatiu pel cas de la temperatura dóna una resposta una mica brusca. Això es degut a la constant derivativa, s'hauria de baixar per donar la resposta més suau. Per la part de l'integrador la resposta l'estableix però encara té algunes fluctuacions que s'haurien d'eliminar. En el control del nivell s'haurà d'ajustar millor ja que dona una resposta molt forta al principi i després varies fluctuacions respecte al valor que es vol obtenir del nivell.

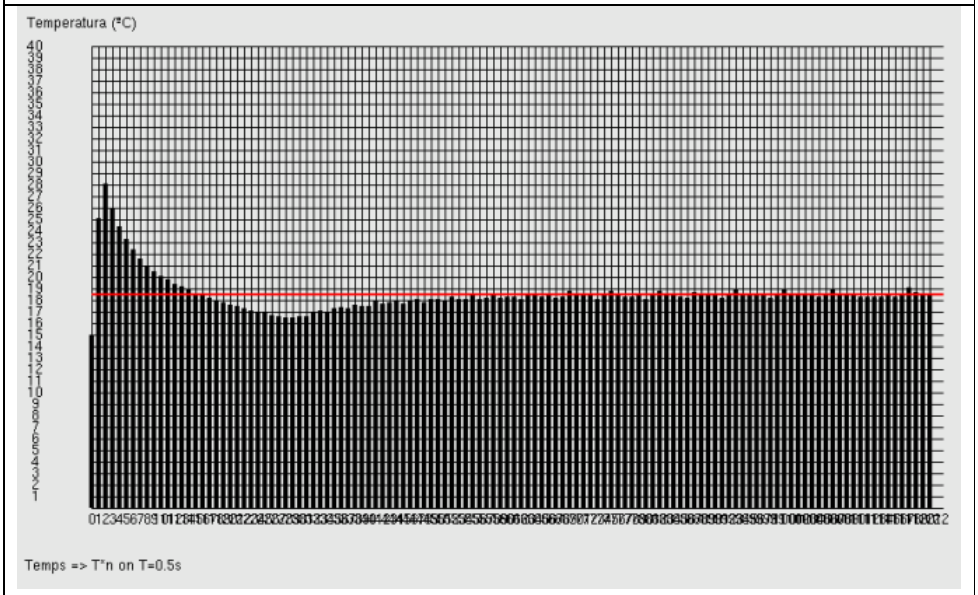
Sintonització 2

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0.1
Constant Derivativa (Kd) Temperatura	0.1
Constant Integral (Ki) Nivell	0.4
Constant Derivativa (Kd) Nivell	0.4

Resultat (Nivell)



Resultat (Temperatura)

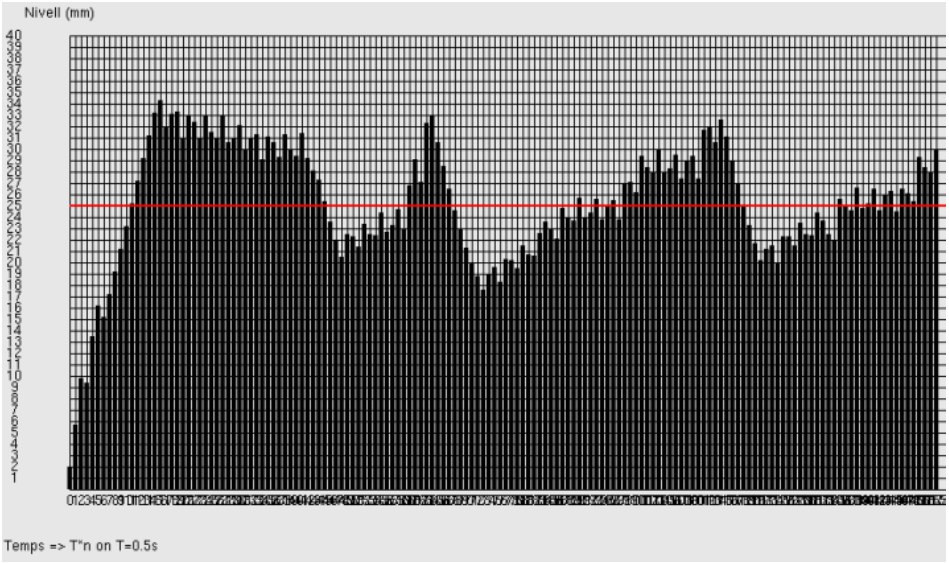


En aquesta prova es pot veure com la sintonització de les constants derivatives són valors baixos i s'obté aleshores una resposta suau al principi i seguidament l'integrador dóna una bona resposta també en el estacionari. Aquesta encara no és la sintonització bona. Aquests comentaris s'han fet mirant la gràfica de la temperatura. La que s'ha notat que costa mes obtenir un control nítid es el del control del nivell, de totes les proves fetes, aquesta es potser la que obté millors resultats pels dos controls. El control del nivell és irregular a causa de la pertorbació externa afegida a la simulació, però es comprova que tot i així aquest sistema reacciona davant aquesta. La pertorbació es la que fa que el sistema estigui contínuament controlant el procés.

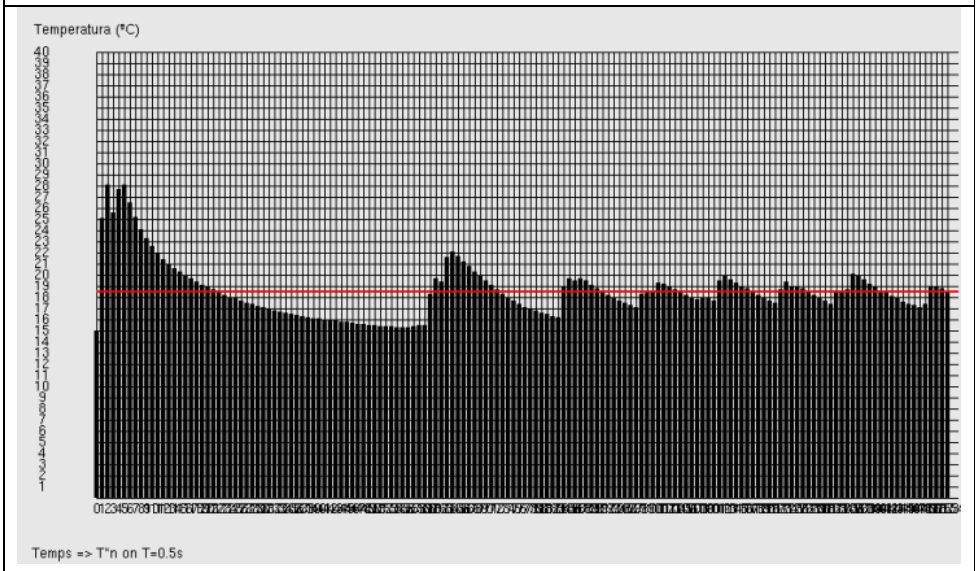
Sintonització 3

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0.9
Constant Derivativa (Kd) Temperatura	0.5
Constant Integral (Ki) Nivell	2
Constant Derivativa (Kd) Nivell	2

Resultat (Nivell)



Resultat (Temperatura)

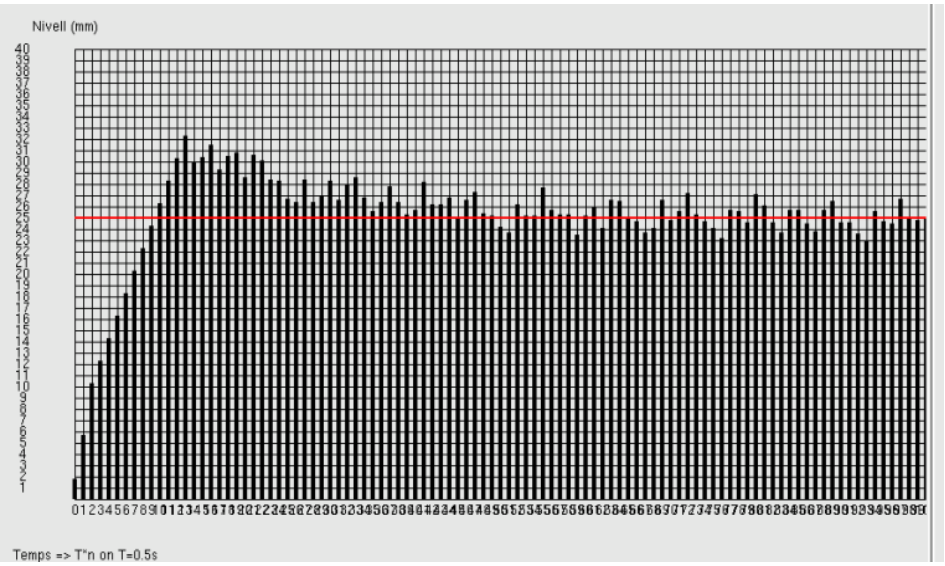


En aquesta prova s'han augmentat els valors de les constants derivatives per veure com afecta en la resposta del sistema. Els valors de l'integrador s'han deixat igual que en la primera sintonització. El resultat d'això és una resposta al transitori molt més forta encara que en el primer exemple i això farà que tardi molt més en arribar en un estat estacionari i es pugui notar l'acció del control integrador. Potser s'hauria d'augmentar les constants del control integrador per poder estabilitzar la sortida en l'estacionari.

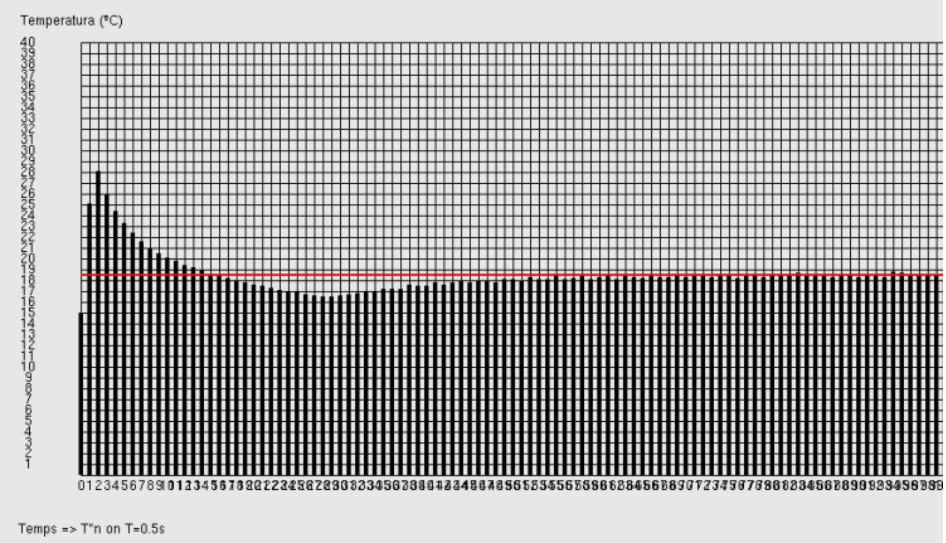
Sintonització 4

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0.2
Constant Derivativa (Kd) Temperatura	0.2
Constant Integral (Ki) Nivell	0.1
Constant Derivativa (Kd) Nivell	0.1

Resultat (Nivell)



Resultat (Temperatura)

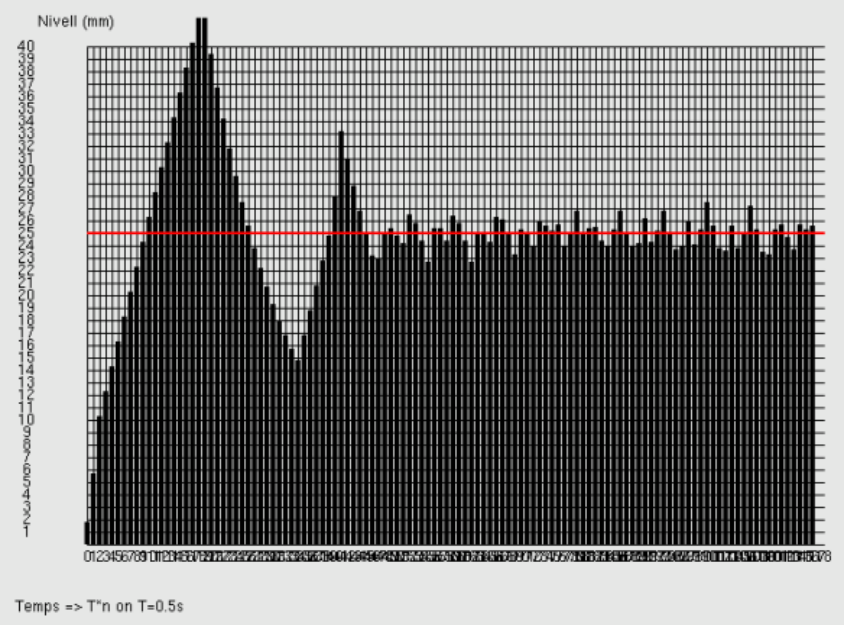


Fins ara la millor configuració que s'ha trobat és la de la segona sintonització, ara aquí s'ha disminuït el valor de la constant derivativa de 0.4 a 0.1. La sortida del sistema és molt semblant a la de la sintonització 2 però sembla ser que la resposta és una mica més suau encara. Es pot dir que aquests són de moment els valors de sintonia millor trobats.

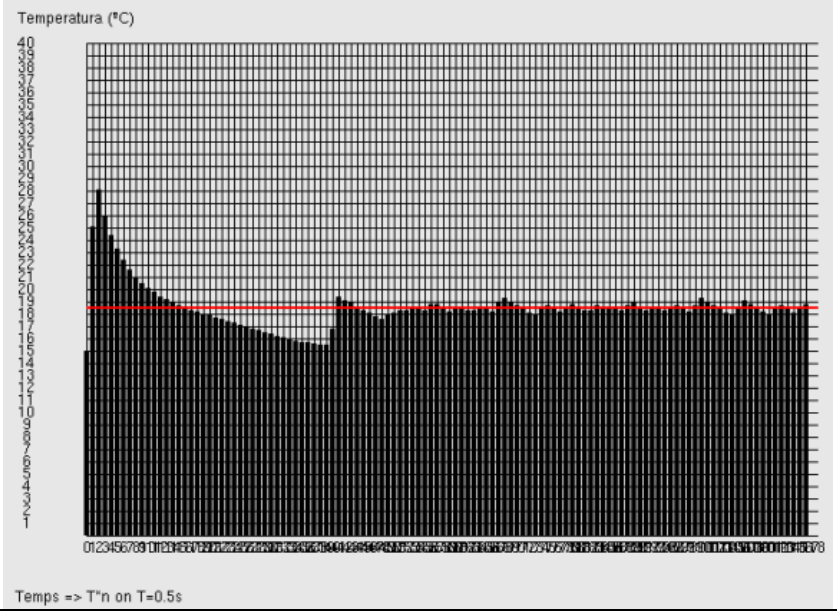
Sintonització 5

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0.2
Constant Derivativa (Kd) Temperatura	0.2
Constant Integral (Ki) Nivell	1
Constant Derivativa (Kd) Nivell	1

Resultat (Nivell)



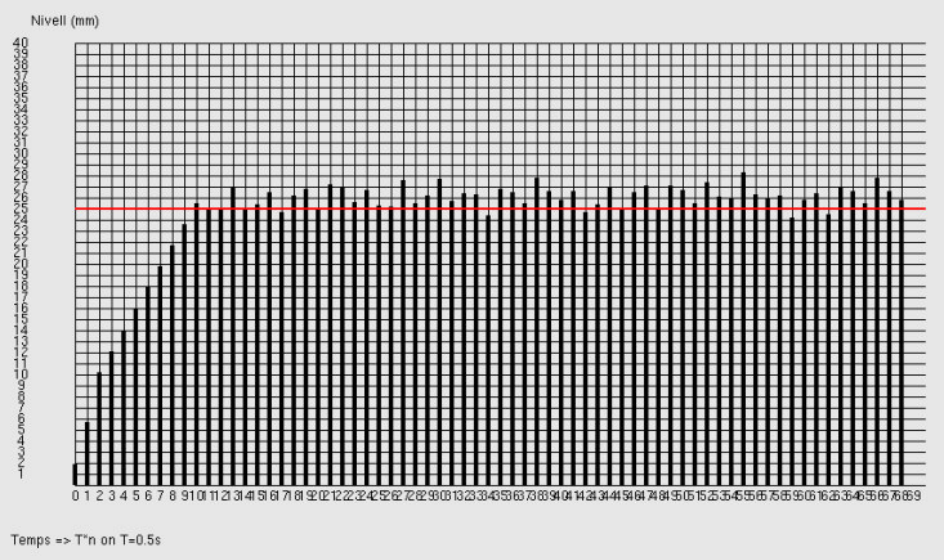
Resultat (Temperatura)



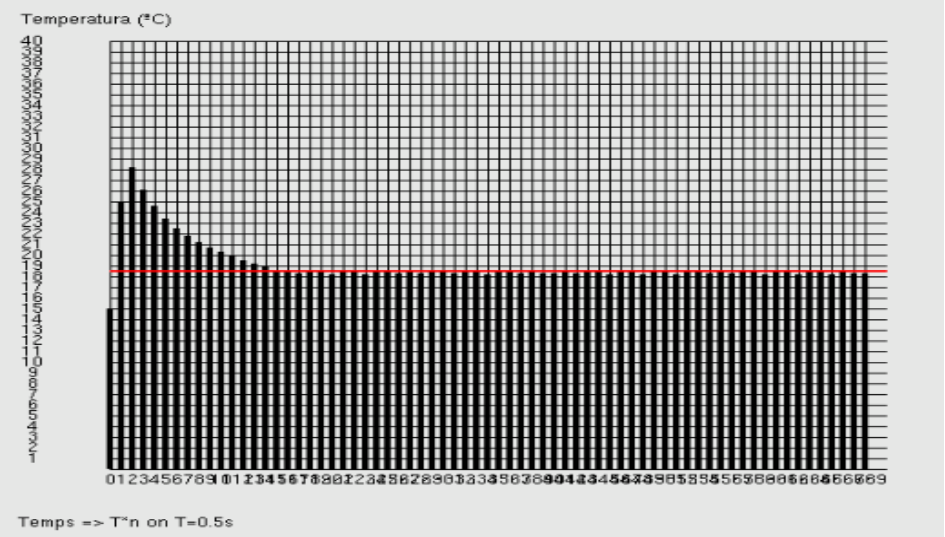
Sintonització 6

Constant de Control	Valor
Constant Integral (Ki) Temperatura	0
Constant Derivativa (Kd) Temperatura	0
Constant Integral (Ki) Nivell	0
Constant Derivativa (Kd) Nivell	0

Resultat (Nivell)



Resultat (Temperatura)



En la "Sintonització 5", es passa a augmentar el valor de la constant integrador i es comprova com el valor en l'estacionari fluctua i no acaba d'assolir la variable de referència. Aquest augment en la constant també afecta al valor transitori donant una resposta més forta al principi.

Un cop fetes aquestes proves s'ha conclòs que com més tendeixen les constants de control I, D a 0 millor és el resultat. Per tant d'aquí s'extrau que el millor control a efectuar en aquest sistema seria amb les constants derivativa i integral igual a 0 i això significarà, que tant sols amb el control proporcional és té prou per obtenir una bona resposta del sistema. Aquesta sintonització es troba en "Sintonització 6".

Caldria doncs estudiar amb mes profunditat perquè s'obtenen millors resultats sense control Integrador i Derivatiu. El sistema és mes complex que el tipus de sistemes explicats anteriorment. El sistema de control és de tipus MIMO i no SISO. Els tipus de sistemes estudiats normalment són els SISO (single input single output), tenen una entrada i una sortida. En canvi els sistemes MIMO (com el de l'exemple) tenen varies entrades i varies sortides. L'exemple té el control de la temperatura i el control del nivell, aquests dos controls són dependents entre ells ja que per exemple si la temperatura s'ha d'augmentar, s'haurà d'afegir aigua calenta i això repercutirà en el nivell del tanc. Per tant tenim una dependència entre els dos controls, això fa que el sistema a controla sigui més complex, del tipus MIMO. Amb un sistema del tipus SISO de ben segur que dissenyant un control PID com el que s'ha dissenyat, s'hagués obtingut uns resultats propers a l'error 0 en el control però en els sistemes MIMO no és tan fàcil. Seguidament es mostra una il·lustració amb esquemes dels dos tipus de sistemes.

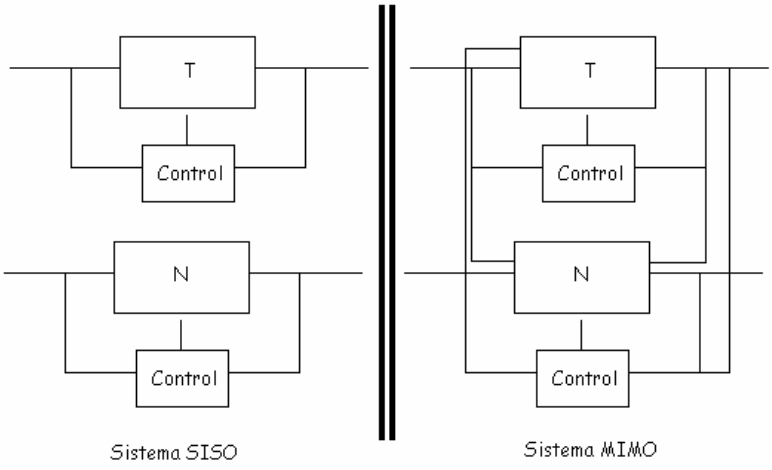


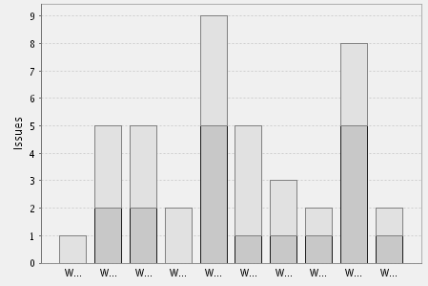
Figura 5.20: *Tipus de sistemes*

A part de tenir un sistema de control complex, el seu comportament no és lineal, cal veure la funció que simula el sistema. Aquest sistema conté l'arrel quadrada, aquest aspecte fa que el sistema no sigui lineal.

Aquestes són les varies raons concloses pel que fa al control PID del exemple. Sinó es sintonitzen bé les constants es pot obtenir pitjor resultats. El control PID anomenat en aquesta memòria es per a sistemes tipus SISO estudiats en la present enginyeria. Per això s'obté millors resultats simplement amb el control proporcional.

Amb aquest últim exemple ha calgut crear un sistema complexe i fer-ne un estudi ampli per veure'n la resposta amb diferents configuracions. Tot això s'ha fet amb l'aplicació creada per generar tasques. Ha estat de gran utilitat tot i que tant sols és una versió inicial. Val a dir doncs que s'ha creat una eina senzilla d'utilitzar i amb unes opcions potents per crear i analitzar sistemes de tot tipus. Però la gran importància que té tot això es que després de fer les proves sobre un sistema simulat en el que es controla tot (l'execució de les tasques que controlen el sistema, el control del sistema, etc.) tant sols treïent la tasca que simula el sistema i modificant les tasques de control per obtenir les dades des dels sensors ja s'obtindria el sistema de control en temps real. Amb l'exemple anterior, tant sols agafant-lo i modificant els controladors per obtenir les dades dels sensors del sistema es podria agafar i posar-ho en la planta real i s'obtindrien els resultats ja simulats i analitzats, ja que s'ha tingut la possibilitat de controlar l'ordre d'execució de les tasques de control.

Aquest és l'objectiu del projecte, provar la viabilitat de crear un entorn de desenvolupament de sistemes en temps real. Un entorn ha de poder desenvolupar sistemes de control i poder-los analitzar de forma senzilla i mantenint el codi per a la planta real i això es el que s'ha aconseguit. Un cop arribat aquest punt tant sols queda pensar, dissenyar i desenvolupar diferents eines per funcionalitats necessàries per l'usuari del sistema operatiu de temps real.



6

Conclusió

---

6. Conclusió

Aquest projecte final de carrera ha tingut la intenció d'estudiar els sistemes en temps real per tal de crear un entorn de desenvolupament d'aplicacions per temps real sota el sistema operatiu RTLinux. Són molts els sistemes que cada cop són més sofisticats i tenen la necessitat de ser controlats per sistemes en temps real tal com sistemes de control d'aviació, de les aeronaus enviades al espai, etc. És un tema amb molt contingut i encara queda molt per explotar. En aquest projecte s'ha estudiat com funcionen aquests tipus de sistemes i s'ha fet possible obtenir-ne un, en un computador personal. El Sistema Operatiu instal·lat ha estat RTLinux després de fer-ne un anàlisis entre diferents Sistemes Operatius en Temps Real. Tot aquest anàlisis s'ha fet sota el requeriment de que siguin de codi lliure. S'ha configurat a mida, s'ha estudiat com funciona el sistema operatiu i finalment s'ha après a programar en la API de temps real. En aquest punt ja es tenia preparat un sistema operatiu en temps real i la necessitat ara radicava en elegir un llenguatge de programació per crear-ne aplicacions per facilitar la tasca al usuari final. El llenguatge utilitzat ha estat el Tcl/Tk per ser de codi lliure i per la seva facilitat d'aprendre i a la vegada per la seva potència. S'ha estudiat com programar amb aquest llenguatge i una vegada s'ha obtingut una certa experiència s'ha començat a dissenyar eines pel desenvolupament d'un entorn per RTLinux. Amb la creació d'una eina per generar tasques automàticament s'ha comprovat la viabilitat per a crear un entorn complex i fàcil d'utilitzar per a la creació d'aplicacions en temps real. Amb el generador de tasques, una aplicació mitjanament complexa s'ha vist la potència que es pot extreure d'un sistemes operatiu i unes eines de programació de codi lliure. Amb el generador de tasques s'ha provat de crear tasques de control i s'ha pogut crear de forma fàcil i flexible. Un exemple clar ha estat el control de temperatura i nivell d'un tanc d'aigua, s'ha vist la possibilitat de crear les tasques de control de forma ràpida i senzilla amb el generador de tasques. El generador de tasques no només té l'avantatge d'una creació ràpida i senzilla sinó que té la fàcil possibilitat de modificació d'aquestes tasques i la posterior posada en marxa. Cal recordar al crear els controladors de temperatura i nivell del tanc d'aigua, que alhora de dissenyar el control es necessitava obtenir la constant de control tant del controlador proporcional com el derivatiu i el integrador. No ha estat necessari tocar codi font i compilar directament, a través de l'aplicació s'ha canviat les dades necessàries i s'ha torna a generar el codi per posteriorment veure'n el resultat. A més a més si es vulgues aplicar el control implementat a una planta real, el codi es mantindria i simplement s'hauria de canviar la forma en que s'obtenen les dades (ara a través dels sensors). Per tant s'ha obtingut una eina prou útil amb l'ús de software lliure .

Després de veure'n les possibilitats que pot donar el temps real, RTLinux i el software lliure, tant sols queda estudiar les necessitats de l'usuari que utilitzi aquests tipus de sistemes i començar a dissenyar e implementar eines per la confecció d'un entorn complerts per la creació, simulació i anàlisis de tasques en temps real per al control de sistemes complexes. Aquest entorn podria ser útil per a la creació de tasques en temps real per a molts camps com la robotica , automobilístic, control aeri, etc. La major part de les implementacions serien per a sistemes de control empotrats. Com es pot veure, serien moltes les sortides que tindria la creació de tal entorn. Aquest projecte a quedat com un estudi preliminar, des del disseny e implementació de varies aplicacions per augmentar la utilitat de l'entorn a crear fins a la confecció de mòduls modificats pel Sistema Operatiu en Temps Real RTLinux, tal com la modificació del planificador

en temps real per exemple, ja que RTLinux ens permet modificar tots els aspectes del Sistema Operatiu cosa que un Sistema Operatiu comercial no ho permetria.

6.1 Objectius assolits

S'ha complert tots els objectius proposats al inici del projecte final de carrera, és més, s'han proposat de nous per a la confecció de nous projectes ja que el tema del temps real dona per molt. Seguidament s'enumeren els objectius en punts explicats breument.

1.- Estudi inicial dels sistemes en Temps Real ✔

Inicialment l'objectiu a complir era comprendre el terme de sistema en temps real i la utilitat d'aquests tipus de sistemes. En aquest punt es comprova la necessitat i la envergadura d'aquests tipus de sistemes en l'actualitat. S'entra en detall en aspectes de rendiment. Aquest objectiu era obligat assolir per poder fer aquest projecte final de carrera.

2.- Anàlisis dels Sistemes Operatius en Temps Real (elecció d'un SO) ✔

Un cop estudiats els sistemes en temps real, el pròxim objectiu complert ha estat l'anàlisi de diferents sistemes operatius en temps real de codi lliure per elegir-ne un. S'analitzen unes quantes de les moltes distribucions que existeixen i s'acaba elegint RTLinux per les seves característiques i documentació.

3.- Instal·lació i Configuració de RTLinux ✔

Instal·lació del kernel de RTLinux sobre el kernel de linux i la posterior configuració per al propòsit del projecte. En aquest punt es troben problemes alhora d'instal·lar i és necessari crear un bon manual pas per pas per a futures instal·lacions, inclòs del lector.

4.- Provar que s'ha obtingut un Sistema Operatiu en Temps Real ✔

Fer varies proves per analitzar si el Sistema Operatiu RTLinux instal·lat compleix els requisits d'un Sistema Operatiu en Temps Real i veure'n el rendiment.

5.- Aprendre a utilitzar RTLinux i programar amb la API de temps real ✔

Un cop s'ha provat que RTLinux compleix els requeriments necessaris es comença a aprendre com utilitzar RTLinux i programar la API de temps real que proporciona. Així es comença a veure quina es la estructura dels mòduls en temps real pel generador de tasques.

6.- Elegir llenguatge de programació per el desenvolupament d'aplicacions i el posterior aprenentatge ✔

S'elegeix el llenguatge Tcl/Tk per desenvolupar-ne les aplicacions, s'arpen a utilitzar aquest llenguatge i obtenir-ne una certa experiència per començar a crear eines per RTLinux.

7.- Desenvolupament d'aplicacions ✔

Es dissenyen e implementen aplicacions per facilitar la tasca de generació i anàlisis de tasques en temps real amb el llenguatge Tcl/Tk. Es crea el generador de tasques per a facilitar la tasca de creació i modificació de tasques en temps real (versió limitada, possibles millores comentades mes endavant)

8.- Exemples per extreure conclusions ✔

Es creen exemples tant de tasques en temps real per comprovar les interrupcions i el planificador de temps real com exemples creats amb l'aplicació desenvolupada anteriorment per generar tasques de control en temps real. En aquest punt es comenten els resultats obtingut i es conclou en que s'ha obtingut un entorn inicial d'eines pel suport al desenvolupament de tasques en temps real sota l'ús constant de software lliure.

9.- Generar idees per a la creació d'un entorn complert ✔

Un cop assolits els objectius es proposen noves idees que generaran nous objectius. El projecte el que cerca es la possibilitat de crear un entorn pel desenvolupament de tasques en temps real utilitzant software lliure. Un cop assolit aquest objectiu tant sols queda generar idees per la creació d'un entorn complert. Aquestes idees poden arribar a generar nous projectes finals de carrera tal com creació de diferents aplicacions o modificació del propi RTLinux per millorar o personalitzar aspectes del propi Sistema Operatiu en Temps Real. Algunes d'aquestes idees estan exposades en aquest mateix capítol, més concretament en el punt 6.4.

6.2 Problemes trobats

Durant la realització del projecte han estat diferents els problemes trobats i que s'han anat solucionant. Inicialment el problema trobat ha estat un problema conceptual, no es tenia clar el concepte de temps real. Les explicacions no quedaven clares del tot però amb la experimentació amb RTLinux i els conceptes obtinguts s'ha anat entenent poc a poc cada cop millor el concepte. El concepte de sistema en temps real no és un concepte fàcil d'entendre, més aviat complex i molts cops es pot arribar a confondre com ja s'ha comentat en aquesta memòria.

Després també es va tenir problemes alhora d'instal·lar RTLinux, ja que cada manual d'instal·lació ho explicava a la seva manera i es van haver de posar en comú diferents manuals per a la instal·lació i configuració de RTLinux, es va arribar a perdre més d'una vegada el sistema complet. Un cop aconseguida la instal·lació ràpidament es va generar un manual per punts per a la instal·lació de RTLinux sota la distribució Mandrake per si és vol repetir la instal·lació o el lector mateix la vol portar a terme sense massa dificultat ni coneixements amplis de linux.

Un altre dels problemes trobats ha estat alhora de programar en RTLinux i tota la dificultat que pot portar aprendre una llibreria de programació al igual que aprendre a programar amb el llenguatge Tcl/Tk.

També s'han anat trobant problemes alhora d'implementar aplicacions, en fer les proves, etc. però s'han anat solucionant sense massa dificultat.

Personalment crec que ha estat un projecte final de carrera bastant complert ja que està compost per una bona part teòrica (sistemes en temps real, sistemes operatius en temps real, rtlinux, tcl/tk, etc.) i la part pràctica (implementació d'aplicacions, exemples, resultats, etc.) Com tot estudi/anàlisi inicial, que podríem dir que és el que s'ha fet amb aquest projecte, és complex al principi ja que entren en joc molts conceptes nous que s'han d'analitzar i provar.
Un cop resolts tots els problemes i complert els objectius es podrà pensar idees per a projectes futurs que sentint les seves bases en el que s'ha estudiat en aquest projecte final de carrera.

6.3 Propostes de treball futur

Un cop assolit els objectius proposats al principi i s'ha conclòs que realment es pot arribar a crear un entorn de desenvolupament de sistemes en temps real amb les eines escollides, és podrien fer multitud d'aplicacions i modificacions del propi RTLinux.
Són moltes les opcions que hi ha per continuar aquest projecte.
Per començar, l'aplicació de generació de tasques implementada en aquest projecte per comprovar la viabilitat de la construcció d'aplicacions per RTLinux de forma ràpida i obtenint aplicacions complexes, aquesta és podria millorar en els següents aspectes:

- Es podria permetre la creació de N real time fifos en comptes de només dues.
- És podria generalitzar molt més la construcció de tasques afegint la possibilitat d'escollir per cada tasca si es vol una tasca periòdica o aperiòdica.
- També es podria afegir la possibilitat d'afegir interrupcions en les tasques, personalitzar el mòdul d'iniciació, el mòdul de finalització, etc.
- Personalització mòdul inici i fi.
- Inserir interrupcions en la creació de tasques.
- Millora en la gestió de les tasques (inserir, eliminar, modificar tasques)
- Donar la possibilitat de guardar les tasques en un fitxer XML per poder obrir en altres instants.

Una altre de les aplicacions creades ha estat la de mostrar l'execució de les tasques en el planificador de temps real. Aquesta aplicació mostra quan les tasques s'executen però no mostra quan una tasca està activada. Per tant, la millora a realitzar en aquesta aplicació seria permetre visualitzar l'activació de les tasques i el bloqueig. Tant sols amb la informació de l'execució de les tasques no n'hi ha prou per fer-ne un anàlisis exhaustiu del que està passant en el planificador.
Mostrant l'activació de les tasques i el bloqueig es tindrà més informació per un bon anàlisis.
Amb aquestes dos millores de les aplicacions ja es tindria un conjunt d'eines per al estudi de tasques en temps real a través de RTLinux. Recordar que la modificació de tasques amb l'aplicació de generació de

tasques és realment ràpid i fàcil de fer i això permet fer un estudi ampli amb el suport de l'eina de visualització de l'execució de les tasques.

A part de la millora de les aplicacions ja existents és podrien crear diferents aplicacions per a millorar i ampliar l'entorn de desenvolupament, tant aplicacions per a la gestió de les tasques en temps real com aplicacions per el anàlisis. Aquest podria ser motiu per un projecte nou, partint de la base de l'estudi realitzat en el present projecte és podria pensar, dissenyar e implementar eines per la millora de l'entorn de desenvolupament de sistemes en temps real.

Un altre tema a tractar serà la modificació del propi Sistema Operatiu en Temps Real RTLinux, per exemple un altre tema que podria ser motiu de projecte seria l'estudi en profunditat dels planificador en temps real per implementar-los sota RTLinux i fer les proves pertinents per analitzar quin dels planificadors seria l'adient (FIFO, Round Robin, Earliest Deadline First, etc..). El tema del planificador està bastant poc desenvolupat en RTLinux i es podria estudiar per fer varies implementacions i millorar-lo. El planificador RTLinux és de prioritats estàtiques com ja s'ha explicat, doncs una possible implementació seria la creació d'un planificador amb prioritats dinàmiques.

Altres treballs a realitzar seria la creació o modificació de mòduls de temps real, existents (fifo, posix, etc.) o de nous (network, comunicació, hardware). Un bon projecte a realitzar podria ser l'estudi d'un mòdul en temps real de comunicació basat en el protocol IP. El mòdul s'anomena RTL-lwIP (lightweightIP). Un mòdul que permetrà la comunicació TCP/IP amb un mínim requeriment de recursos. RTL-lwIP inclou els protocols IP, ICMP, UDP i TCP. Aquest projecte analitzaria i provaria el mòdul en temps real el qual seria de molta utilitat per a la comunicació de tasques en temps real. Si el mòdul analitzat fos útil es podria integrar en el sistema creat en aquest projecte i afegir utilitats a les aplicacions creades per millorar la gestió de la comunicació entre tasques de temps real. Seria una gran millora poder disposar de comunicació entre tasques. Això permetria la possibilitat del control remot de tasques en temps real. Un altre mòdul existent s'anomena rtsock. Imaginem el següent escenari, un controlador de qualsevol procés que es vol controlar remotament, amb aquest mòdul ho podríem aconseguir. A mes a mes es podria modificar el propi mòdul per afegir millores en la comunicació. Personalment crec que aquesta última proposta és especialment interessant per a realitzar un projecte final de carrera.

6.4 Valoració personal del Projecte Final de Carrera

Personalment he trobat interessant el tema tractat en aquest projecte, en cada avanç que s'ha fet, la motivació ha crescut. Crec que l'elaboració d'un projecte final de carrera per part de l'estudiant és una bona forma de posar en funcionament tot el que s'ha après durant els anys d'estudiants abans d'arribar a aquest tràmit, abans de posar fi al estudis, encara que els estudis mai s'acaben per un enginyer. Valoro molt positivament l'elaboració d'un projecte final de carrera ja que entren en joc els aspectes més importants que l'estudiant es trobarà en la vida real alhora de treballar. El projecte realitzat consta dels següents passos: Un estudi/anàlisis del tema tractat, la implementació, uns resultats i unes conclusions. Un cop fet tot això l'estudiant en genera una documentació en forma de memòria per reflectir tot el que s'ha fet durant el projecte i finalment es fa un resum d'aquest document per presentar-lo. Cal dir doncs que l'estudiant posa a prova la propietat d'aprendre, analitzar nous temes, escriure tot el que ha fet durant el projecte i presentar-ho. Tot aquests aspectes s'han posat a prova en aquest projecte. Valoro positivament el treball realitzat, ja que en un inici no és tenia moltes nocions sobre el tema dels sistemes en temps real, RTLinux, etc. i s'ha aconseguit obtenir un petit entorn de desenvolupament de tasques en temps real a part d'entendre com funciona tot aquests tipus de sistemes. Crec que s'ha tractat un tema bastant interessant i amb moltes possibilitats de futur ja que el camp del temps real encara no està prou explotat per les gran empreses del software en l'actualitat i donarà molt a parlar en els pròxims anys. Els sistemes operatius més famosos, Windows/MAC-OS no disposen de bones tècniques de temps real, encara que s'anuncien en algunes les característiques però de ben segur que en un futur pròxim en sentirem més a parlar.

Personalment valoro l'esforç realitzat per a portar a terme aquest projecte final de carrera ja que m'ha suposat una carrega molt alta per causa de projectes externs als estudis (treball). M'ha ajudat a aprendre a planificar el temps en diferents períodes per poder realitzar diferents projectes i poder-los dur a terme. Com ja s'ha dit a estat bastant dur però un cop arribat al final personalment es valora molt. Si el temps ho hagués premés s'hagués profunditzat molt més en les aplicacions creades, en els exemples de controladors PID (Proporcional, Integral, Derivador) i sobretot m'hagués agradat molt haver pogut fer proves de comunicació entre tasques mitjançant TCP/IP. De ben segur que fora de l'àmbit d'aquest projecte es faran proves amb aquests tipus de mòdul en temps real i crear eines per a la gestió de comunicació mitjançant sockets per a tasques en temps real ja que és un tema realment interessant de tocar.

Inicialment al escollir aquest projecte, al no tenir massa idea de que era el temps real, una de les úniques motivacions que tenia era el repte d'afrontar temes nous, com ja s'explica en el capítol introductori d'aquesta memòria, però a mesura que avançava el temps i s'assimilaven els conceptes i es feien les proves, les motivacions creixien fins al punt que al acabar aquí el projecte no n'he tingut prou i tinc la necessitat de fer més proves e intentar realitzar alguns dels treballs futurs proposats en el punt anterior.

Per tant, després del gran esforç i de les dificultats que ha presentat el projecte, al finalitzar aquest s'ha quedat amb les ganes de seguir endavant.

En resum, es pot dir que ha estat una experiència molt positiva encara que ha suposat molt d'esforç fins arribar alguna vegada al límit de la desesperació per no poder complir els plaços temporals imposats personalment. També considero que aquesta és potser la gràcia del projecte que fa que cada dia es presentin nous reptes en el desenvolupament del projecte.

Aquí arriba tant sols el final del principi, dels estudis de la meva vida.

7

Annex

7. Annex

7.1 Codi font aplicacions

(i) Generador de Tasques

Codi Procediment Generació capçalera

```
##########################################################################
####
## Procedure:  GenerateHeader

proc ::GenerateHeader {out n rtf0 rtf1} {
#Includes i capçaleres
global includes
global returns
global declareThread
global pointandcoma
#set includes "#include <rtl.h> \n#include <time.h> \n#include
<pthread.h>"
set returns "\n\n"
set pointandcoma ";"
puts $out "$includes"
puts $out ""
puts $out "//Declaracions variables compartides"
puts $out [Text3 get 0.0 end]
#Declarem rtfs..
if {$rtf0 == 1} {
    puts $out "int fifo0;"
    }
if {$rtf1 == 1} {
    puts $out "int fifo1;"
    }

}
```

Codi Procediment Generació Tasca

```
##########################################################################
####
## Procedure:  GenerateTasc

proc ::GenerateTasc {out tasca i} {
#Per a cada tasca
global nameRoutine
global argsRoutine
global int
global structSchedParam
global structSchedParam_priority
global hrtime_t
global setSchedParam1
global makePeriodic
global period
global ret
```

```
global closee
global opeen
global whilee
global npPeriodic
global pointandcoma

set pointandcoma ";"
set nameRoutine "void *start_routine"
set argsRoutine "(void *arg)"
set int "int"
set structSchedParam "struct sched_param p"
set structSchedParam_priority "p.sched_priority="
set hrtime_t "hrtime_t now"
set setSchedParam1 "pthread_setschedparam(pthread_self(),SCHED_FIFO,&p)"
set period "unsigned long period = "
set makePeriodic "pthread_make_periodic_np(pthread_self(),gethrtime(),"
set makePeriodic1 ")"
set ret "return 0"
set closee "\}"
set opeen "\{"
set whilee "while(1)"
set npPeriodic "pthread_wait_np()"


            set prio [lindex [lindex $tasca 0] 0]
            set period [lindex [lindex $tasca 0] 1]
            set code [lindex [lindex $tasca 0] 2]
            set vars [lindex [lindex $tasca 0] 3]
            set name [lindex [lindex $tasca 0] 4]

            puts "Prio: $prio \n"
            puts "Period: $period \n"
            puts "Code: $code \n"
            puts "Vars  $vars \n"

        puts $out "$nameRoutine$name$argsRoutine"
        puts $out "$opeen"
        #puts $out "$structSchedParam$pointandcoma"
        puts $out "$vars"
        #puts $out "$structSchedParam_priority$prio$pointandcoma"
        #puts $out "$setSchedParam1$pointandcoma"
        puts $out "$makePeriodic$period$makePeriodic1$pointandcoma"
        puts $out "$whilee$opeen"
        puts $out "$npPeriodic$pointandcoma"
        puts $out "$code"
        puts $out "$closee"
        #puts $out "$ret$pointandcoma"
        puts $out "$closee"
}
```

Codi per desar una Tasca (al prémer el botó "Guardar")

```
lblStatus configure -text "Saving..."
##Codi source de la tasca
set code [Text1 get 0.0 end]
#Declaració variables
set vars [Text2 get 0.0 end]
#prioritat de la tasca
set prioTasc $txtPrio
#periode de la tasca
set periodTasc $txtPeriode
#Nom de la tasca
set nameTasc $txtName
##########################################
#llista on tenim els 4 paràmetres anteriors
#llista :     -periode tasca
#             -id tasca
#             -prioritat tasca
#             -code
#             -Nom
##########################################
set dada "$prioTasc:$periodTasc:$code:$vars:$nameTasc"
set llista [split $dada :]

#Definim un array on contindrem totes les dades de les diferents tasques
set ArrayTasc($i) $llista

lblNtask configure -text [array size ArrayTasc]
lblStatus configure -text "none"
```

Codi per desplaçament Tasques a l'esquerra (botó "<<")

```
if { $i > 1 } {
   set i [expr $i-1]
   set tasca $ArrayTasc($i)
   set prio [lindex $tasca 0]
   set period [lindex $tasca 1]
   set code [lindex $tasca 2]
   set vars [lindex $tasca  3]
   set name [lindex $tasca 4]

   #Netejem formulari
   txtName1 delete 0 end
   txtPrio1 delete 0 end
   txtPeriode1 delete 0 end
   Text2 delete 0.0 end
   Text1 delete 0.0 end

   #insertem variables de $i
   txtName1 insert 0 $name
   txtPrio1 insert 0 $prio
   txtPeriode1 insert 0 $period
   Text2 insert 0.0 $vars
   Text1 insert 0.0 $code
   .top45.tit46 configure -text "Tasca $i"
   }}
```

Codi per desplaçament Tasques a la dreta (botó ">>")

```
if { [array size ArrayTasc] > $i} {
   set i [expr $i+1]
   set tasca $ArrayTasc($i)
   set prio [lindex $tasca 0]
   set period [lindex $tasca 1]
   set code [lindex $tasca 2]
   set vars [lindex $tasca  3]
   set name [lindex $tasca 4]

   #Netejem formulari
   txtName1 delete 0 end
   txtPrio1 delete 0 end
   txtPeriode1 delete 0 end
   Text2 delete 0.0 end
   Text1 delete 0.0 end

   #Insertem variables
   txtName1 insert 0 $name
   txtPrio1 insert 0 $prio
   txtPeriode1 insert 0 $period
   Text2 insert 0.0 $vars
   Text1 insert 0.0 $code
   .top45.tit46 configure -text "Tasca $i"

} else {

if { [array size ArrayTasc] == $i} {
        set i [expr $i+1]
        .top45.tit46 configure -text "Tasca $i"
        Text2 delete 0.0 end
        Text1 delete 0.0 end
        txtName1 delete 0 end
        txtPrio1 delete 0 end
        txtPeriode1 delete 0 end

}

}}
```

Codi per generar el fitxer de codi font de la tasca en temps real (botó "Generar")

```
set Ntasc [array size ArrayTasc]
puts "ola"
set out [open "code.c" w]
GenerateHeader $out $Ntasc $rtf0 $rtf1
puts $out ""
set nameThread "pthread_t thread"
set pointandcoma ";"
                for {set i 1} {$i <= $Ntasc} {incr i 1} {
                        set tasca $ArrayTasc($i)
                        set name [lindex $tasca 4]
                  puts $out "$nameThread$name$pointandcoma"
            }
puts $out ""
            for {set i 1} {$i <= $Ntasc} {incr i 1} {
                  GenerateTasc $out [list $ArrayTasc($i)] $i
}

set ret "return 0"
set initt "int init_module(void)\{"
set pthreadCreate1 "pthread_create("
set pthreadCreate2 ",&attr,"
set pthreadCreate3 ",0)"
set thread "&thread"
set routine "start_routine"
set pointandcoma ";"
set closee "\}"
set clean "void cleanup_module(void)\{"
set delete1 "pthread_delete_np("
set delete2 ")"
set thread1 "thread"
set schedParam "struct sched_param sched_param;"
set attr "pthread_attr_t attr;"
set attrInit "pthread_attr_init (&attr);"
set prio "sched_param.sched_priority ="
set sched "pthread_attr_setschedparam (&attr, &sched_param);"
set setfp "pthread_attr_setfp_np(&attr, 1);"


#Generate Init
puts $out "$initt"
set entra 0
if {$rtf0 == 1} {
    puts $out "int fifo_status;"
    set entra 1
}
if {$rtf1 == 1} {
    if {$entra == 0} {
        puts $out "int fifo_status;"
        }
    }

puts $out $schedParam
puts $out $attr

if {$rtf0 == 1} {
```

```
puts $out "rtf_destroy(0);"
   puts $out "fifo_status = rtf_create(0, 4000);"
   puts $out "if (fifo_status) {"
   puts $out "rtl_printf(\"RTLinux measurement test fail.
fifo_status=%d\\n\",fifo_status);"
   puts $out "return -1;"
   puts $out "}"
   puts $out "fifo0 = open(\"/dev/rtf0\", O_NONBLOCK);"
   puts $out "if (fifo0 < 0) {"
   puts $out "rtl_printf(\"/dev/rtf0 open returned %d\\n\", fifo0);"
   puts $out "return (void *) -1;"
   puts $out "}"
}

if {$rtf1 == 1} {
   puts $out "rtf_destroy(1);"
   puts $out "fifo_status = rtf_create(1, 4000);"
   puts $out "if (fifo_status) {"
   puts $out "rtl_printf(\"RTLinux measurement test fail.
fifo_status=%d\\n\",fifo_status);"
   puts $out "return -1;"
   puts $out "}"
   puts $out "fifo1 = open(\"/dev/rtf1\", O_NONBLOCK);"
   puts $out "if (fifo1 < 0) {"
   puts $out "rtl_printf(\"/dev/rtf1 open returned %d\\n\", fifo1);"
   puts $out "return (void *) -1;"
   puts $out "}"
}

for {set i 1} {$i <= $Ntasc} {incr i 1} {
                        set tasca $ArrayTasc($i)
                        set name [lindex $tasca 4]
                        set prioritat [lindex $tasca 0]
                        puts $out $attrInit
                        puts $out $prio$prioritat$pointandcoma
                        puts $out $setfp
                        puts $out $sched
                  puts $out
"$pthreadCreate1$thread$name$pthreadCreate2$routine$name$pthreadCreate3$p
ointandcoma"
            }
puts $out "$ret$pointandcoma"
puts $out "$closee"

#Generate Cleanup
puts $out "$clean"

if {$rtf0 == 1} {
    puts $out "close(fifo0);"
    puts $out "rtf_destroy(0);"
}

if {$rtf1 == 1} {
    puts $out "close(fifo1);"
    puts $out "rtf_destroy(1);"
}
```

```
for {set i 1} {$i <= $Ntasc} {incr i 1} {
                        set tasca $ArrayTasc($i)
                        set name [lindex $tasca 4]
                    puts $out "$delete1$thread1$name$delete2$pointandcoma"
             }
puts $out "$closee"

close $out

#exec rtlinux start
exec make test
```

Codi per cridar l'aplicació de simulació de les tasques

```
exec wish DiagramaTemps.tcl
```

(ii) Fitxer de Makefile cridat en l'aplicació anterior

```
all: code.o

include /usr/src/rtlinux/rtlinux-3.1/rtl.mk
clean:
      rm -f *.o

test: all
      (cd /usr/src/rtlinux/; scripts/rmrtl)
      (cd /usr/src/rtlinux/; scripts/insrtl)
      insmod code.o
      sleep 1
      rmmod.old code
      ./monitor&
      sleep 2
      sh killmonitor.sh
      @echo "Now start the real-time tasks  module"
      @echo "Type <return> to continue"
include $(RTL_DIR)/Rules.make
```

(iii) Codi font del script "killmonitor.sh"

```
#!/bin/bash

kill  `ps uxc | grep -i "monitor" | awk '{print $2}'`
```

(iv) Codi font del Simulador de Tasques del Planificador en Temps Real

```
proc ScrollTexto { parent totalTime AUX args } {
# Crea la frame on colocar el widget texto.
frame $parent -borderwidth 10
pack  $parent -side top  -expand yes -fill y
# Crea dos barras de scroll, una vertical y otra horizontal.
scrollbar $parent.sy -command "$parent.texto yview"
scrollbar $parent.sx -orient horizontal -command "$parent.texto xview"
# Crea un widget texto amb les barres de scroll associades
set i [expr $totalTime/$AUX]
eval {canvas $parent.texto  \
-bd 2 \
-scrollregion "-100 0 $i 0" \
-yscrollcommand "$parent.sy set" -xscrollcommand "$parent.sx set" -width
20 -height 10 } \
$args
# Coloca tot a la pantalla
grid $parent.texto  -row 0 -column 0 -rowspan 1 -columnspan 1 -sticky
news
grid $parent.sy -row 0 -column 1 -rowspan 1 -columnspan 1 -sticky ns
grid $parent.sx -row 1 -column 0 -rowspan 1 -columnspan 1 -sticky ew
grid rowconfig $parent 0 -weight 1 -minsize 0
grid columnconfig $parent 0 -weight 1 -minsize 0
# Retorna el widget texto
return $parent.texto
}
wm title . "Simulació de les tasques creades"

#Posem els identificadors de temps
     set fp [open "monitor2.txt" r]
     set data [read $fp]
     set lista ()
     close $fp
     # Fitxer a processar
     set data [split $data "\n"]
     set totalTime 0
     set aux 0
     set pt 0
     set AUX 4000
     set firstLine 0
     set min 999999999
     set max 0
     #Per cada linea
     foreach line $data {
                  set data1 [split $line " - "]
                  set var [lsearch $lista [lindex $data1 0]]
                  if { $var == -1 } {
                        set lista [linsert $lista [llength $lista] [lindex
$data1 0]]
                        set b [lindex $data1 3]
                        set totalTime [expr $b + $totalTime]
                  }
```

```
                  set b [lindex $data1 3]
                  set totalTime [expr $b + $totalTime]
                  puts $totalTime
                  if { $min > $b } {
                        if { $b > 0 } {
                              set min $b
                        }
                  }
                  if { $max < $b } {
                        set max $b
                  }
      }

      set AUX [expr $max/$min]
      set min [expr $AUX/2]
      # Es crida el procediment per crear la GUI
      ScrollTexto .p $totalTime $AUX -width 900 -height 600
      .p.texto xview moveto 0.0
      button .in -text ZomIn -width 30 -command {set AUX [expr $AUX +
$min]; Zoom $AUX $data $line $lista $totalTime $pt $pos $aux}
      button .out -text ZomOut -width 30 -command {set AUX [expr $AUX -
$min]; Zoom $AUX $data $line $lista $totalTime $pt $pos $aux}
      pack  .in .out -side top
      set firstLine 0
      set totalTime 0
      set pos 520
      proc Zoom { AUX data line lista totalTime pt pos aux} {
      set tasca1 0
      set tasca2 0
      .p.texto delete all
      foreach line $data {
                  set data1 [split $line " - "]
                  set var [lsearch $lista [lindex $data1 0]]
                  set b [lindex $data1 3]
                  set totalTime [expr $b + $totalTime]
                  if { [lindex $lista 1] == [lindex $data1 0] } {
                  if { [lindex $lista 1] > 0 } {
                  .p.texto create rectangle [expr $pt/($AUX)] 300 [expr
$totalTime/($AUX)] 310 -fill blue -tag rectangulo -width 0
                  .p.texto create text [expr $pt/($AUX)] 315 -text [expr
$pt]
                  .p.texto create line [expr $pt/($AUX)] 495 [expr
$pt/($AUX)] 505
                  .p.texto create text -50 305 -text "Tasca 1"
                  set tasca1 [expr $tasca1 + 1]
   }
                  set pt $totalTimetotalTime
                  } elseif { [lindex $lista 2] == [lindex $data1 0] } {
                  if { [lindex $lista 2] > 0 } {
                  .p.texto create rectangle [expr $pt/($AUX)] 350 [expr
$totalTime/($AUX)] 360 -fill red -tag rectangulo -width 0
```

```
.p.texto create text -50 355 -text "Tasca 2"
                  .p.texto create text [expr $pt/($AUX)] 365 -text [expr
$pt]
                  .p.texto create line [expr $pt/($AUX)] 495 [expr
$pt/($AUX)] 505
                  set tasca2 [expr $tasca2 + 1]
                  }
                  set pt $totalTime
                  } elseif { [lindex $lista 3] == [lindex $data1 0] } {
                  if { [lindex $lista 3] > 0 } {
                  .p.texto create rectangle [expr $pt/($AUX)] 250 [expr
$totalTime/($AUX)] 260 -fill brown -tag rectangulo -width 0
                  .p.texto create text -50 255 -text "Tasca 3"
                  .p.texto create text [expr $pt/($AUX)] 265 -text [expr
$pt]
                  .p.texto create line [expr $pt/($AUX)] 495 [expr
$pt/($AUX)] 505
                  }
                  set pt $totalTime
                  } elseif { [lindex $lista 4] == [lindex $data1 0] } {
                  if { [lindex $lista 4] > 0 } {
                  .p.texto create rectangle [expr $pt/($AUX)] 400 [expr
$totalTime/($AUX)] 410 -fill orange -tag rectangulo -width 0
                  .p.texto create text -50 405 -text "Tasca 4"
                  .p.texto create text [expr $pt/($AUX)] 415 -text [expr
$pt]
                  .p.texto create line [expr $pt/($AUX)] 495 [expr
$pt/($AUX)] 505
                  }
                  set pt $totalTime
                  } elseif { [lindex $lista 5] == [lindex $data1 0] } {
                  if { [lindex $lista 5] > 0 } {
                  .p.texto create rectangle [expr $pt/($AUX)] 450 [expr
$totalTime/($AUX)] 460 -fill black -tag rectangulo -width 0
                  .p.texto create text -50 455 -text "Tasca 5"
                  .p.texto create text [expr $pt/($AUX)] 465 -text [expr
$pt]
                  .p.texto create line [expr $pt/($AUX)] 495 [expr
$pt/($AUX)] 505
                  }
                  set pt $totalTime

                  } else {
                  set pt $totalTime
                  }
                  if { $pos == 510 } {
                        set pos 520
                        } else {
                        set pos 510
                  }

      }
```

```
.p.texto create line 0 100 0 500 -tag liney -width 2
      .p.texto create line 0 500 [expr $totalTime/($AUX)] 500 -tag linex -
width 2

      .p.texto create text -10 90 -text "Tasques"
      .p.texto create text 10 550 -text "temps (ns)"
      set n 500
      for {set i 0} {$i < $totalTime} {incr i $AUX} {
                  #.p.texto create line [expr $i/($AUX)] 495 [expr
$i/($AUX)] 505
                  if { $aux == [expr $AUX/2] } {
                        #.p.texto create text [expr $i/($AUX)] 510 -text
[expr $i]
                        .p.texto create line [expr $i/($AUX)] 495 [expr
$i/($AUX)] 505
                        set aux 0
                  } else {
                        incr aux 1
                  }

      }
      set t  [expr $totalTime/$AUX]
      .p.texto configure -scrollregion "-100 0 $t 0"
      puts "#######"
      puts $tasca1
      puts $tasca2
      }
      Zoom $AUX $data $line $lista $totalTime $pt $pos $aux
```

(v) Codi font gràfiques de sortida (Resultats exemples de control Temperatura)

```
wm title . "Resultat Control de Temps"

     set fp [open "./controlP/temp.txt" r]
     set data [read $fp]
     set lista ()
     close $fp
     set data [split $data "\n"]
     set totalTime [llength $data]
     set AUX [expr $totalTime/4]
     set min [expr $AUX/10]
     # Funció ja creada en el codi anterior
      ScrollTexto .p $totalTime $AUX -width 900 -height 600
      .p.texto xview moveto 0.0
      button .in -text ZomIn -width 30 -command {set AUX [expr $AUX +
$min]; Zoom $data $totalTime $AUX}
      button .out -text ZomOut -width 30 -command {set AUX [expr $AUX -
$min]; Zoom $data $totalTime $AUX}
      pack  .in .out -side top
      #set AUX 100000
      set firstLine 0
      set pos 520
```

```
proc Zoom { data totalTime AUX } {
      set line ()
      .p.texto delete all
      set i 0
      set line
      foreach line $data {
            if {$i < $totalTime-1} {
            set point0 [lindex $data $i ]
            .p.texto create line [expr $i*$AUX]  [expr (500 -
($point0*10))] [expr $i*$AUX] 500 -tag liney -width 4
            .p.texto create line [expr $i*$AUX] 500 [expr $i*$AUX] 100
            set n 40
            for {set t 100} {$t < 500} {incr t 10} {
                  .p.texto create line 0 $t [expr $totalTime*$AUX] $t
                  .p.texto create text -50 $t -text $n
                  set n [expr $n - 1]
            }
            for {set t 0} {$t < $totalTime} {incr t 1} {
                  .p.texto create text [expr $t*$AUX] 510 -text $t
            }
            puts [expr $i*$AUX]
            puts [expr (500 - $point0)*$AUX ]
            set i [expr $i + 1]
            }
      }

      .p.texto create line 0 100 0 500 -tag liney -width 0
      .p.texto create line 0 500 [expr $totalTime*$AUX] 500 -tag linex -
width 0
      .p.texto create line 0 [expr 500 - (18.5*10)] [expr $totalTime*$AUX]
[expr 500 - (18.5*10)] -width 2 -fill "red"
      .p.texto create text -10 80 -text "Temperatura (°C)"
      .p.texto create text 10 550 -text "Temps => T*n on T=0.5s"
      set n 500
      .p.texto configure -scrollregion "-100 0 $totalTime 0"
      }

      Zoom $data $totalTime $AUX
```

(vi) Codi font gràfiques de sortida (Resultats exemples de control Nivell, funció Zoom)

```
proc Zoom { data totalTime AUX } {
      set line ()
      .p.texto delete all
      set i 0
      set line
      foreach line $data {

                  if {$i < $totalTime-1} {
                  set point0 [lindex $data $i ]
                  .p.texto create line [expr $i*$AUX]  [expr (500 -
($point0*10))] [expr $i*$AUX] 500 -tag liney -width 4
                  .p.texto create line [expr $i*$AUX] 500 [expr $i*$AUX]
100
                  set n 40
```

```
                for {set t 100} {$t < 500} {incr t 10} {
                .p.texto create line 0 $t [expr $totalTime*$AUX] $t
                .p.texto create text -50 $t -text $n
                set n [expr $n - 1]
                }
                for {set t 0} {$t < $totalTime} {incr t 1} {
                      .p.texto create text [expr $t*$AUX] 510 -text $t
                }
                puts [expr $i*$AUX]
                puts [expr (500 - $point0)*$AUX ]
                set i [expr $i + 1]
                }
      }
      .p.texto create line 0 100 0 500 -tag liney -width 0
      .p.texto create line 0 500 [expr $totalTime*$AUX] 500 -tag linex -
width 0
      .p.texto create line 0 [expr 500 - (25*10)] [expr $totalTime*$AUX]
[expr 500 - (25*10)] -width 2 -fill "red"
      .p.texto create text -10 80 -text "Nivell (mm)"
      .p.texto create text 10 550 -text "Temps => T*n on T=0.5s"
      set n 500
      .p.texto configure -scrollregion "-100 0 $totalTime 0"
      }
      Zoom $data $totalTime $AUX
```

(vii) Codi font del planificador de tasques en Temps Real (rtl_sched.c)

En aquest codi tant sols es mostrà la funció "init_module" i "scheduler" que són les que és modifiquen.

Codi init_module

```
int init_module(void)
{
      rtl_irqstate_t interrupt_state;
      int i;
      int fifo_status;
      int ret;
      int my_cpu_id;
      char init;
      schedule_t *s;
      unsigned int cpu_id = rtl_getcpuid();
      now1 = gethrtime();
      rtl_spin_lock_init (&rtl_tqueue_lock);
      zombie_threads = 0;
      time1 = 0;
      time2 = 0;
      first = 0;
      contadorTime=0;
      ret  = rtl_get_soft_irq (sched_irq_handler, "RTLinux Scheduler");
      proc = 0;
```

```
if (ret  > 0) {
            rtl_sched_irq = ret ;
      } else {
            rtl_printf ("Can't get an irq for RTLinux scheduler");
            return -EINVAL;
      }
      proc = 0;
      rtl_no_interrupts(interrupt_state);
      my_cpu_id = cpu_id;

      rtf_destroy(0);
      fifo_status = rtf_create(0,4000);
      if (fifo_status)
            {
                  rtl_printf("RTLinux measurement test fail.
fifo_status=%d\n",fifo_status);
                  return -1;
            }
      rtl_printf("RTLinux measurement module on CPU %d\n",rtl_getcpuid());
fd_fifo = open("/dev/rtf0", O_NONBLOCK);
                  if (fd_fifo < 0) {
                        rtl_printf("/dev/rtf0 open returned %d\n",
fd_fifo);
                        return (void *) -1;
                        }
      init = '-';
      write(fd_fifo,&init,sizeof(char));

      for (i = 0; i < rtl_num_cpus(); i++) {
            cpu_id = cpu_logical_map (i);
            s = &rtl_sched [cpu_id];
            s -> rtl_current = &s->rtl_linux_task;
            s -> rtl_tasks = &s->rtl_linux_task;
            s -> rtl_new_tasks = 0;

            rtl_spin_lock_init (&s->rtl_tasks_lock);

            s -> rtl_linux_task . magic = RTL_THREAD_MAGIC;
            rtl_sigemptyset(&s -> rtl_linux_task . pending);
            rtl_sigaddset(&s -> rtl_linux_task . pending,
RTL_SIGNAL_READY);
            s -> rtl_linux_task . blocked = 0;
            s -> rtl_linux_task . threadflags = 0;
            s -> rtl_linux_task . sched_param . sched_priority = -1;
            s -> rtl_linux_task . next = 0;
            s -> rtl_linux_task . uses_fp = 1;
            s -> rtl_linux_task . fpu_initialized = 1;
            s -> rtl_linux_task . creator = 0;
            s -> rtl_linux_task . abort = 0;


            s -> rtl_task_fpu_owner = &s->rtl_linux_task;
            s -> sched_flags = 0;
            rtl_posix_init (&s->rtl_linux_task);
            s-> clock =  rtl_getbestclock (cpu_id);
```

```
            } else {
                  rtl_printf("Can't get a clock for processor
%d\n",cpu_id);
                  rtl_restore_interrupts (interrupt_state);
                  return  -EINVAL;
            }
      }
      cpu_id = my_cpu_id;

#ifdef CONFIG_SMP

      for (i = 0; i < rtl_num_cpus(); i++) {
            int cpu;
            int ret;
            cpu = cpu_logical_map (i);
            s = &rtl_sched [cpu];
            ret = rtl_request_ipi(resched_irq, cpu);
      }

#endif
      rtl_restore_interrupts (interrupt_state);
      save_errno_location = __errno_location;
      __errno_location = thread_errno_location;

/*    rtl_setdebug (RTLDBG_ALL); */
      return 0;
}
```

Codi rtl_schedule

```
static hrtime_t ini_time=0LL;
int rtl_schedule (void)
{
      schedule_t *sched;
      struct rtl_thread_struct *t;
      struct rtl_thread_struct *new_task;
      struct rtl_thread_struct *preemptor = 0;
      unsigned long oldthread;
      unsigned long newthread;
      int prio;
      char *oldTask;
      char *newTask;
      char *time;
      unsigned long interrupt_state;
      unsigned long _pthread;
      int cpu_id = rtl_getcpuid();
      hrtime_t now;
      rtl_sigset_t mask;

      rtl_no_interrupts(interrupt_state);
      rtl_trace2 (RTL_TRACE_SCHED_IN, (long) pthread_self());
      /* new_task = &sched->rtl_linux_task;*/
      new_task = 0;
      sched = &rtl_sched[cpu_id];
      now = sched->clock->gethrtime(sched->clock);
```

```
      if ((sched->rtl_current != NULL))
            {
            if (first == 0)
            {
                  idOld = (unsigned long)sched->rtl_current;
                  taskLinux = idOld;
                  printk("Primera tasca a executar-se %ul \n\n",idOld);
                  first = 1;
                  time1 = 0;
                  now1 = gethrtime();
            }
            else
            {
                  if (idOld != (unsigned long)sched->rtl_current)
                  {
                        _pthread =(unsigned long)sched->rtl_current;
                  if ((unsigned long)sched->rtl_current == taskLinux) {
                        printk("LINUX: %ul amb prioritat %d - %ld
\n",(unsigned long)sched->rtl_current,sched->rtl_current-
>sched_param.sched_priority,time1);
                        }
                        else
                        {
                        printk("OTHER: %ul amb prioritat %d - %ld
\n",(unsigned long)sched->rtl_current,sched->rtl_current-
>sched_param.sched_priority,time1);
                        tasca = tasca + 1;
                        printk("%ld\n",tasca);
                        }
                        write (fd_fifo,&_pthread, sizeof(unsigned long));
                        write (fd_fifo,&sched->rtl_current-
>sched_param.sched_priority, sizeof(int));
                        write (fd_fifo,&time1, sizeof(long int));
                        idOld = (unsigned long)sched->rtl_current;
                        time1 = 0;
                  }

            }
      }


if (sched->clock->mode == RTL_CLOCK_MODE_ONESHOT) {
            sched->clock->value = now;
      }

      for (t = sched->rtl_tasks; t; t = t->next) {
            /* expire timers */

            if (test_bit(RTL_THREAD_TIMERARMED, &t->threadflags)) {
                  if (now >= t->resume_time) {
                        clear_bit(RTL_THREAD_TIMERARMED, &t->threadflags);
                        rtl_sigaddset (&t->pending, RTL_SIGNAL_TIMER);
                        if (t->period != 0) { /* periodic */
                              t->resume_time += t->period;
                              /* timer overrun */
#ifndef CONFIG_RTL_OLD_TIMER_BEHAVIOUR
```

```
                                while (now >= t->resume_time) {
                                      t->resume_time += t->period;
/*                                    rtl_printf("overrun"); */
                                }
#endif
                          } else {
                                t->resume_time = HRTIME_INFINITY;
                          }
                    }
              }

              /* and find highest priority runnable task */
              if ((t->pending & ~t->blocked) && (!new_task ||
                    (t->sched_param.sched_priority > new_task-
>sched_param.sched_priority))) {
                          _pthread = (unsigned long)pthread_self();
                          if (proc == 0) {
                                proc = _pthread;
                                printk ("ID Linux: %ul\n\n", proc);
                          }
                          prio = pthread_self()->sched_param.sched_priority;
                          contadorTime = contadorTime + 1;
                          time1 = time1 + ((gethrtime()-now1));
                          if(proc!=_pthread){
                                new_task = t;
                                now1 = gethrtime();
                                contadorTime = 0;
                          }
                          else {
                                new_task = t;
                                now1 = gethrtime();
                          }

              }
        }

        if (sched->clock->mode == RTL_CLOCK_MODE_ONESHOT && !test_bit
(RTL_SCHED_TIMER_OK, &sched->sched_flags)) {
              if ( (preemptor = find_preemptor(sched,new_task))) {
                    (sched->clock)->settimer(sched->clock, preemptor-
>resume_time - now);
              } else {
                    (sched->clock)->settimer(sched->clock, (HRTICKS_PER_SEC
/ HZ) / 2);
              }
              set_bit (RTL_SCHED_TIMER_OK, &sched->sched_flags);
        }

        if (new_task != sched->rtl_current) { /* switch out old, switch in
new */
              if (new_task ==  &sched->rtl_linux_task) {
                    rtl_make_rt_system_idle();

              } else {
                    rtl_make_rt_system_active();
              }
              rtl_trace2 (RTL_TRACE_SCHED_CTX_SWITCH, (long) new_task);
              rtl_switch_to(&sched->rtl_current, new_task);
```

```
            /* delay switching the FPU context until it is really needed
*/
#ifdef CONFIG_RTL_FP_SUPPORT
            if (sched->rtl_current-> uses_fp &&\
                              sched->rtl_task_fpu_owner != sched-
>rtl_current)
            {
                  if (sched->rtl_task_fpu_owner)
                  {
                        rtl_fpu_save (sched,sched->rtl_task_fpu_owner);
                  }
                  rtl_fpu_restore (sched,sched->rtl_current);
                  sched->rtl_task_fpu_owner = sched->rtl_current;
            }
#endif /* CONFIG_RTL_FP_SUPPORT */
      }
      mask = pthread_self()->pending;

      if (pthread_self()->pending & ~(1 << RTL_SIGNAL_READY))
do_signal(pthread_self());

      rtl_trace2 (RTL_TRACE_SCHED_OUT, (long) pthread_self());
      rtl_restore_interrupts(interrupt_state);
      return mask;
}
```

7.2 Codi font exemples generats

(i) Exemple1: Interrupció del teclat

```
#include <rtl_core.h>
#include <rtl.h>
#include <pthread.h>


#define KEYBOARD_INTERRUPT  1

hrtime_t start,end;
pthread_t pthread_id;


unsigned my_keyboard_interrupt_handler(unsigned int irq, struct pt_regs
*regs){
    start = gethrtime();
    pthread_wakeup_np(pthread_id); // desperta tasca la interrupció
    return 0;
}
```

```
void *my_pthread_handler(void *arg){
while (1) {

      pthread_suspend_np(pthread_self());
      end = gethrtime();
      rtl_printf("+"); // La tasca activada escriu el carácter +
                        // i torna a esperar per una interrupció
      rtl_global_pend_irq(KEYBOARD_INTERRUPT);
      rtl_printf("%ld\n",end-start);
      // Temps que tarda a executar-se rutina tractament de la interrupció
      // Latència de la interrupció.
  }

}

int init_module(void) {

      pthread_create(&pthread_id,NULL,my_pthread_handler,NULL);
      return
rtl_request_irq(KEYBOARD_INTERRUPT,my_keyboard_interrupt_handler);
}

void rtl_cleanup_module(void) {
      rtl_free_irq(KEYBOARD_INTERRUPT);
      pthread_delete_np(pthread_id);
}
```

(ii) Exemple2: Prova de só

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>
#include <rtl_fifo.h>
#include <asm/io.h>

pthread_t tasca;

/*Filtre del só*/

static int filter(int x)
{
   static int oldx;
   int ret;
   if(x & 0x80)
     {
      x = 382 - x;
     }
   ret = x > oldx;
   oldx = x;
   return ret;
}
```

```
void * sound(void *arg)
{
   char dat;
   char t;
   struct sched_param p;
   p.sched_priority=1;
   pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
   pthread_make_periodic_np (pthread_self(),
gethrtime(),1000000000/8192LL); /* freq = 8192 Hz */
   while(1)
   {
     pthread_wait_np ();
     if(rtf_get(4, &dat, 1) > 0)
     {
      dat = filter(dat);
      t = inb(0x61);
      t &= 0xfd;
      t |= (dat & 1) << 1;
      outb(t,0x61);
     }
   }
}


int init_module(void)
{
  rtf_create(4, 4000); /* crear fifo */
  /* preparar el speaker */
  outb_p(inb_p(0x61)|3, 0x61);
  outb_p(0xb0, 0x43);
  outb_p(3, 0x42);
  outb_p(00, 0x42);
  return pthread_create(&tasca, NULL,sound, 0);
}

void cleanup_module(void)
{
  pthread_delete_np(tasca);
  rtf_destroy(4);

}
```

(iii) Exemple3: Prova de la periodicitat

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>

pthread_t threadTasca1

void * start_routineTasca1(void *args)
{
      struct sched_param p;

      p.sched_priority=1;
      pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
      pthread_make_periodic_np(pthread_self(),gethrtime(),10000000);
```

```
	while(1){
		pthread_wait_np();
		i = 2+2;
	}
	return 0;
}

int init_module(void){
pthread_create(&threadTasca1,NULL,start_routineTasca1,0);
return 0;
}
void cleanup_module(void){
pthread_delete_np(start_routineTasca1);
}
```

(iv) Exemple4: Control d'un procés simple

```
#include <rtl.h>
#include <time.h>
#include <pthread.h>

//Declaracions variables compartides
int i=0; //Sortida de la simulació
int u=0; //SOrtida de l'acció de control


pthread_t threadControl;
pthread_t threadSimulacio;

void * start_routineSimulacio(void *args)
{
	struct sched_param p;
	p.sched_priority=100;
	pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
	pthread_make_periodic_np(pthread_self(),gethrtime(),5000000);
	while(1){
		pthread_wait_np();
		i = u + i;
	}
	return 0;
}

void * start_routineControl(void *args)
{
struct sched_param p;
int a = 0;

p.sched_priority=100;
pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);
pthread_make_periodic_np(pthread_self(),gethrtime(),7000000);
while(1){
	pthread_wait_np();
```

```
        if(a==10){
                a = 0;
                if(u==0)
                {
                        u = 1;
                }
                else
                {
                        u = 0;
                }
        }
        else
        {
                a = a + 1;
        }

   }
 return 0;
}

int init_module(void){
pthread_create(&threadSimulacio,NULL,start_routineSimulacio,0);
pthread_create(&threadControl,NULL,start_routineControl,0);
return 0;
}

void cleanup_module(void){
pthread_delete_np(threadSimulacio);
pthread_delete_np(threadControl);
}
```

(v) Exemple5: Control PID d'un tanc d'aigua (Temperatura i Nivell)

```
#include "rtl.h"
#include "rtl_sched.h"
#include <math.h>
#include <rtl_fifo.h>

//Declaracions variables compartides
extern double sqrt(double x);
static double Level= 0.00;
static double Temp = 15.00;
static double Qt ,Qtm = 16.00;
static double Qin, Qinput = 10.0;
static double Qs = 30.00;
static double Qtotal = 0.00;
static double Pvt = 0.0, Pvs = 0.0;

// Variables de control
static double klevel = 0.9;
static double ktemp = 0.5;
static double Nref=25.00;
static double Tref=18.50;
```

```
//Variables Controlador D
static double dtempT = 0.4;
static double dtempN = 0.1;
static double eAnteriorT = 0;
static double eAnteriorN = 0;

//Variables Controlador I
static double iAnteriorT = 0;
static double iAnteriorN = 0;
static double itempT = 0.7;
static double itempN = 1;

int fifo1;

pthread_t threadSimulacio;
pthread_t threadVisualitzacio;
pthread_t threadControlNivell;
pthread_t threadControlTemp;

void *start_routineSimulacio(void *arg)
{
  double F1,F2;
  double Te=15.00;
  double Tc=45.00;
  double cte1=0.02;
  double cte2=3.8;
  double random;
  long int_random;

  pthread_make_periodic_np(pthread_self(),gethrtime(),50000000);
  while(1){
    pthread_wait_np();
    rdtscl(int_random);
    random = ((double)(int_random & 0x1f)) / 256.0;
    Qin = Qinput + (random - 0.5);
    Qt = Qtm * Pvt;
    Qs = cte2 * sqrt(Level) * Pvs;
    F1 = (Qin * Te) + (Qt * Tc) + ((Qtotal - Qs) * Temp);
    F2 = (Qin + Qt + Qtotal - Qs);
    Temp = F1 / F2;
    Level = Level + ((Qin - Qs + Qt) * cte1);
    Qtotal = Qtotal - Qs + Qin + Qt;
    }
}



void *start_routineVisualitzacio(void *arg)
{
      pthread_make_periodic_np(pthread_self(),gethrtime(),500000000);
      while(1){
        pthread_wait_np();
      write (fifo1, &Temp, sizeof(double));
      write (fifo1, &Level, sizeof(double));
      }
}
```

```
void *start_routineControlNivell(void *arg)
{
double pvs;
double dvt;
double ivt;

pthread_make_periodic_np(pthread_self(),gethrtime(),700000000);
while(1){
      pthread_wait_np();
      pvs = (Level - Nref)*klevel;
      dvt = ( pvs - eAnteriorN)*dtempN;
      eAnteriorN = pvs;
      ivt = iAnteriorN + (itempN*pvs);
      iAnteriorN = ivt;
      pvs = pvs + ivt + dvt;
            if (pvs<0) pvs=0;
            if (pvs>1.00) pvs=1.00;
      Pvs = pvs;
  }
}


void *start_routineControlTemp(void *arg)
{
double pvt;
double dvt;
double ivt;

pthread_make_periodic_np(pthread_self(),gethrtime(),700000000);
while(1){
      pthread_wait_np();
      pvt = (Tref - Temp)*ktemp;
      dvt = ( pvt - eAnteriorT )*dtempT;
      eAnteriorT = pvt;
      ivt = iAnteriorT + (itempT*pvt);
      iAnteriorT = ivt;
      pvt =  pvt + ivt + dvt;
      if (pvt<0.0) pvt=0;
      if (pvt>1.00) pvt=1.0;
      Pvt = pvt;
   }
}

int init_module(void){
  int fifo_status;
  struct sched_param sched_param;
  pthread_attr_t attr;
  rtf_destroy(1);
  fifo_status = rtf_create(1, 4000);
  if (fifo_status) {
      rtl_printf("RTLinux measurement test fail.
fifo_status=%d\n",fifo_status);
      return -1;
  }
  fifo1 = open("/dev/rtf1", O_NONBLOCK);
  if (fifo1 < 0) {
      rtl_printf("/dev/rtf1 open returned %d\n", fifo1);
      return (void *) -1;
  }
```

```
 pthread_attr_init (&attr);
 sched_param.sched_priority =100;
 pthread_attr_setfp_np(&attr, 1);
 pthread_attr_setschedparam (&attr, &sched_param);
 pthread_create(&threadSimulacio,&attr,start_routineSimulacio,0);
 pthread_attr_init (&attr);
 sched_param.sched_priority =20;
 pthread_attr_setfp_np(&attr, 1);
 pthread_attr_setschedparam (&attr, &sched_param);
 pthread_create(&threadVisualitzacio,&attr,start_routineVisualitzacio,0);
 pthread_attr_init (&attr);
 sched_param.sched_priority =100;
 pthread_attr_setfp_np(&attr, 1);
 pthread_attr_setschedparam (&attr, &sched_param);
 pthread_create(&threadControlNivell,&attr,start_routineControlNivell,0);
 pthread_attr_init (&attr);
 sched_param.sched_priority =100;
 pthread_attr_setfp_np(&attr, 1);
 pthread_attr_setschedparam (&attr, &sched_param);
 pthread_create(&threadControlTemp,&attr,start_routineControlTemp,0);
 return 0;
}
void cleanup_module(void){
 close(fifo1);
 rtf_destroy(1);
 pthread_delete_np(threadSimulacio);
 pthread_delete_np(threadVisualitzacio);
 pthread_delete_np(threadControlNivell);
 pthread_delete_np(threadControlTemp);
}
```

Estructura del CD adjuntat

Amb la memòria s'adjunta un CD amb tota la informació inserida. El CD conté aquest propi document, conté el codi font de les aplicacions creades, els exemples generats, els kernels utilitzats per instal·lar RTLinux, la imatge creada de RTLinux, el paquet de Tcl/Tk i vTcl.
La forma en que es distribueix el CD es la següent:

D:/Memòria del Projecte/: En aquest directori es troba en format electrònic la memòria del present projecte.

D:/Codi font Exemples/: En aquest directori es troben tots els exemples generats de tasques en temps real, ja sigui amb l'aplicació de generació de tasques o de forma manual.

D:/Codi font Aplicacions/: Aquí es troba el codi font de les aplicacions, Generador de Tasques, simulador planificador en temps real, aplicacions de graficació dels resultats.

D:/Instal·lació RTLinux/: En aquest directori hi ha els dos kernels utilitzats per la instal·lació de RTLinux (kernel RTLinux i linux) i també la imatge generada que és la que arranca al iniciar RTLinux i tot el que fa referència a RTLinux.

D:/Tcl/Tk/: Aquí es troba el paquet de Tcl/Tk i la eina de desenvolupament visual tcl.

D:/RTLinux exemples/: Conjunt d'exemples de programes utilitzant la API RTLinux (exemples proporcionats per RTLinux).

D:/Documents/: Aquí es troben una sèrie de documents interessants sobre tots els temes tractats en aquest projecte: temps real, RTLinux, Tcl/Tk, control computacional, etc.

Referències Bibliogràfiques

1.- Sistemes de temps real

[1.a] Comp.realtime: Frequently Asked Questions (FAQs) (version 3.6)
http://www.faqs.org/faqs/realtime-computing/faq/

[1.b] Bina Ramamurthy
http://www.cs.buffalo.edu/faculty/bina/cse421/spring00/lec10/index.htm

[1.c] Sistemas de Tiempo Real y Lenguajes de Programación (3ª Edición).
Autor: Alan BURNS y Andy WELLINGS, Editorial:ADDISON-WESLEY Iberoamericana España

[1.d] G. Bernat, A. Llamosí, R. Puigjaner. "Diseño de Sistemas de Tiempo Real"

[1.e] N. Audsley and A. Burns. "Real-Time System Scheduling"

2.- Sistemes Operatius

[2.a] "Sistemas Operativos", William Stallings, 4ª edición. Prentice-Hall, 2001

[2.b] Andrew S. Tanenbaum y Albert S. Woodhull. "Sistemas Operativos: Diseño e Implementación (Segunda Edición)". Prentice-Hall, 1998.

3.- Sistemes Operatius de Temps Real

[3.a] Facultad de Ciencias Exactas y Naturales y Agrimensura.
http://exa.unne.edu.ar/

[3.b] Depto. Lenguajes y Sistemas Informàicos. (Universidad de Granada)
http://lsi.ugr.es/~jagomez/disisop_archivos/

[3.c] Ciclope Group "Analisis de Sistemas Operativos de Tiempo Real Libres"
http://www.ciclope.info/

[3.d] Artícle "Comparing real-time Linux alternatives" , Kevin Dankwardt, of K Computing
http://www.linuxdevices.com/

[3.e] Artícle "WP1 - RTOS State of the Art Analysis", OCERA
http://www.mnis.fr/opensource/ocera/rtos/

[3.f] "Adaptive Domain Environment for Operating Systems"
http://www.opersys.com/adeos/

[3.g] "School of Computer science / Carnegie Mellon University"
http://www.cs.cmu.edu/~aml/chimera/chimera.html

[3.h] "RTAI - the RealTime Application Interface for Linux from DIAPM"
https://www.rtai.org/

[3.i] "RTLinux Portal at Valencia"
http://rtportal.upv.es/

[3.j] "QNX Software Systems"
http://www.qnx.com/
[3.k] "Wind River" VxWorks OS
http://www.windriver.com/vxworks/

4.- RTLinux

[4.a] "RTLinux Portal at Valencia"
http://rtportal.upv.es/
[4.b] Web de OS3, empresa de implementació de software en codi lliure
http://www.os3sl.com/
[4.c] "OCERA Project, Open Components for Embedded Real-time Applications"
http://www.ocera.org/
[4.d] Plana personal de Sergio Pérez (Univerisada Politecnica de Valenia)
http://www.rtlinux-gpl.org/~serpeal/
[4.e] Plana personal de Ismael Ripoll (Univerisada Politecnica de Valenia)
http://www.gii.upv.es/personal/iripoll/
[4.f] Manual instal·lació RTLinux
http://www.ciclope.info/doc/rtos/cache/doc/man_instal_RTLinux.htm
[4.g] Artícle "The RTLinux Manifesto", Victor Yodaiken
http://www.it.iitb.ac.in/~venkat/rtlmanifesto.html
[4.h] Artícle "RTLinux Whitepaper", Victor Yodaiken
http://www.vmlinux.org/rtl/docs/RTLinux-Approach.pdf
[4.i] Conjunt d'artícles sobre RTLinux, web de "vmlinux"
http://www.vmlinux.org/
[4.j] "Getting Started with RTLinux" FSM Labs, Inc
http://courses.engr.uky.edu/fall05/ECE/ee599-004/rtldoc-3.2-pre1/doc/html/GettingStarted/
[4.k] Artícle " Desarrollo cruzado de sistemas empotradors, basados en RTLinux",
Pau Mendoza, Ismael Ripoll, Joan Vila, Alfons Crespo. (Univerisada Politecnica de Valenia)
http://trecom.upv.es/articles/Sist_empotrados_en_RTLinux_Jornadas_RT-2001.pdf
[4.l] RTLinux Lightweight TCP/IP Stack (RTL-lwIP)
http://rtl-lwip.sourceforge.net/

5.- Tcl/TK

[5.a] "Tutorial de Tcl/Tk" Universidad de Oviedo
http://www6.uniovi.es/tcl/tutorial/
[5.b] Web de "Tcl Developer Xchange"
http://www.tcl.tk/

[5.c] Wiki de Tcl/Tk “The Tcler’s Wiki”
http://wiki.tcl.tk/
[5.d] Exemples d’aplicacions Tcl/Tk
http://tcltk.free.fr/
[5.e] Visual Tcl
http://vtcl.sourceforge.net/

6.- Control computacional

[6.a] Control PID (Tutorial Matlab)
http://ib.cnea.gov.ar/~control2/Links/Tutorial_Matlab_esp/PID.html
[6.b] Apunts Sistemes de control (Sistemas Realimentados)
http://avellano.fis.usal.es/~bcurto/docencia/controlIQ/pdf/RepresM-IV.pdf
[6.c] Apunts professor.

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

L'alumne, Marc Franco i Farré

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat per ell.

I per tal que consti firma la present.

Signat: .....

Bellaterra, 11 de Juny de 2007

- Resum -

El projecte "Anàlisi del sistema operatiu RTLinux e implementació d'un entorn de desenvolupament de tasques en temps real" analitza la possibilitat de crear un entorn de desenvolupament de tasques en temps real per poder crear sistemes de control complex, tot això es fa utilitzant codi lliure. Inicialment es fa un aprenentatge sobre el concepte de temps real, després s'elegeix el sistema operatiu en temps real RTLinux per a crear l'entorn de desenvolupament utilitzant el llenguatge de programació Tcl/Tk. Es creen un conjunt d'aplicacions (pel control computacional) per estudiar la viabilitat de la construcció de l'entorn desitjat per facilitar la tasca de l'usuari final. Aquest projecte obre multitud de possibles camins a continuar: comunicació remota, implementació de planificadors, estudi de controladors, etc.

- Resumen -

El proyecto "Análisis del sistema operativo RTLinux e implementación de un entorno de desarrollo de tareas en tiempo real" analiza la posibilidad de crear un entorno de desarrollo de tareas en tiempo real para poder crear sistemas de control complejos, todo esto se hace usando código libre. Inicialmente se hace un aprendizaje del concepto de tiempo real, después se elige el sistema operativo RTLinux para crear el entorno de desarrollo usando el lenguaje de programación Tcl/Tk. Se crean un conjunto de aplicaciones (para el control computacional) para estudiar la viabilidad de la construcción del entorno deseado para facilitar la tarea del usuario final. Este proyecto abre multitud de posibles caminos a seguir: comunicación remota, implementación de planificadores, estudio de controladores, etc.

- Summary -

This project, "Analysis of the RTLiunx operative system and implementation of a real time task-development environment", analyses the posibility to create a real time task development environment to create complex real time control systems using a free code. Initially, the concept of "real time" is learned; afterwards, RTLinux operative system is chosen to create the real time task development environment with Tcl/Tk language. A set of aplications are created(for control computation) to study the feasibility of the construction of the desired environment to make tasks easier for the final user. This project opens a great range of paths to carry on the research: remote communication, schedulers development, contorller study, etc.