Planificador de processos en temps real per un microcontrolador 8051

Aleix Badia i Bosch
Tutor: Sebastià Vila i Marta

curs 2002-2003

Índex

Capítol 1

Introducció

L'objectiu d'aquest projecte és realitzar un estudi, teòric i pràctic, d'un planificador de processos en temps real per un sistema encastat basat en un microcontrolador 8051.

Al llarg de la carrera d'Enginyeria Tècnica en Electrònica Industrial s'estudien diferents models de control en entorns industrial. La implementació que es proposa en aquest projecte pretén introduir un nou model en funció de les línies de treball que es presenten a continuació:

- Sistemes operatius
- Temps real

Pel desenvolupament en els camps citats anteriorment, cal profunditzar els coneixements adquirits sobre els següents conceptes:

- Arquitectura sistemes encastats
- Arquitectura 8051
- Gestió de hardware
- Gestió de processos

La implementació de les línies que s'introdueixen ajuden a millorar el control industrial en el següent apartats:

- Estabilitat

- Determinació
- Multiprocés
- Transportabilitat
- Agilitat de desenvolupament

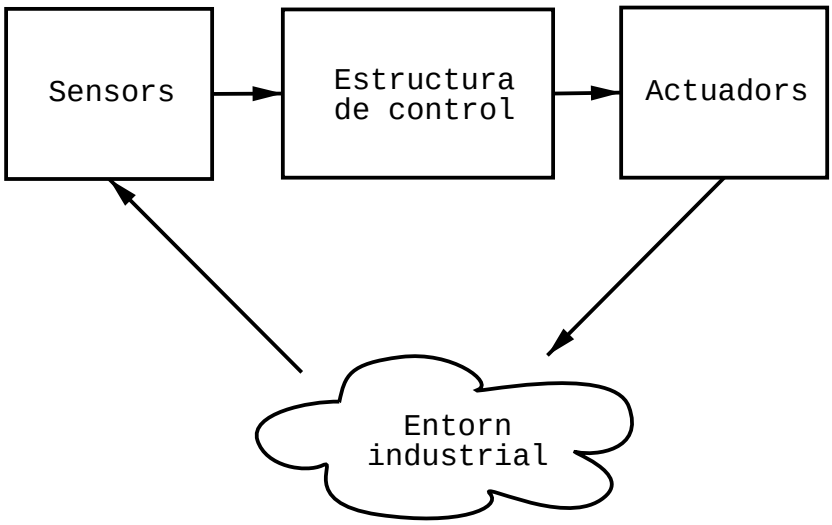
Capítol 2

Definició del problema: Context i funcionalitat

2.1 Context del projecte

2.1.1 Sistemes encastats

En els entorns industrials actuals, els sistemes de control formen part essencial de l'estructura interna de la maquinària que s'utilitza en diferents processos productius. L'arquitectura bàsica d'aquests sistemes de control, interacciona amb l'entorn seguint l'esquema de la figura següent



El sistema de control es pot dividir en diferents subapartats segons la seva funció

- **Control de regulació o d'anell tancat**

  Gestiona l'acció dels actuadors en funció de l'estat actual de l'entorn per tal que aquest assoleixi l'estat objectiu

- **Detecció d'errors**

  Gestiona la detecció d'errors, les casuístiques que els provoquen i els procediments a seguir per recuperar l'estat de funcionament normal

- **Control de supervisió**

  Controla l'execució dels diferents sistemes simples que poden formar un sistema

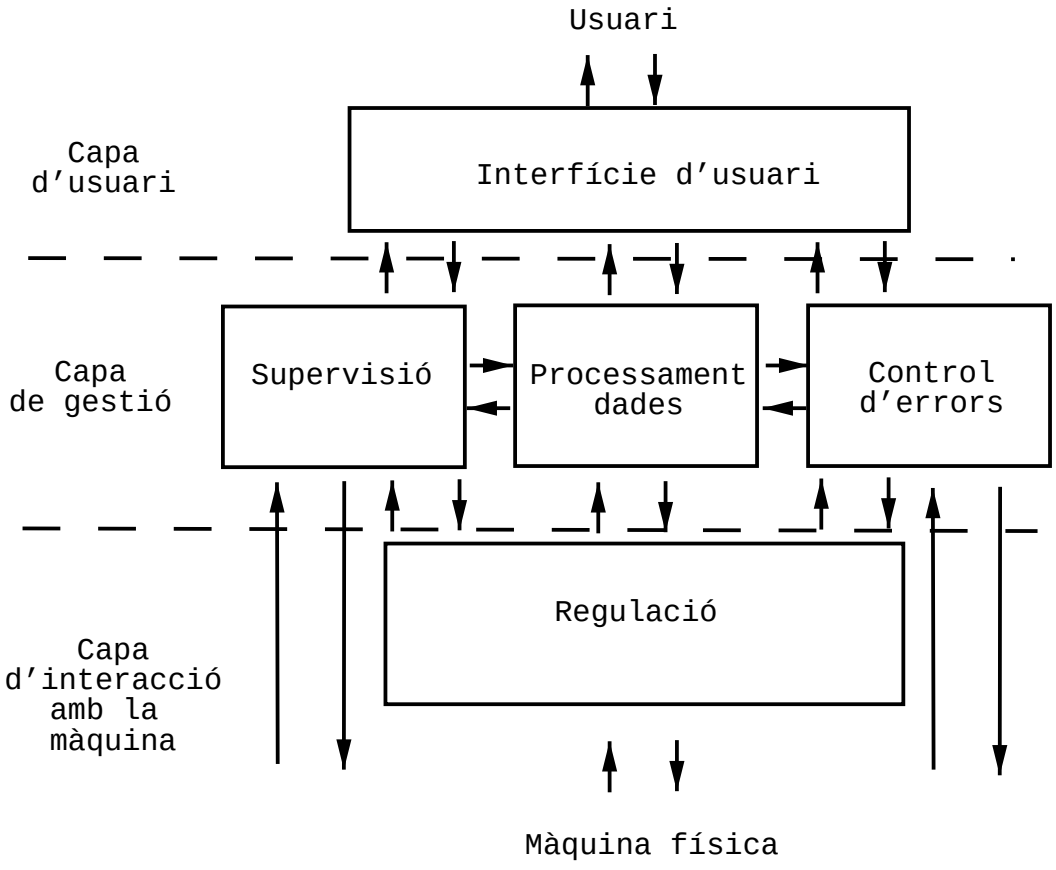
- **Processament de dades**

  Gestiona i coordina les dades que recopila el sistema

- **Interfície d'usuari**

  Controla la interacció entre l'usuari i el sistema de control

La relació entre els diferents subsistemes funcionals anteriors, es pot veure en la figura següent



Les relacions d'interacció de la figura anterior mostrem com la divisió en tres capes del funcionament del sistema de control

- **Capa d'interacció amb la màquina**

  Gestiona la interacció entre sensors i actuadors. Alguns dels components del sistema de control, interaccionen directament amb els elements de la màquina a través del regulador o a través del gestor d'errors

- **Capa de gestió**

  Planifica i coordina els diferents elements del sistema de control. Intercanvia informació entre els elements, gestiona els errors i fa de pont entre la màquina física i l'usuari

- **Capa d'usuari**

  Gestiona la interacció entre l'usuari i la màquina donant l'accés a l'estat i modificació dels paràmetres de funcionament de la segona

La utilització del sistemes encastats en sistemes de control, és fruit d'una evolució tecnològica que es descriu cronològicament a continuació

- **Treball manual**

  La producció es realitza de forma manual, amb l'única ajuda d'eines que col·laboren amb la tasca del treballador qualificat. La qualitat i el control d'errors estan exclusivament en funció de les habilitats personals del treballador; la producció és peça a peça

- **Maquinària universal**

  La producció es realitza utilitzant màquines amb funcionalitats específiques i controlades per un treballador. La qualitat i el control d'errors són compartits entre màquina i treballadors; la producció comença a ser en sèrie

- **Maquinària específica**

  La producció es realitza utilitzant màquines autònomes que realitzen tasques cícliques, algunes vegades màquines universals automatitzades. El producte que surt de cada màquina es traspassa a la següent utilitzant element semiautomàtics. El control del funcionament de la màquina es realitza de manera mecànica,elèctrica o hidràulica i amb una control d'enllaç tancat limitat. La qualitat i control d'errors és una tasca de la màquina i compartida amb un treballador que en controla vàries

- **Enllaços de maquinària específica**

  La producció es realitza utilitzant màquines autònomes que realitzen tasques cícliques i que interconnecten els productes realitzats mitjançant elements automàtics. Aquests elements automàtics passen a formar part del propi sistema esdevenint una producció en cadena. El control de la producció es realitza a través de programari i s'intercanvia la informació entre

les diferents màquines del procés de producció. La qualitat i la detecció d'errors és tasca de la màquina, compartint la resolució d'errors complexes amb el treballador

- **Enllaços de maquinària específica amb control intel·ligent**

  És una extensió del model d'enllaços de maquinària específica on s'introdueixen elements de control que gestionen la totalitat de la producció. Aquests elements mantenen la qualitat de la producció i monitoritzen el sistema gestionant el control d'errors. La volatilitat del sistema permet una funcionament molt dinàmic a nivell de paràmetres de funcionament.

Arran de la complexitat de la maquinària que es desenvolupa a nivell industrial, sorgeix la necessitat de disposar d'elements de computació dinàmics. Aquests, estan integrats a la maquinària, en alguns casos, en algun dels subapartats destinats a realitzar tasques de control sobre elements específics. Aquests elements de computació integrats són els anomenats sistemes encastats, que tenen les següents característiques:

- **Element del sistema**

  És un subsistema del sistema de control realitzant tasques específiques

- **Hardware específic**

  Al realitzar tasques de control sobre elements específics de la màquina, precisa d'interfícies específiques d'interacció amb aquesta última

- **Requisits temporals**

  Al realitzar tasques molt específiques el temps de resposta és crític. Compleix els requisits que precisi la tasca en la velocitat d'interacció en l'intercanvi d'informació amb la màquina

- **Limitacions de hardware**

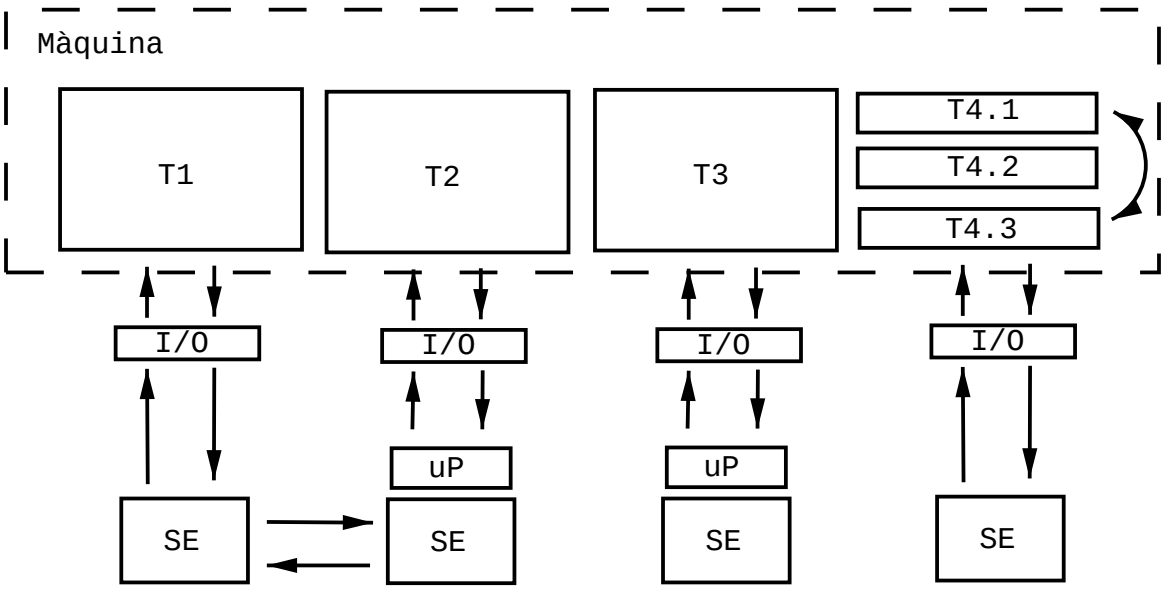
  Al realitzar tasques molt específiques i estar integrat en la mateixa màquina, compleix uns requisits molt estrictes. Qüestions com el nivell de potència, soroll, espai són bàsics en la seva integració al sistema

- **Processadors específics**

  Al realitzat tasques molt específiques pot utilitzar processadors addicionals per realitzar tasques específiques com la gestió de senyals digitals

- **Funcionament dinàmic**

  Al realitzar tasques sobre un sistema dinàmic s'adapta a l'estat d'aquest a l'hora de definir el seu funcionament.



2.1.2 Característiques generals dels sistemes encastats

Els models de control dels sistemes encastats que s'han presentat durant la carrera, tenen les següents característiques:

Aplicacions específics

La utilització del sistemes encastats s'ha realitzat per a tasques molt específiques. El tipus d'entorns que es presentaven, estaven molt ben especificats i no tenien característiques dinàmiques, entenent com a característiques dinàmiques els següents punts

- Modificacions en l'element a produir
- Modificacions en l'entorn de producció
- Modificacions en el maquinari utilitzat
- Modificacions en les normatives de seguretat
- Errors en la producció

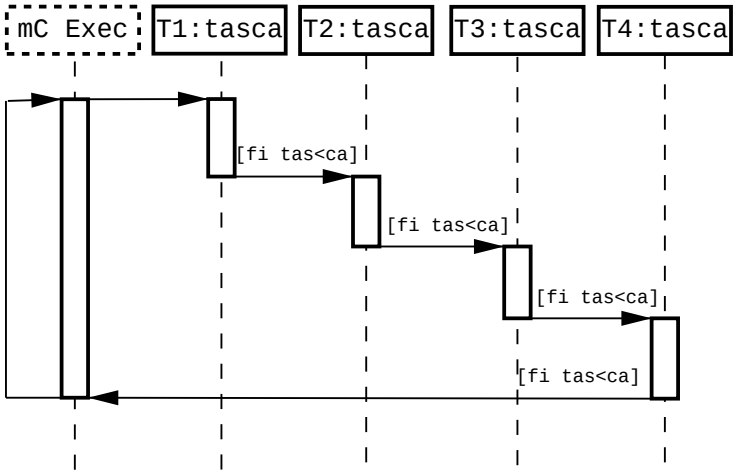
Aquests factors, fan que els programes siguin difícilment adaptables a altres entorns, o el correcte funcionament en el pròpi entorn d'orígen amb petites variacions.

Processos

La utilització del sistemes encastats s'ha realitzat per a tasques molt específiques. Aquest fet, permet simplificar la planificació de l'execució del sistema de control utilitzant algun dels models següents

- **Bucle infinit**

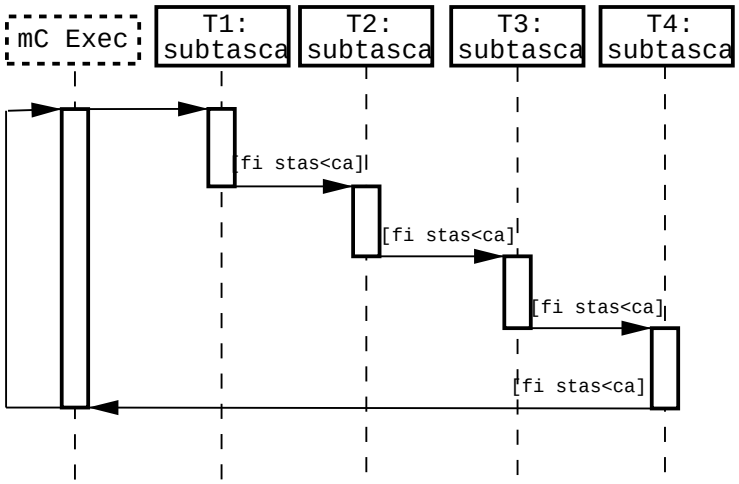
  El conjunt d'accions que defineixen una tasca s'executen de forma seqüencial. Quan s'executa l'última acció de la seqüència, l'execució continua altre cop al principi. L'esquema es reprodueix indefinidament



  Aquesta model pot servir per alguns sistemes encastats en el quals el temps d'execució de la seqüència completa és suficientment ràpid. També cal tenir en compte que aquest model no implementa interrupcions, i per tant, la interacció amb els dispositius es realitza per enquesta.

- **Execució cíclica bàsica**

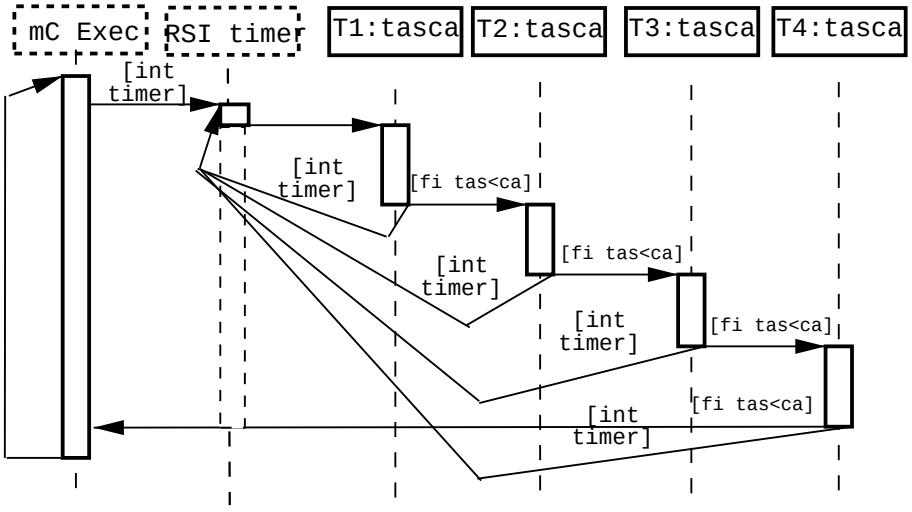
  Aquest model és molt similar al bucle infinit. En aquest cas s'agrupen les accions en subtasques que permeten modularitzar l'execució de la general. En el model, l'execució de les subtasques també és seqüencial i indefinida i s'utilitza la memòria compartida per comunicar unes amb les altres.



Aquest model tampoc implementa interrupcions, i per tant, la interacció amb els dispositius es realitza per enquesta.

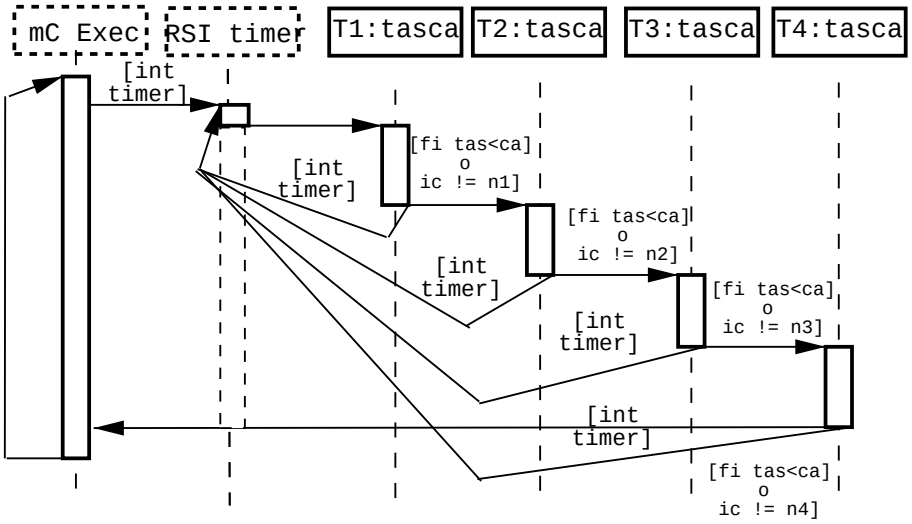
- **Execució cíclica limitada per temps**

  Aquest model utilitza una interrupció de temps que provoca l'execució de la tasca. L'execució de les accions es realitza de manera seqüencial igual que en el cas del bucle infinit , i només es veurà interrompuda per la interrupció de temps que reiniciarà l'execució.



Aquest model tampoc implementa interrupcions, i per tant, la interacció amb els dispositius es realitza per enquesta.

- **Execució cíclica limitada per temps proporcional** Aquest model és molt similar a l'execució cíclica limitada per temps. En el cas anterior, era probable que totes les accions s'executéssin en cada període, en aquest model es planteja la possibilitat que les accions s'executin en diferent períodes de temps. El reinici d'execució és provocat per una interrupció de temps, però l'execució d'un acció depèn del nombre de vegades que la interrupció succeeixi.



Aquest model tampoc implementa interrupcions, i per tant, la interacció amb els dispositius es realitza per enquesta.

Els models anteriors es caracteritzen, bàsicament, per tractar-se d'execucions cícliques. Utilitzant la interrupció de temps s'assegura una execució seqüencial de tasques, execució en diferents intervals de temps i comunicació entre tasques.

D'altra banda, l'execució cíclica té els següents defectes:

- La interacció amb els dispositius es fa per enquesta enlloc de per interrupció
- L'interval d'execució de les tasques no es pot determinar de manera exacta i, per tant, provoca una imprecisió en l'execució. La dependència de finalització d'una tasca en un execució seqüencial, provoca les inexactituds temporals

L'execució de múltiples processos de forma no seqüencial utilitzant models preemptius, augmenta el control sobre l'execució de les tasques.

La utilització de polítiques de temps real en els model de múltiples processos, millora el determinisme de l'execució de les tasques, adaptant els models a la realitat dels processos industrials.

Partint de les dues premises anteriors, es pot augmentar la flexibilitat dels sistemes encastats.

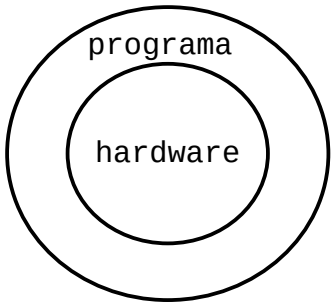
2.2 Funcionalitat del projecte

2.2.1 Modificacions del model general de sistemes encastats

Segons les característiques generals dels sistemes encastats exposades anteriorment, sorgeixen un conjunt de possibles millores que resolen alguns dels plantejaments anteriors.

Sistema operatiu

En els sistemes plantejats durant la carrera, les tasques que es programaven, interaccionaven directament amb el hardware tal i com s'exposa en la figura



Aquest fet, condiciona l'elaboració de programes de control en els següents punts:

- **Coneixement complet del hardware**

  Al programar directament sobre el hardware, cal disposar de les especificacions exactes de les característiques del hardware que s'utilitza. Al treballar tan a baix nivell, les tasques més simples parteixen d'un nivell de coneixement molt alt.

- **Uniplataforma**

  Al programar directament sobre el hardware específic, en cas de canviar qualsevol característica del hardware que s'utilitza, cal modificar el programa utilitzat.

- **Protecció**

  Al programar directament sobre el hardware, és té un accés complet a les seves característiques. Per la mateixa raó que s'exposa al primer punt, és difícil tenir present la totalitat de les especificacions, i qualsevol error en el programa, afecta directament al hardware i la seva execució.

En els punts anteriors, el sistema operatiu millora la gestió del maquinari en els següents aspectes:

- **Gestió del hardware**

  El sistema operatiu coneix i implementa les característiques del hardware sobre el que s'està executant. El programa cedeix la part d'interacció amb el hardware al sistema operatiu, que unificant l'accés al primer, simplifica la complexitat del programa. Dirigeix el processador en la utilització d'altres recursos del sistema i en el control del temps d'execució.

- **Multiplataforma**

  El fet que els programes no interaccionin directament amb el hardware, fa que la seva execució no hi estigui condicionada. És a dir, en el cas que el sistema operatiu s'implementi per un hardware diferent, els programes desenvolupats es mantindran vàlids.

- **Accés als dispositius d'entrada i sortida**

  Cada dispositiu d'entrada i sortida disposa d'un conjunt propi i específic d'instruccions o de senyals de control. El sistema operatiu és capaç de gestionar les característiques específiques del dispositiu, alliberant la tasca al programa i, per tant, facilitant el seu accés de forma simplificada.

- **Execució de programes**

  En el procés d'execució d'un programa s'han de realitzar diferents tasques prèvies. Les instruccions i les dades s'han de carregar a la memòria principal, s'han d'incialitzar els sistemes d'arxius i els dispositius d'entrada i sortida, i la totalitat de recursos que precisi el programa.

- **Detecció i resposta als errors**

  En l'execució de qualsevol programa de control es poden produir diferents errors. Els errors es poden dividir en externs i interns. Els errors poden ser deguts a problemes del maquinari com errades en el funcionament del dispositiu, errors de lectura, inconsistència dels estats, etc. En cada casa, el sistema operatiu dóna una resposta que elimina la condició d'error amb el menor impacte possible sobre les aplicacions que s'estan executant.

- **Comptabilitat**

  El sistema operatiu recull informació sobre l'execució i sobre els dispositius que formen el maquinari, d'aquesta manera, és possible disposar d'estadístiques del seu funcionament.

- **Creació de programes**

  El sistema operatiu ofereix una varietat de característiques i serveis, tals com editors i depuradors, per ajudar al programador en la creació de programes. Aquestes utilitats no formen part del propi sistema operatiu, però és a través d'aquest que és possible accedir-hi.

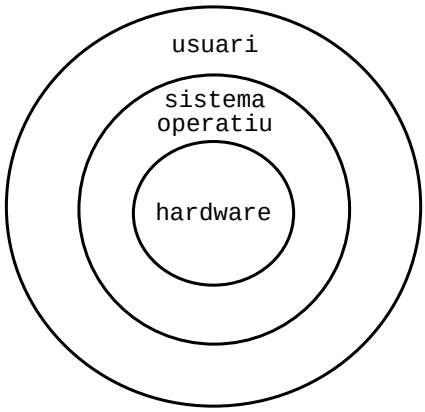
- **Accés controlat als arxius**

  En el cas dels dispositius d'emmagatzemament, s'ha de coordinar la gestió del maquinari i les seves característiques físiques amb l'estructura i característiques del mètode d'emmagatzematge. A través d'aquest control, el sistema operatiu gestiona el multiaccés a arxius, problemes de consistència, etc.

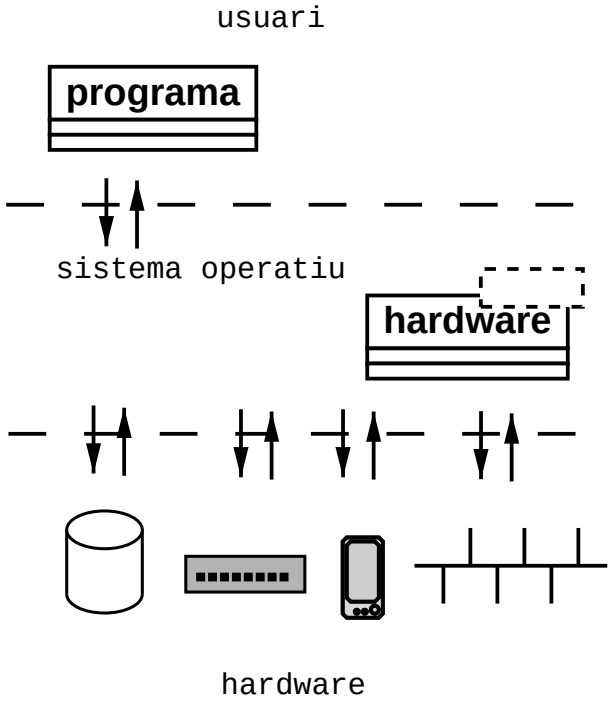
- **Accés al sistema**

  El sistema operatiu controla l'accés als diferents dispositius com també a l'execució de programes.

El nou model que es planteja és el de la figura següent



A nivell pràctica l'esquema seria el de la figura

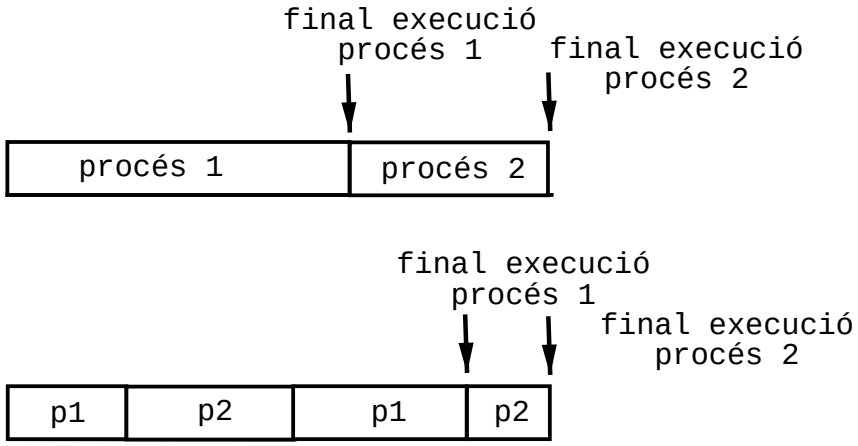


Multiprogramació

En els sistemes plantejats durant la carrera, les tasques eren un únic procés seguint els models que s'exposen a l'apartat de processos cíclics. Les pròpies conclusions de l'apartat porten a plantejar un nou model on es resolguin alguns dels problemes que s'hi plantegen.

En el cas dels sistemes multiprogramació, el model es basa en l'execució virtual de diferents processos en paral·lel, mantenint l'execució real d'un únic procés. És a dir, l'accés al processador està limitat a un únic procés, però aquest procés pot ser diferent en cada moment en funció dels interessos del sistema sense la necessitat que hagin acabat d'executar-se.

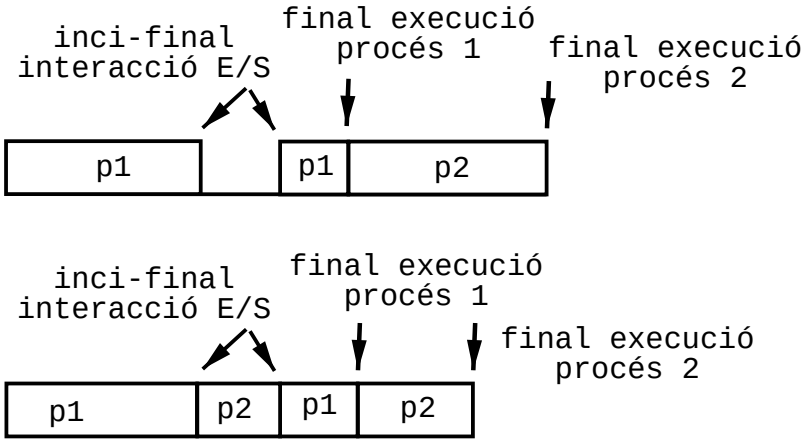
La comparació entre el model uniprocés i multiprogramació es pot observar a l'esquema



En el cas d'un únic procés, es pot donar el cas que l'execució de les tasques quedi bloquejada en funció d'una interacció amb els dispositius d'E/S. En aquesta situació, no es pot complir l'execució de la totalitat de les tasques que componen la seqüència d'una execució cíclica. Pel mateix motiu, no s'utilitza la totalitat de recursos com pot ser el cas del processador.

En el model multiprogramació, es mantenen ocupats el processador i els dispositius d'E/S, ja que en algun moment una o altra tasca s'està executant. D'aquesta manera, s'aconsegueix la major eficiència possible.

La comparació entre el model uniprocés i multiprogramació es pot observar a l'esquema



Comunicació entre processos

En els sistemes plantejats durant la carrera s'utilitzaven models d'únic procés i execució cíclica, per la qual cosa es podia utilitza la memòria compartida per passar informació entre tasques. En el model de multiprogramació, la comunicació entre processos es complica i cal tenir present els següents factors

- **Sincronització incorrecta** Dependència errònia entre tasques
- **Exclusió mútua** Accés únic a un recurs compartit. Exclusió d'accés
- **Interbloqueig** Bloqueig mutu de dues tasques degut a dependències errònies

La solució als problemes anteriors es basa en la utilització d'elements coordinadors de tasques o recursos compartits com

- Banderes
- Semàfors
- Monitor
- Missatges

Els conceptes anteriors s'ampliaran en un apartat posterior.

Temps Real

En els sistemes plantejats durant la carrera, no es contemplava el temps de resposta del sistema de control. A l'utilitzar un únic procés, es dissenyava de tal manera que complís tots els requisits de temps i de correcció de la sortida. D'alguna manera, el concepte del temps real ja s'incloia mentalment en el disseny estàtic del sistema. Amb la introducció de diferents tasques i del dinamisme del sistema (variable hardware, etc.), l'execució en temps real assoleix nous nivells de complexitat que fan que la seqüència d'execució no es pugui dissenyar de manera estàtica sense tenir present la relació entre tasques i aquestes amb el hardware.

El processament en temps real es pot definir com un tipus de processament en el qual l'exactitud del sistema, no només depèn del resultat lògic d'un càlcul, sinó també del compliment del temps d'execució i del nivell de correcció del resultat. Vegeu exemples de l'aplicació del temps real en sistemes operatius en les implementacions realitzades per Labrosse [15] i les modificacions sobre Minixi realitzades per Wainer exposades en el document [30].

Amb el processament en temps real, el sistema guanya en les següents qüestions:

- Determinisme
- Sensibilitat
- Control de l'usuari
- Fiabilitat
- Tolerància a errades

Els conceptes anteriors s'ampliaran en un apartat posterior.

2.3 Entorn de desenvolupament

En el procés de desenvolupament del projecte s'utilitza un conjunt d'eines per tal d'emular el sistema de control final, vegeu [14]

2.3.1 Microcontrolador 8051

El hardware del sistema encastat plantejat es basa en l'arquitectura del 8051 estàndard d'intel.

La tecnologia del 8051 sorgeix a principis dels 80 a mans de l'empresa INTEL. Hereva del 8048 i està desenvolupada utilitzant tecnologia HMOS. Inicialment ja es desmarquen tres branques diferents

- 8051 Arquitectura base amb una ROM interna programada de fàbrica
- 8751 Arquitectura base equipada amb una EPROM
- 8031 Arquitectura base utilitzant memòria ROM externa

Més tard apareix el 8052, hereu de l'arquitectura deol 8051 amb un tercer perifèric i 128 bytes de RAM interna. Tots aquests models integren l'anomenada sèrie del MCS-51.

La multiplicitat dels models es justifica per un concepte que ja s'ha tingut en compte abans de l'aparició dels 8051: Utilitzar una única unitat central modificant únicament l'entorn en el que s'integra. La unitat de la família està garantitzada pel fet que totes les noves versions es construeixen al voltant del mateix nucli de programari i hardware. Aquesta perspectiva incita a diferents fabricants com Siemens, MHS, OKI, etc. a desenvolupar productes amb aquesta arquitectura. Dissenyar un nou microcontrolador de la família dels 8051 consisteix en completar la màscara de fabricació del 8051 amb noves funcions perifèriques.

2.3.2 Compilador

En el desenvolupament del projecte, s'utilitza com a compilador el "Keil C51 C Compiler", vegeu el tutorial d'introducció [7] i la documentació oficial referent a utilitats [11], ensamblador [12] i compilador de C51[13]. El compilador genera el fitxer objecte (veure format a [8]) que gestionarà el sistema de control, i permet incorporar-hi informació addicional per a la corresponent depuració utilitzant l'emulador o a través del propi circuit de control. El compilador té les següent característiques:

- Implementa nou tipus de dades diferents, inclòs el punt flotant de 32-bits
- Assignació flexible de les variables en els diferents tipus de memòria: bit, data, bdata, idata ,xdata i pdata
- Les interrupcions es poden programar en C
- Accés a la totalitat dels bancs de registres
- Informació addicional per la posterior depuració
- Utilització de les instruccions AJMP i ACALL
- Objectes direccionables bit a bit

- Suport pels punters de dades duals dels microcontroladors Atmel, AMD, Cypress, Dallas, Semiconductor, Infineon, Philips i Triscend
- Suport pel conjunt d'instruccions del Philips 8xC750, 8xCT51 i 8xC752
- Suport per la unitat aritmètica de Infineon 80C517

Com que l'arquitectura del 8051 ha estat implementada per múltiples empreses, el compilador ha de suportar el model específic de microcontrolador desenvolupat per cada una. Les empreses de les quals suporta models de microcontroladors són:

Acer LAB	AerFlex UTMC	Analog Devices
Anchor Chips	Atmel	Chipcon
CML Microcircuits	Cybernetic Micro Systems	CyberTech
Cygnal Integrated Products	Cypress Semiconductor	Daewoo
Dalla Semiconductors	Domosys	Goal Semiconductor
HoneyWrill	Hynix Semiconductor	Hyundai
Infineon	InnovASIC	Intel
ISSI	Maxim	Mentor
Micronas	MXIC	Myson Technology
OKI	Oregano Systems	Philips
Sanyo	Sharp	Silliconians
SMSC	SST	ST Microcontrollers
TDK	TI	Triscend
Winbond	_	_

Taula 2.1: Llistat d'empreses que fabriquen microcontroladors de l'arquitectura del 8051

Per a més informació sobre el microcontrolador 8051 vegeu els llibres d'introducció la seva arquitectura escrits per d'Odant [19] i Schultz [22]

2.3.3 Emulador

En el desenvolupament del projecte, s'utilitza l'emulador de 8051 "Keil uVision2", vegeu [14]. Aquest entorn permet l'administració de projectes, facilita els procés de compilació, edició del codi font, depuració del programa i una emulació completa de l'entorn del 8051. L'entorn té les següent característiques:

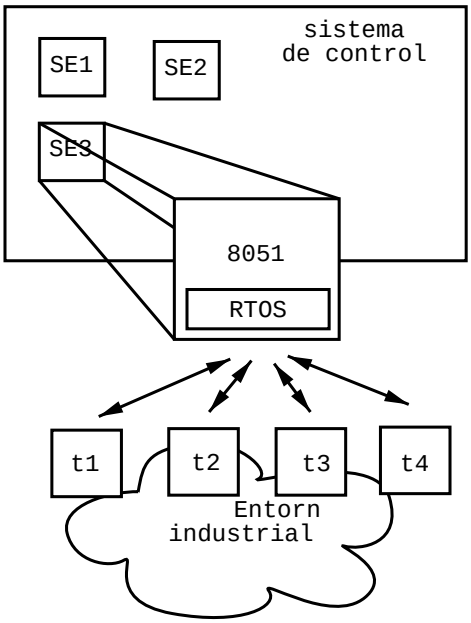
- Base dades amb les característiques de tots els models de microcontroladors, fet que permet optimitzar la compilació
- Administració de projectes. Els projectes inclouen codi font, objectes, etc.

- Sistema integrat de planificació de la compilació que gestiona la totalitat dels elements del programa
- Resolució de problemes en calent en temps de compilació
- Simulació completa en el PC
- Depuració del programa en el microcontrolador utilitzant l'ordinador com a monitor
- Depuració del programa en el microcontrolador utilitzant un emulador
- Breakpoint (simples, complexes i condicionals)
- Informació completa per la depuració
- Watchpoints
- Emulació de conversors A/D - D/A
- Emulació del port d'E/S
- Emulació d'interrupcions
- Emulació de temporitzadors i comptadors
- Emulació del port sèrie
- Emulació de temporitzadors Watchdog

2.4 Estudi d'implementació

A partir de la fusió dels diferents conceptes que s'han descrit fins el moment, s'arriba a assumir bona part dels requisits que demana un sistema de control industrial. El projecte, intenta desenvolupar, partint de la cap de més baix nivell, tot un sistema de control de processos en un 8051 en temps real, utilitzable en sistemes encastats.

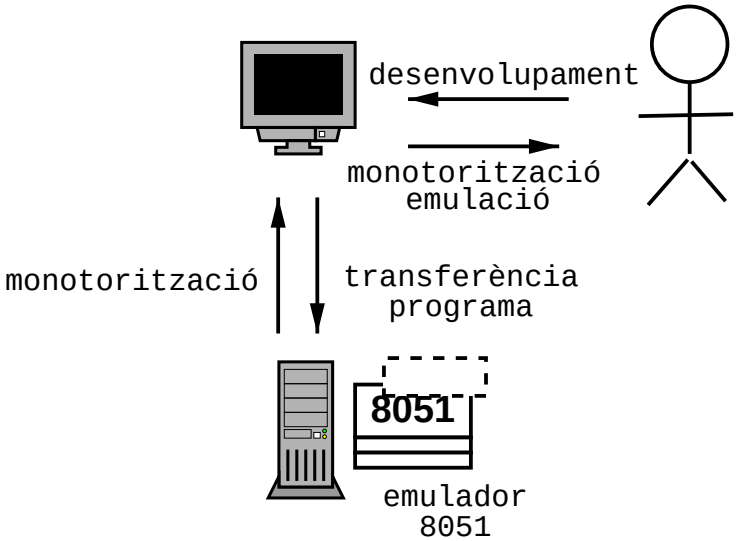
L'esquema de l'entorn es descriu en la figura



El procés de desenvolupament és el següent:

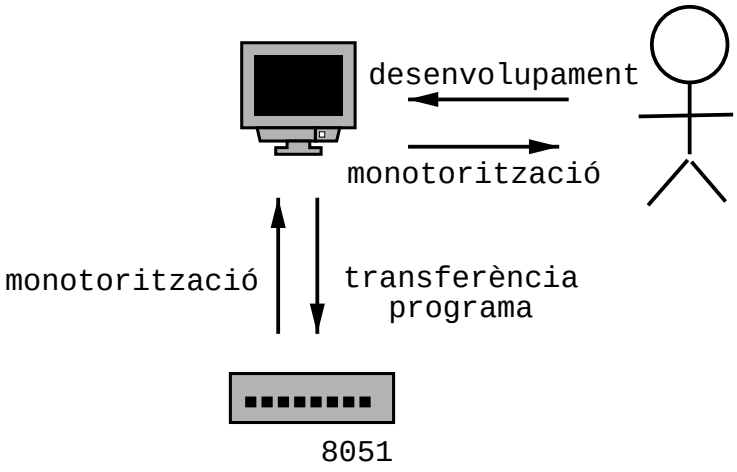
1. Desenvolupament del sistema

   El desenvolupament es realitza sobre una plataforma PC, utilitzant l'emulador i compilador del sistema.



2. Monitorització del sistema real

   El sistema desenvolupat s'executa sobre la plataforma real del 8051, utilitzant el PC com a monitor.



3. Sistema autònom

   S'instal.la definitivament el sistema en el 8051, i s'inicia en producció.

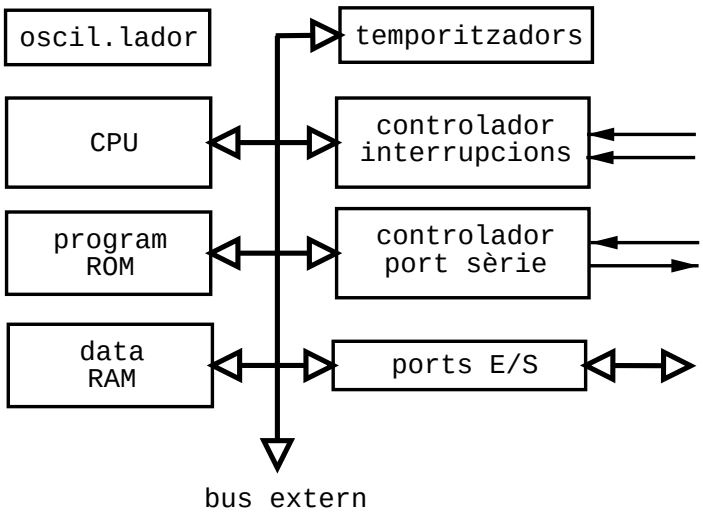
Capítol 3

Conceptes teòrics del model a implementar

3.1 Microcontroladors

3.1.1 Descripció

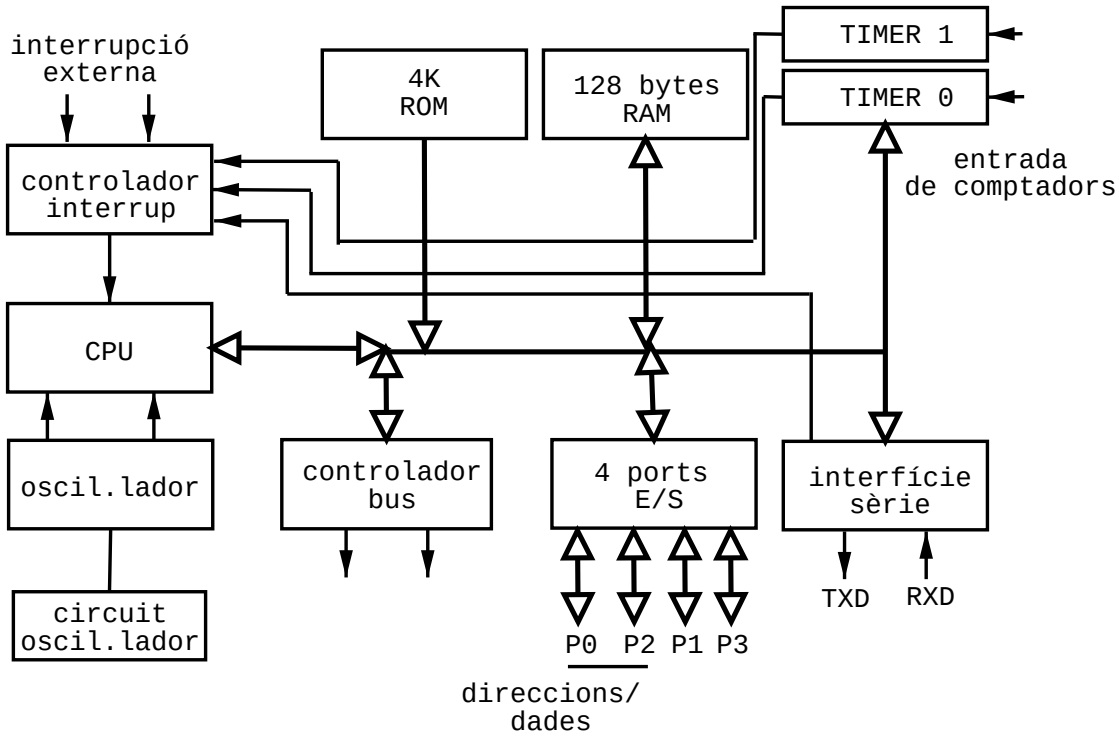
En moltes de les instal·lacions on s'utilitzen sistemes encastats com a sistema de control, els fabricants de circuits integrats utilitzen microcomputadors amb tots els components necessaris integrats en un sol xip. Aquest xip disposa d'una petita quantitat de memòria ROM i RAM, tot i que és ampliable. Un dispositiu que segueix aquestes directives és el microcontrolador. La diferència entre un microcomputador i el microcontrolador és que aquest últim incorpora algun dispositiu perifèric de suport integrat a la placa, disposa d'un rellotge de temps real i un temporitzador watch-dog. Per més informació sobre arquitectures vegeu [26].



En el desenvolupament de projectes utilitzant dispositius de la família MCS-51, cal tenir present les restriccions i interpretacions que han fet l'empresa KEIL en el desenvolupament del seu compilador i emulador.

3.1.2 Nucli del 8051

El nucli central de l'arquitectura del família MCS-51 és el de la figura següent:



3.1.3 CPU

Els elements bàsics de la unitat central de processament són

- **Unitat de control**

  Element capaç de recuperar un codi d'instrucció, descodificar-lo i començar una seqüència adequada per la seva execució. La major part d'aquestes seqüències duren només un cicle de màquina.

- **Unitat aritmètico-lògica**

  Aquest element permet realitzar multiplicacions i divisions amb números de 8 bits, com també realitzar les operacions lògiques més comunes.

3.1.4 Memòria

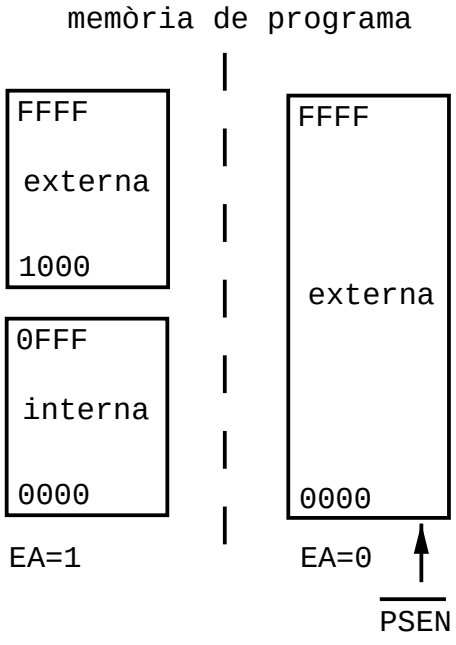
Memòria de programa

Memòria on hi ha emmagatzemat tot el codi màquina executat pel processador. Només és accessible per la seva lectura i està ubicat en una memòria externa o interna de tipus ROM/EPROM. L'accés es realitza a través del comptador de programa que funciona com a punter de la seqüència d'execució i del punt de dades.

La memòria de programa pot estar integrada en una ROM o EPROM, col·locada al principi de l'espai de direccionament, o em una ROM externa, l'accés a la qual s'indica mitjançant la senyal de control PSEN. Durant la fase d'inicialització, el microcontrolador comprova l'estat de la pota EA, la qual indica el tipus d'accés (intern/extern) a l'espai inferior de la memòria de programa.

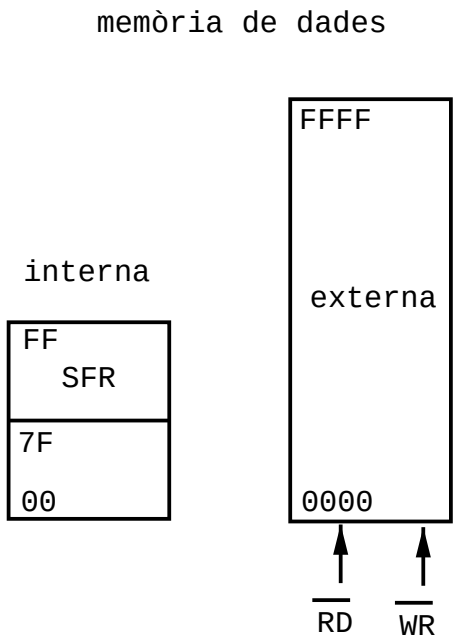
Pels microcontroladors que no disposen de memòria de programa interna, cal forçar l'estat de la pota EA obligant el processador de buscar les instruccions del programa a la memòria externa.

Després d'una operació d'inicialització, el comptador de programa conté el valor 0, en conseqüència, el processador recupera la primera instrucció que ha d'executar d'aquesta direcció. En la part inferior de l'espai de memòria de programa hi ha una zona dedicada als serveis d'interrupció. Les direccions d'aquesta zona corresponen als vectors de diferents fonts d'interrupció. La zona assignada a cada servei d'interrupció és de 8 bytes, i permet saltar a una altra zona de memòria en cas que no sigui suficientment gran per gestionar l'interrupció.



Memòria de dades

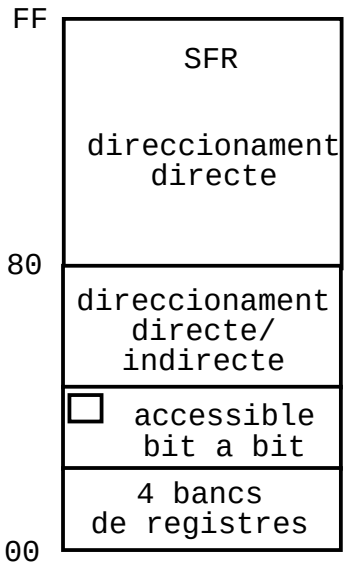
Memòria on hi ha emmagatzemades totes les dades que utilitza el programa i, en conseqüència, que utilitza el processador. És accessible per la seva lectura i escriptura i està ubicada en una memòria externa o interna de tipus RAM.



Aquesta memòria es divideix en dues parts de 128 bytes.

- Part baixa de la memòria interna 0-7Fh

- SFR. Registres amb funció especial 80h-FFh



3.1.5 Espais de memòria

La disposició física dels espais de memòria en un 8051 es divideixen en els següents apartats:

1. Segment de dades
2. SFR
3. Segment de dades amb direccionament indirecte
4. Segment de codi
5. Segment de dades extern

Segment de dades

L'espai de memòria està definit per l'interval 00-07F i s'utilitza per emmagatzemar dades del programa. La zona de memòria és accessible mitjançant direccionament directe.

S'utilitza per emmagatzemar les dades a les quals s'accedeix de manera freqüent i que precisin d'alta velocitat d'accés. En el cas de les interrupcions, al precisar d'una ràpida resposta, s'han de declarar en aquest espai de memòria. En cas d'utilitzar funcions reentrants, les piles d'aquestes tasques també s'han de situar en aquest espai.

En aquest espai no s'hi han de declarar les taules i estructures complexes que ocupin més d'uns quants bytes.

SFR

Els registres de funció especial, són posicions de memòria internes del 8051 que permeten controlar els perifèrics integrats en el xip com ports, temporitzadors, interrupcions i característiques del processador. La zona de SFR, només és accessible mitjançant direccionament directe. Els diferents registres apareixen a la taula 3.1.5

Registre	Descripció	Memòria
SP	Punter de pila	81
DPTR	Punter de dades	82-83
PCON	Power control	87
TCON	Control del temporitzador	88
TMOD	Mode del temporitzador	89
Tx0	Temporitzador 0	8A - 8C
Tx1	Temporitzador 1	8B - 8D
P1	Port 1	90

Taula 3.1: Taula de registres de funció especial

Segment de dades amb direccionament indirecte

L'espai de memòria es sobreposa, en direccionament, al SFR. Aquesta zona constitueix i amplia l'àrea de on-chip RAM que és la millor per utilitzar com a pila degut al direccionament indirecte del punter de memòria.

En aquest espai, s'hi situen les estructures lleugerament complexes (sobre 36 bytes però inferior a 64 bytes) que precisin que la velocitat d'accés sigui un factor a tenir en compte. Les pròpies característiques del direccionament d'estructures, igual que en el cas de la pila, els fan idonis.

L'espai no és recomanable en el cas de taules de gran magnitud o variables que precisin d'una alta velocitat d'accés.

Segment de codi

L'espai de memòria està reservat per codi i s'hi accedeix a través de l'execució d'instruccions seguint el comptador de programa o a través del punter de dades (DPTR).

S'utilitza per emmagatzemar constants i taules complexes.

Segment de dades extern

L'espai de memòria de dades situat en un xip de memòria externa al qual s'accedeix a través del DPTR.

S'utilitza per emmagatzemar taules complexes de variables i estructures i també variables a les quals s'accedeix de manera poc freqüent. També s'utilitza per la gestió de variables, que es monitoritzen en temps real a través d'un emulador.

L'espai no és recomanable per a variables a les quals s'ha d'accedir de manera freqüent o variables d'interrupció.

Utilitzant el registre R0(8 bits) 256 byte d'aquesta zona es poden direccionar de manera paginada, definint la zona PDATA. Aquesta zona s'utilitza per interrupcions de mitja velocitat, variables de tipus caràcter (8bits) i estructures i taules d'una magnitud moderada.

La zona PDATA no és recomanable per taules de dades, estructures de mida superior a 256 bytes ni dades a les qual s'accedeixi de manera molt freqüent.

3.1.6 Models de memòria

SMALL (RAM total 128 bytes)

S'utilitza en els desenvolupament en els que hi ha únicament un 8051, sense memòria de programa ni dades externa. Cal tenir presents les restriccions següents:

- Minimitzar la utilització de variables globals
- El codi no pot ser superior a 4K i cal verificar constantment l'estat de la pila

Totes les variables i paràmetres de les les funcions estaran en la memòria interna del 8051

COMPACT (RAM total 256 bytes off-chip, 128 o 256 bytes on-chip)

Es pot utilitzar en desenvolupaments en els quals s'utilitza un sistema operatiu emmagatzemat en la memòria on-chip; combinant el model COMPACT i el SMALL per les rutines d'interrupció. És especialment útil en entorns en els que s'utilitzin variables de 8 bits i s'utilitzi molt la pila i, per tant, emmagatzemant les dades off-chip.

Aquest model parteix de l'estat del port 2. El segment de dades extern es direcciona a través del punter de dades (DPTR) el qual situa l'adreça de 16 bits als ports 0 i 2. En el model COMPACT, s'utilitza el registre R0 com a punter de 8 bits que situa l'adreça al port 0. El port 2 està controlat per l'usuari i s'utilitza per la gestió de les pàgines.

Les variables es guarden en adreces de les pàgines de memòries seleccionades a través del port 0 i 2.

LARGE (RAM total fins a 64K off-chip, 128 o 256 bytes on-chip)

S'utilitza en desenvolupaments en els quals s'ha d'accedir a una gran quantitat de memòria i també s'acostuma a utilitzar en combinació amb el model SMALL.

Les variables estan emmagatzemades en la memòria externa direccionada a través del DPTR. Els registres del xip es continuen utilitzant per a variables locals i els arguments de les funcions.

3.1.7 Ports d'entrada i sortida

La família MCS-51 disposa de 4 ports, P0, P1, P2 i P3 que constitueixen un conjunt de 32 línies bidireccionals que tan es poden utilitzar per entrades com per sortides, sense la necessitat de cap procés d'inicialització. Cada una d'aquestes línies correspon a una pota del circuit integrat. El registre associat a cada port, està constituït per vuit bàscules de tipus D.

3.1.8 Temporitzadors

Cada un dels temporitzadors del 8051 disposa d'un comptador de 16 bits accessible en forma de dos registres de 8 bits que són configurables a través dels registres TMOD i TCON.

- TH0 i TL0 pel timer 0
- TH1 i TL1 pel timer 1

Un timer pot realitzar dues funcions diferents

- **Comptador** Un comptador incrementat fruit de la detecció d'events externs
- **Temporitzador** El comptador s'incrementa a partir de la senyal de rellotge del microcontrolador

3.1.9 Port sèrie

El port sèrie és accessible a través de la funció secundària del port P3. El terminal RXD per la recepció i el terminal TXD per transmissió. El port sèrie és capaç de rebre i transmetre simultàniament. També disposa d'un buffer que permet la recepció d'un byte abans de la lectura anterior. El port sèrie pot funcionar en els quatre modes següents:

- Registre de desplaçament. El port sèrie funciona com un registre de desplaçament de 8 bits
- UART 8 bits. El port sèrie funciona com un UART de 8 bits amb un bit d'arrancada i un bit de parada
- UART 9 bits. El port sèrie funciona com un UART de 9 bits amb un bit d'arrancada i un bit de parada
- UART 9 bits. El port sèrie funciona com en el mode 2 amb velocitat de transmissió variable

3.2 Sistema operatiu

3.2.1 Descripció

L'execució del sistema operatiu és similar a qualsevol altre programa amb accés al processador. La diferència essencial, és que disposa d'accés il·limitat als recursos del sistema; aquesta característica li permet gestionar i coordinar la utilització d'aquests recursos per part d'altres programes en execució. Per més informació sobre sistemes operatius,vegeu el llibres de referència de Tanenbaum [28] i Stallings [25].

La majoria de processadors suporten dos mètodes d'execució

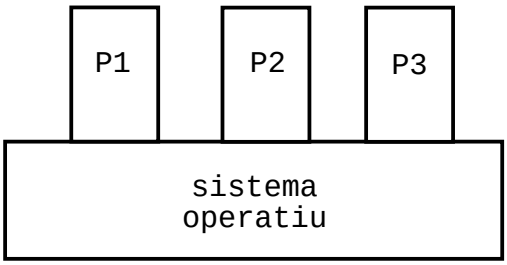
- **Mode nucli de sistema operatiu** Accés complet al control del processador, registres, memòria, etc.
- **Mode d'usuari** Accés restringit als recursos

En les diferents estructures que defineixen un sistema operatiu, hi ha els tres models bàsics següents:

- Sistema operatiu independent a qualsevol procés

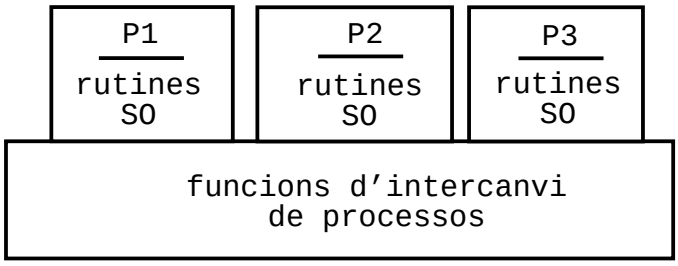
  Durant l'execució de qualsevol procés, en cas de produïr-se una interrupció o una crida al sistema, es salva l'entorn del procés i el sistema operatiu assumeix el control de l'execució i

de la màquina. Al disposar del control total, pot executar i accedir a qualsevol funcionalitat de la màquina i retornar al procés que s'executava anteriorment. El sistema operatiu té una regió de memòria pròpia i també una pila pròpia per gestionar els arguments de les rutines.



- **Sistema operatiu en un procés d'usuari**

  El sistema operatiu és una col·lecció de rutines que crida l'usuari i que s'executa en l'entorn del procés d'usuari. En qualsevol moment de l'execució del sistema operatiu gestiona n imatges de processos. Cada imatge conté l'entorn del procés de l'usuari, les zones de programes, les dades i la pila.



  La pila del nucli del sistema operatiu s'utilitza per administrar les crides i el retorn de valors mentre el procés executa les tasques del sistema operatiu (mode nucli). Aquest últim, utilitza l'espai de memòria compartit entre processos per intercanviar dades i codi.

- **Sistema operatiu basat en processos** El sistema operatiu s'implementa com una col·lecció de processos d'usuari, tot i que les rutines del nucli del sistema operatiu s'executen en mode de nucli. La major part de les tasques es realitzaran en mode de l'usuari, deixant petites parts com l'administració i comunicació de processos al nucli del sistema. El sistema és molt modular i especialment idoni per sistemes de multiprocessadors treballant en paral·lel.

3.2.2 Estructures de dades del sistema

Entenent el procés com la base d'execució de tasques, cal tenir present les característiques de l'entorn que defineixen el propi sistema. Aquestes característiques inclouen des de qüestions físiques de la pròpia màquina, com qüestions lògiques de la seqüència d'execució. El sistema operatiu manté un conjunt d'estructures de control que es representa a continuació.

- **Taula de processos**

  Conté la informació sobre els diferents processos del sistema

- **Taula de memòria**

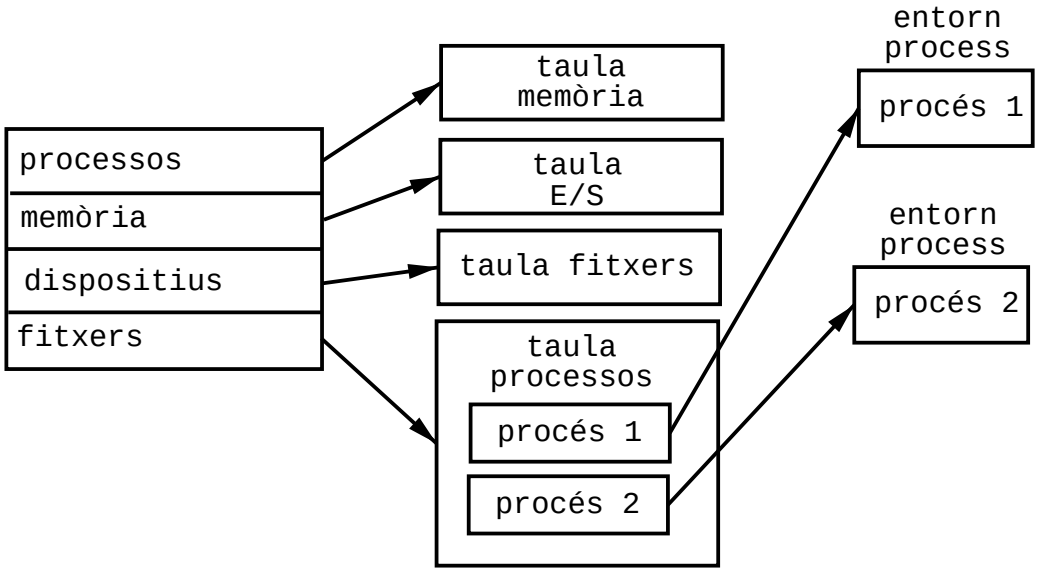
  Conté la informació referent a la gestió de la memòria (principal-secundària), inclosa la pròpia gestió de la memòria del sistema operatiu, la memòria assignada a cada procés, la protecció d'accés específic a cada segment i qualsevol informació addicional per l'administració

- **Taula d'entrada-sortida**

  Conté informació sobre els dispositius d'entrada-sortida, la corresponent assignació i estat per cada un dels processos, la posició a memòria, etc.

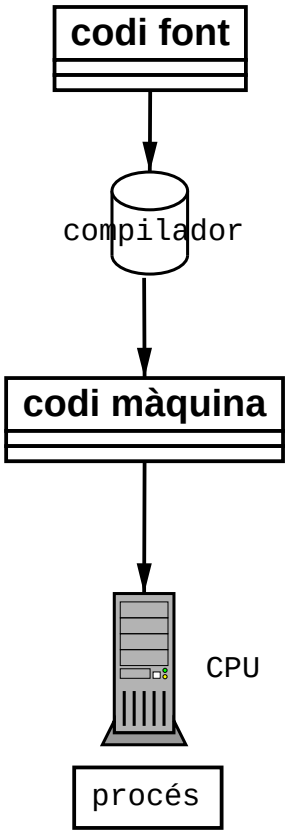
- **Taula de fitxers**

  Conté informació sobre l'estat dels fitxers oberts, propietaris, posició de memòria i atributs



3.2.3 Procés

En el desenvolupament de qualsevol projecte que impliqui l'execució d'una sèrie de tasques sobre un hardware determinat, es parteix de la descripció d'aquest conjunt de tasques. Aquesta descripció s'acostuma a realitzar a través del codi font, que implementat amb qualsevol llenguatge, i posteriorment compilat, genera la seqüència d'instruccions que interpretarà la màquina. El concepte procés, s'entén com el conjunt d'instruccions en estat d'execució.



En el model de processos seqüencials utilitzat en sistemes operatius, tot el programari executable per la màquina, inclòs el sistema operatiu, està organitzat en una sèrie de processos. Juntament al conjunt d'instruccions en execució, s'inclouen els valors actuals del comptador de programa, els registres i les variables. Des del punt de vista de cada procés individual, disposa de la totalitat de la màquina, talment com si cada un dels processos disposés d'una màquina dedicada. En aquest model, la màquina, i específicament la CPU com a unitat de control, commuta l'execució entre diferents processos implementant així aquesta sensació d'execució en paral·lel.

Estats d'un procés

Durant l'execució de qualsevol procés, i tenint presents les restriccions que imposa el model exposat anteriorment, passa per diferents estats.

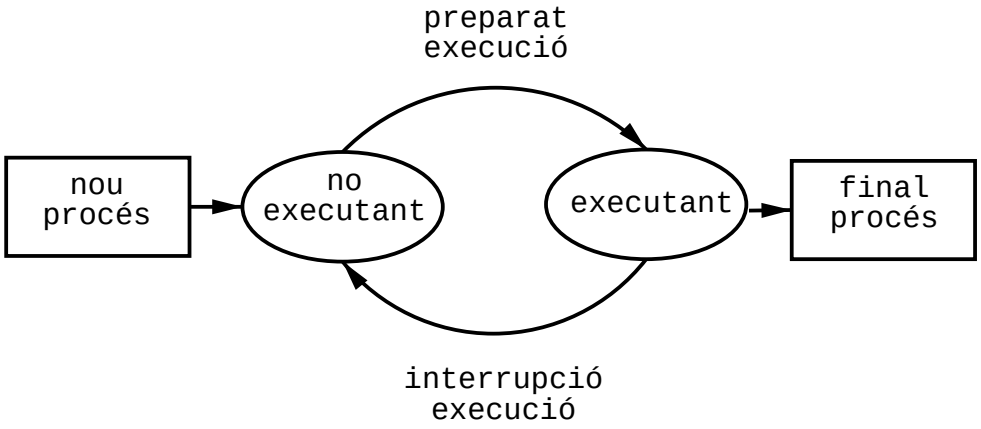
La creació d'un procés pot tenir diferents orígens, i està lligat a l'entorn del sistema de control que gestiona.

- En l'execució seqüencial de tasques que es presenta fins la introducció de la multiprogramació, un nou procés sorgeix de la finalització de l'execució d'un procés anterior

- En el model de multiprogramació, un procés pot inicialitzar un segon procés per realitzar una segona tasca en un pseudoparal·lalisme. Relació entre processos anomenada pare-fill
- En el sistema de control, la interacció amb l'usuari administrador pot generar processos independents

El sistema operatiu gestiona una cua de tasques per cada possible estat. A partir d'aquest mètode pot saber, en cada moment, en quin estat està cada procés El model més simple d'estats de processos, contempla dos estats.

- **Executant** En aquest estat el procés s'està executant i, per tant, disposa d'accés a la CPU, memòria principal i registres
- **No-executant** El procés no s'està executant tot i que el sistema operatiu té constància que el procés està llest per fer-ho



L'execució d'un procés es pot veure interrompuda i passar a l'estat de no-execució, o finalitzada, degut als següents factors

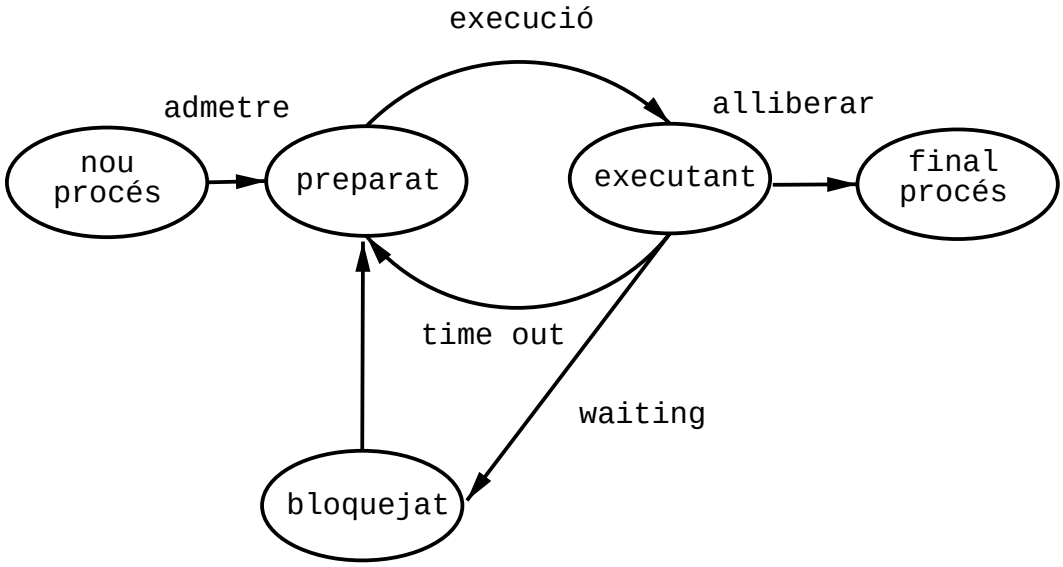
- Finalització normal de l'execució del procés
- Temps d'execució assignat sobrepassat
- Decisió de l'administrador del sistema
- Finalització d'execució del procés pare
- Decisió de l'execució del procés pare
- Memòria insuficient

- Violació del límit de memòria accessible
- Error en l'intent d'accés a un recurs no assignat
- Error aritmètic
- Error dels recursos d'entrada i sortida
- Instrucció no vàlida

Degut a la utilització de recursos per part dels processos, com també la utilització de recursos compartits, cal tenir present que per qualsevol de les dues raons, un dels dos processos pot restar bloquejat. El control de l'execució per part d'un procés bloquejat minimitza el rendiment dels recursos, i és per això que es modifica el model.

El model de cinc estats contempla els següents estats:

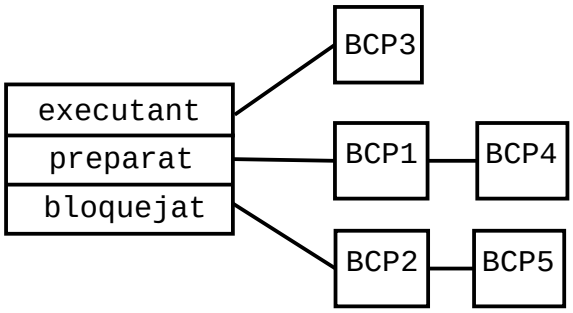
- **Preparat** El procés s'ha acceptat per una posterior execució, i està preparat per fer-ho en qualsevol moment
- **Executant** En aquest estat el procés s'està executant i, per tant, disposa d'accés a la CPU, memòria principal i registres
- **Bloquejat** El procés que s'estava executant ha quedat bloquejat fruit d'alguna dependència externa



Es crea un procés en funció dels orígens exposats anteriorment. Aquest procés passa a l'estat de preparat en funció de les decisions que pren el sistema operatiu, en aquest cas, molt probablement

degudes a les limitacions de la pròpia màquina. En el moment que el sistema operatiu ho considera oportú, extreu de l'estat d'execució un procés, i hi introdueix un de nou de l'estat preparat. L'extracció d'aquest procés es pot deure a la restriccions de temps d'execució que pot assignar el sistema operatiu, o al bloqueig degut a la dependència d'alguna situació de l'entorn com pot ser l'accés a un recurs. En el primer cas, la tasca sortint retorna a l'estat de preparat, esperant un nou accés a l'execució; en el segon cas, la tasca passa a l'estat de bloquejat, en la qual restarà esperant el recurs, i posteriorment passarà a l'estat de preparat, esperant de nou, igual que en la situació anterior, l'accés a l'execució.

La implementació dels diferents estats es realitza a través d'un seguit de cues, o estructures entrellaçades, que permeten seguir tots els processos de cada una de les cues i accedir al bloc de control del procés segons l'esquema de la figura següent.



Cal tenir present que la integritat d'aquestes cues condiciona la totalitat del sistema i, per tant, cal que estiguin protegides. En cas de produïr-se un error en el processament d'algun d'aquests blocs de control, afectaria a la totalitat del sistema. Al dependre els bloc uns amb els altres, dependència de mòduls, l'accés s'ha de restringir a un únic mètode, simplificant així el control d'accés.

Descripció d'un procés

Un procés, com a programa en execució, precisa d'uns certs requisits que el defineixen.

- Variables locals o globals emmagatzemades a memòria
- Constants
- Pila utilitzada per emmagatzemar dades temporals
- Atributs del procés (bloc de control del procés)

Bloc de control del procés

La informació que apareix en el bloc de control de procés es pot classificar segons els següents apartats:

- **Identificació de procés**

  Cada procés té un identificador numèric únic que es representa a través d'un índex a la taula principal de processos que apunta al corresponent bloc de control. Aquest identificador és utilitzat per d'altres mòduls del sistema operatiu per descriure les característiques del sistema (definició de memòria, comunicació entre processos, etc).

Atribut	Descripció
Identificadors	Identificador del procés Identificador del procés pare Identificador de l'usuari

- **Estat del processador**

  L'estat del processador es descriu a través d'un sèrie de registres com els visibles per l'usuari, els de control, els d'estat i els de pila.

Atribut	Descripció
Registres accessibles per l'usuari	Registres visibles
Registres de control i estat	Comptador de programa Banderes d'estat Informació d'estat. Estat de les interrupcions i mode d'execució
Punter de pila	Punter al capdamunt de la pila

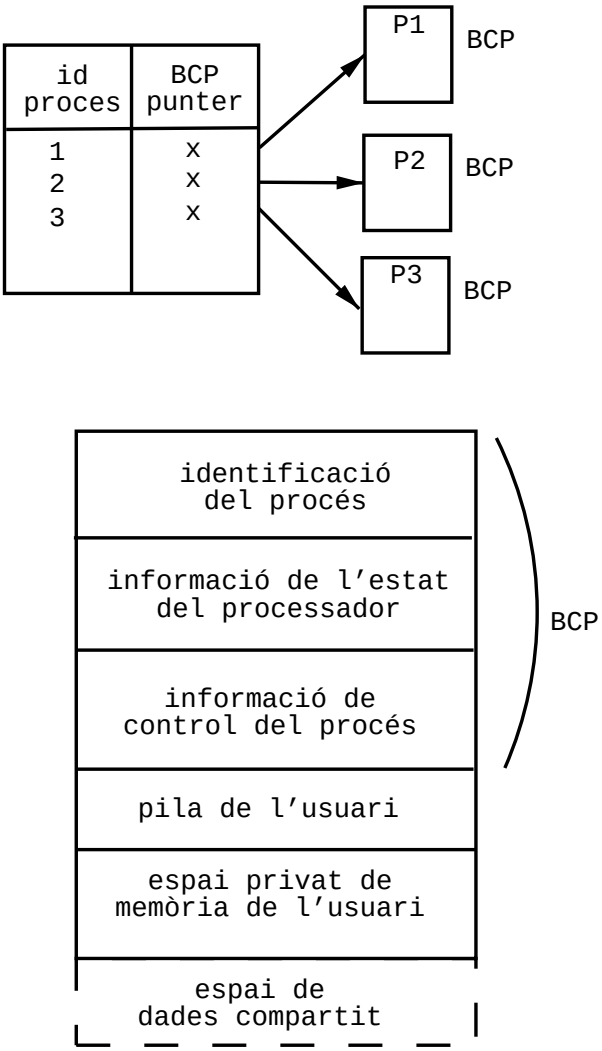
- **Atributs del procés**

Atribut	Descripció
Informació i planificació d'estat	Estat del procés Prioritat Informació de planificació Senyal d'espera de bloqueig
Estructura de dades	Si hi ha diversos processos en la cua d'una mateixa prioritat hi ha una relació de punters
Comunicació entre processos	Dades utilitzades entre processos
Privilegi	
Administració de memòria	Punters a les taules que descriuen els segments o taules assignades a un procés
Propietat i utilització d'un recurs	Recursos associats a un procés

Juntament amb el bloc de control de procés, les dades que defineixen l'entorn d'una tasca són:

- Pila de la tasca
- Espai privat de la memòria (programa i dades)
- Espai de dades compartit

A partir de la taula de processos i la definició d'un procés, l'esquema de gestió de processos és el de la figura



3.2.4 El sistema operatiu i els processos

L'administració dels processos, sempre es realiza sota control del sistema operatiu, aquesta és l'encarregat de

- Crear i eliminiar processos
- Planificar l'execució processos
- Intercanviar l'execució de processos
- Sincronitzar els processos
- Gestionar la comunicació entre processos
- Administrar el bloc de control de procés
- Administrar l'espai de memòria del procés
- Administrar els dispositius d'entrada i sortida

Crear un procés

La seqüència de tasques que realitza el sistema operatiu per crear un procés són:

1. Assignar un identificador únic per a un nou procés, i afegir una nova entrada a la taula principal de processos on hi entrades per a cada un dels processos del sistema
2. Assignar l'espai de memòria pel procés. Aquest espai inclou tots els elements que defineixen la imatge d'un procés, per la qual cosa el sistema operatiu ha de conèixer l'espai de memòria que necessita cada procés i la mida de la pila. Aquesta valors poden ser estàtics o heretats del procés pare que el crea
3. Inicialitzar el bloc de control del procés. La part d'identificació del procés conte l'identificador del procés i del seu pare. La part d'informació del processador s'inicialitza amb valors nuls, exceptuant el comptador de programa i els punters de pila que definiran els límits d'aquesta. La part d'informació de control del procés s'inicia amb valors per defecte
4. Introduir el nou procés en la cua d'estat corresponent i restablir els enllaços corresponents a aquestes cues
5. Crear o modificar altres estructures del sistema

Intercanviar un procés

L'intercanvi en l'execució d'un procés per part del sistema operatiu, pot ser causat per les següents raons:

- **Interrupcions**

  La interrupcions es deuen a senyals externes al procés, la major part de vegades, fruit de canvis d'estat d'algun dispositiu. L'administració de les interrupcions ordinàries es transmet a l'administrador d'interrupcions, el qual implementa les tasques bàsiques de gestió i posteriorment salta a la rutina del sistema que reacciona a la interrupció succeïda.

  Un exemple podria ser una interrupció d'un dispositiu d'entrada i sortida, que en el cas que algun procés estigués bloquejat esperant la senyal del dispositiu, el sistema operatiu hauria de transferir el procés de la cua de bloquejats a la cua de preparats

- **Excepcions**

  Les excepcions es deuen a ïnterrupcionsgenerades pel propi sistema davant situacions anormals en l'execució d'algun procés, accés il·legal a fitxers, etc. El sistema operatiu és l'encarregat de discernir la importància de l'error i, en funció d'això, decidir si finalitzar l'execució del procés, executar alguna rutina especial de gestió d'errors i restaurar el procés o ignorar la situació i continuar amb l'execució

- **Crides de sistema**

  Les crides als sistema s'esdevenen en situacions en les quals els procés en execució precisa d'alguna característica (accés a dispositius, etc.) a la qual només pot accedir el sistema operatiu. Aquesta crida es transmet al sistema operatiu, que executa la rutina associada

Intercanviar l'entorn d'un procés

En les arquitectures dels sistemes de control, i específicament la dels microcontroladors que constitueixen el nucli dur dels sistemes encastats, s'introdueix un cicle d'interrupció en el cicle d'instrucció. En aquest cicle es comprova si hi ha hagut alguna interrupció, condicionant la seqüència d'execució posterior. En cas que no hi hagi cap interrupció pendent, el processador continua amb la seva execució normal. En cas que hi hagi alguna interrupció pendent:

1. Salva l'entorn del procés. El comptador de programa, els registres del processadors i la informació de la pila
2. Assigna l'adreça de la rutina d'interrupció al comptador de programa

En cas que a posteriori de l'execució de la interrupció hi ha un canvi de procés en execució, s'executa la seqüència de tasques que defineixen un intercanvi de procés.

Intercanviar un procés

En l'intercanvi d'un procés, la seqüència de tasques a realitzar amplia les tasques bàsiques que es realitzen en un intercanvi d'entorn d'un procés fruit d'una interrupció, vegeu [10]. L'intercanvi de procés, implica modificar l'estat d'un procés i, en conseqüència, transferir-lo d'una cua a una altra. Segueix la següent seqüència de tasques:

1. Salvar l'entorn del processador (registres, comptador de programa, etc.)
2. Actualitzar l'estat del bloc de control del procés passant d'executant al següent estat
3. Transferir el bloc de control de procés a la cua del nou estat
4. Seleccionar el nou procés a executar
5. Actualitzar l'estat del bloc de control del nou procés passant de l'estat preparat a executant
6. Actualitzar les estructures de memòria del procés
7. Restaurar l'entorn del processador salvat anteriorment en l'últim intercanvi o inicialització del procés

3.3 Temps real

3.3.1 Descripció

En els sistemes en temps real, la correcció en l'execució de les tasques no depèn únicament dels resultats lògics, sinó també de les limitacions temporals. És un sistema que ha de satisfer les estrictes restriccions del temps de resposta que poden induir a riscs severs en l'entorn que controlen. Podeu trobar més informació a l'obra d'anàlisi de Laplante[16], la d'anàlisi i llenguatges de programació de Burns [3] i l'obra de Audsley [1]. La tasca de dissenyar la seqüència d'execució per assolir la totalitat de les restriccions de les diferents tasques, la realitza el planificador de tasques.

El temps real s'acostuma a caracteritzar per la utilització en sistemes multitasca degut a les pròpies restriccions temporals. Les tasques es classifiquen en:

- **Periòdiques** Una tasca periòdica es reprodueix indefinidament en intervals constants
- **Aperiòdiques** Una tasca aperiòdica es produeix a manera aleatòria fruit d'alguna d'alguna canvi de l'entorn del sistema de control

En funció d'aquestes restriccions, s'identifiquen dos tipus de models de temps real, vegeu [2]:

- **Soft Real Time** En aquest model, el no compliment de les restriccions temporals únicament degrada el comportament del sistema
- **Hard Real Time** En aquest model, el no compliment de les restriccions temporals poden provocar la fallida del sistema

Els sistemes tradicionals de planificació de tasques estan basats en el repartiment equitatiu dels recursos del sistema. Alguns dels models de planificació que implementen aquest concepte són els següents:

- **FCFS** La primera tasca que arriba és la primera que s'executa. La mitjana del temps d'espera és excessivament alt tot i la facilitat de la seva implementació
- **SJF** S'executa la tasca que precisa de menys temps d'execució. La complexitat d'aquest model recau en la dificultat de predir el temps d'execució de la tasca
- **Round Robin** S'assigna un temps d'execució límit per qualsevol tasca, de tal manera que en cada interval de temps una nova tasca accedeix a l'estat d'execució.

La falta de determinació dels models anteriors de planificació entren en confrontació amb la necessitat de satisfer les estrictes restriccions d'un sistema en temps real.

3.3.2 Planificació

Els models de planificació en temps real busquen un comportament determinista del sistema i assolir les restriccions temporals del sistema, vegeu l'obra de TimeSys [4] i Frazer [5]. Els models es poden classificar en dos grups

- Sistemes Off-line
- Sistemes On-line

Sistemes Off-line

La planificació de l'accés als recursos del sistema es realitza abans de la seva entrada en funcionament. En aquests models, es disposa de la totalitat de les característiques del sistema com també de

les restriccions de les tasques. Aquest fet comporta que el resultat final de la planificació compleixi estrictament totes les restriccions i el funcionament sigui completament determinista. La complexitat d'aquests models, recau en la dificultat de descriure la totalitat del sistema i la dificultat de modificació del model en cas de modificacions en l'entorn i, en conseqüència, a les característiques utilitzades pel planificador

Tot i que per definició no planifica tasques aperiòdiques, es pot intentar convertir una tasca aperiòdica a periòdica i així planificar segons aquest model.

La planificació de l'execució de tasques també està condicionada per les relacions que pugui haverhi entre les diferents tasques del sistema. Situacions en les que comparteixen diferents recursos, cal definir quines part de la seqüència d'execució d'una tasca es poden interrompre per passar l'execució a una altra tasca, i especificar-ho com a requisit específic en la planificació.

Un altre factor que el fa interessant, és la utilització del mètode que precisi de menys canvis de context i, per tant, precisi de menys recursos i disminueixi menys el temps d'execució de tasques al sistema.

Sistemes On-line

La planificació de l'accés als recursos del sistema es realitza durant el funcionament d'aquest, a mida que es van generant nous processos, la planificació absoluta del sistema es va modificant. El desconeixement de les característiques dels processos que es generaran en un futur, fan que sigui difícil realitzar una correcte planificació del sistema i, per tant, sigui complex complir amb els requisits de les limitacions. La necessitat d'adaptar la planificació en cada instant de l'execució del sistema, com també el continu intercanvi de tasques, implica utilitzar gran quantitat de recursos únicament per la coordinació

Per contra, s'adapta molt més a les característiques dinàmiques del sistema com també de la gestió de les tasques aperiòdiques.

Algorismes de planificació

- **Algorismes de prioritat estàtica no preemptius**

  En alguns entorns on s'utilitzen sistemes en temps real, l'ordre d'execució i el fet que que una tasca no es pugui executar fins finalitzada una altra (interrelació entre tasques), són característiques conegudes. Aquests sistemes es poden planificar de manera no-preemptiva i per tan simplificar la planificació i la quantitat de recursos necessaris per la coordinació.

- **Algorismes de prioritat estàtica preemptius**

  L'algorisme es basa en l'assignació de prioritats, de forma estàtica o dinàmica, a cada una de les tasques en funció de les restriccions de temps. La tasca en execució serà la tasca amb la

prioritat més alta: si s'està executant una tasca de baixa prioritat, i arriba una tasca de major prioritat, la primera deixa d'executar-se i la segona accedeix a l'ús del processador. Si les prioritats es poden assignar en funció de les limitacions de temps i les interrelacions entre tasques, es pot assegurar una planificació que compleixi les restriccions de temps de la tasca.

- **Algorismes de basats en la planificació dinàmica**

  Contràriament als algorismes plantejats fins ara, la planificació dinàmica permet la modificació de la seqüència de d'execució en temps d'execució. L'acceptació d'execució d'una nova tasca només està condicionada pel fet de poder complir les seves restriccions temporals. En cas d'arribar una nova tasca, i abans d'executar-se, s'intenta trobar una planificació que compleixi els requisits de la nova i les que han arribat anteriorment. Si l'intent falla i es disposa de suficient temps abans d'assolir el temps límit d'execució, es poden intentar prendre decisions alternatives, vegeu [21].

- **Algorismes de millor rendiment dinàmic**

  Igual que en el cas anterior, la planificació del millor rendiment permet la modificació de la seqüència en temps d'execució. S'assigna una prioritat per a cada tasca en funció de les característiques temporals d'aquesta, i el planificador genera la seqüència d'execució en funció de les prioritats. Aquest algorisme s'utilitza en sistemes en el que el compliment dels requisits temporals d'execució no resulten fatals pel sistema i, especialment en situacions de sobrecarregament. En qualsevol moment de l'execució, una tasca pot ser interrompuda i intercanviada per a una altra, vegeu [21]

Per més informació vegeu els diferents models proposats per Liu [18], Ramamritham [27] i Ripoll [20]

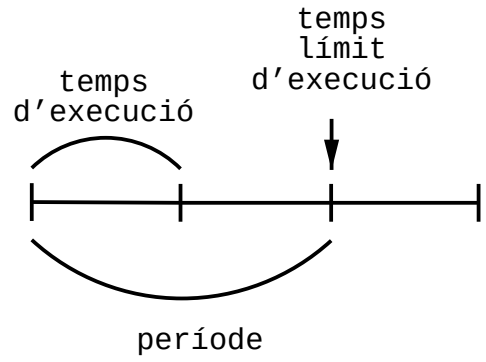
3.3.3 Rate Montonic Analysis

La teoria de l'algorisme de planificació Rate Montonic, de tipus estàtic preemptiu, és desenvolupat per Lui i Layland el 1973 amb l'objectiu d'analitzar si un conjunt de tasques, amb les seves respectives restriccions temporals, compleixen les restriccions de límits temporals d'execució. És a dir, si el conjunt de tasques és planificable i, per tant, cada una de les tasques s'executa abans de finalitzar el seu temps límit. Per més informació vegeu els estudis sobre el RMA realitzar per Sha[23], Lehoczky [17] i Heidmann [6]

En l'enunciat inicial de la teoria, hi ha un conjunt de restriccions que limitaven de forma important la seva implantació en sistemes de temps real

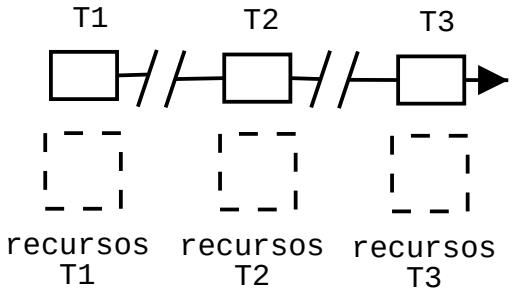
- **Planificació de tasques periòdiques**

  Les tasques que pot planificar l'algorisme són únicament periòdiques. El límit temporal d'execució de la tasca serà el mateix que el període.



- **Independència entre tasques**

  Les tasques han de ser absolutament independents unes de les altres i, en conseqüència, no hi poden bloquejos degut a recursos compartits o seqüències d'execució preestablertes.



- **Temps d'execució constant**

  Les tasques han de tenir un temps d'execució constant

- **Privilegi tasques aperiòdiques**

  Les tasques aperiòdiques desplacen en execució les periòdiques, i no tenen restriccions temporals d'execució

L'assignació de les prioritats en l'agorisme RMA, es basa en el període de les tasques, i més específicament, entre la relació de períodes entre les diferents tasques. Les tasques que tenen el període més petit obtenen la prioritat més gran.

L'anunciat del RMA, defineix les característiques que ha de tenir un conjunt de tasques per tal que s'assoleixin tots els temps d'execució, i es centra en el compliment de l'expressió

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \left(2^{\frac{1}{k}} - 1\right) \tag{3.1}$$

on C_i és el temps d'execució i T_i és el període de la tasca i

Solucions a les restriccions del RMA

La limitació de planificar tasques periòdiques es pot resoldre a partir d'algun dels següents plantejaments:

- **Tasca periòdica enquestant tasques aperiòdiques**

- **Protocol d'intercanvi de prioritats (interchange priority protocol)**

  Tasca periòdica creada per processar tasques aperiòdiques. Si no hi ha cap sol·licitud per part d'una tasca aperiòdica al principi del seu període, permet l'execució d'alguna tasca d'inferior prioritat fins que aquesta acabi o es generi alguna tasca aperiòdica. Si la tasca a la qual s'ha cedit l'accés finalitza la seva execució, es dóna accés a una nova tasca d'inferior prioritat mantenint les restriccions. Si durant l'execució d'alguna tasca d'inferior prioritat arriba una tasca aperiòdica, aquesta s'executa amb els privilegis de la tasca interrompuda, que són més baixos que els que tenia al principi del període de la tasca que les gestiona.

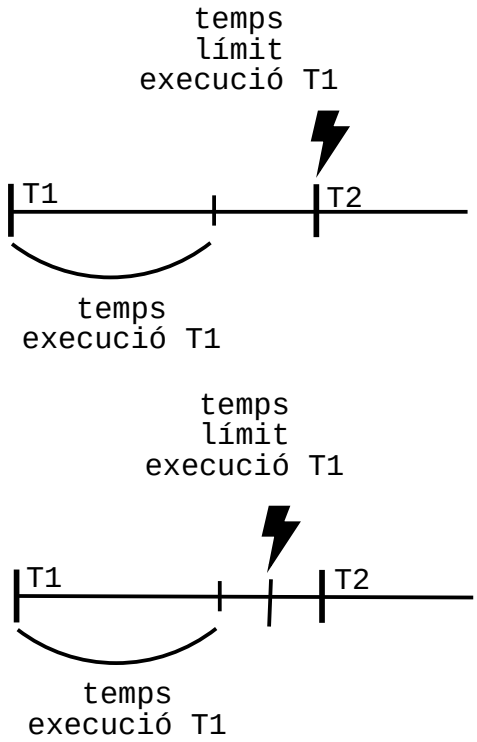
- **Servidor diferit (deferrable server)**

  Tasca periòdica creada per processar tasques aperiòdiques. Si en cada període de la tasca no hi ha cap tasca aperiòdica esperant, el servidor suspèn la seva execució esperant que n'arribi alguna. Quan arriba la tasca aperiòdica, en cas de no haver utilitzat el temps assignat del servidor, atura la tasca executant-se i assumeix el control de l'execució. La reassignació del temps del servidor es realitza periòdicament.

- **Protocol de servidor esporàdic (sporadic server protocol)**

  Tasca periòdica creada per processar tasques aperiòdiques. Si no hi ha cap sol·licitud per part d'una tasca aperiòdica al principi del seu període, permet l'execució d'alguna tasca d'inferior prioritat mantenint el temps d'alta prioritat assignat al servidor. Si durant l'execució d'alguna tasca d'inferior prioritat arriba una tasca aperiòdica, aquesta utilitza el temps assignat al servidor. La reassignació del temps del servidor es realitza un període sencer després de ser consumit. Per més informació vegeu [29].

En el cas de les limitacions de temps d'execució limitades per la l'intercanvi amb una a segona tasca, es poden resoldre implementen un gestor de cues de tasques, controlador de les tasques en execució, que inicia l'execució de les tasques independentment dels límits d'execució de les anteriors.



La independència entre tasques que planteja el tercer punt, limita extremadament la utilització en processos reals en els que els tasques comparteixen recursos utilitzant canals de comunicació entre tasques, semàfors, etc. Per resoldre les situacions que hereten de la utilització d'aquestes recursos, s'implementen diferents protocols, vegeu [24]. Per resoldre la utilització de semàfors, s'utilitzen el següents protocols:

- **Resoldre la prioritat inversa. Protocol de prioritat heretada**

  Si una tasca bloqueja una tasca de prioritat superior, la tasca d'inferior prioritat hereta la prioritat de la primera durant l'execució de la secció crítica. En la planificació de tasques cal tenir presents aquests bloquejos, que s'exposaran en una apartat posterior

- **Deadlocking. Protocol amb sostre de prioritat (ceiling protocol)**

  Cada semàfor t'he assignat una prioritat sostre definida per la prioritat més alta de les tasques que poden precisar la seva coordinació. Si una tasca vol entrar a una zona crítica coordinada per un semàfor, la seva prioritat ha de ser superior a l'assignada com a sostre del semàfor, en cas contrari queda bloquejada. Si la tasca bloqueja tasques de prioritat superior, hereta la seva prioritat durant l'accés a la secció crítica. En la planificació de tasques cal tenir presents aquests bloquejos, que s'exposaran en una apartat posterior

La impossibilitat d'interrupció de l'execució d'una tasca i per tant la constància en el temps d'execució, es pot variar tenint present els condicionant dels nous elements introduïts. En el cas de necessitar la introducció de noves tasques cal tenir presentes les següents qüestions:

1. La modificació de la prioritat sostre d'un semàfor no-bloquejat es pot realitzar directament

2. L'augment de la prioritat sostre d'un semàfor bloquejat es pot realitzar quan el semàfor s'ha desbloquejat

3. La disminució de la prioritat sostre d'un semàfor bloquejat es pot realitzar quan la prioritat de les tasques bloquejades és inferior a la nova

4. La introducció d'una nova tasca està condicionada a la disponibilitat de recursos

Anàlisi estricte per càrrega de treball

La planificació de tasques es realitza per un interval anomenat LCM (least common multiple) que defineix l'interval en el que s'anirà reproduint la planificació desenvolupada. La part esquerra de l'ecuació bàsica de l'anàlisi, suma les fraccions de temps execució/període per a cada una de les tasques obtenint l'utilització total del processador. L'algorisme RMA és suficient per determinar si conjunt de tasques són planificables, però no és necessari. L'anàlisi estricte és necessari i suficient per tal de trobar la planificació per resoldre les planificacions que podrien fallar amb l'anàlisi bàsic. Per més informació vegeu [9]

En l'anàlisi estricte, els temps de càrrega del processador es compara amb el límit temporal d'execució de la tasca. Per realitzar el càlcul cal primerament ordenar les tasques de més a menys prioritat, permetent veure les relacions d'utilització entre tasques.

$$W_i(t) = \sum_{j=1}^{n} C_i \left\lceil \frac{t}{p_j} \right\rceil \tag{3.2}$$

on C_j és el temps d'execució, T_j és el període de la tasca j i t és l'interval de temps del càlcul

Per comprovar la viabilitat de la planificació s'ha de complir que

$$W_i(t) \leq d_i \tag{3.3}$$

on W_i és la càrrega temporal del processador i d_i és el límit temporal d'execució

3.3.4 Protocols de gestió de recursos compartits

La limitació que incorpora el RMA respecte a la independència de les tasques, i la necessitat de compartir recursos a partir d'algun dels mètodes següent

- Comunicació entre processos utilitzant semàfors, cues, missatges
- Regions de memòria compartida
- Recursos compartits
- Comunicació via sockets
- Invocació de mètodes remots

Per resoldre els problemes de planificació que s'esdevenen de la utilització dels elements de la llista anterior, s'introdueixen els següents protocols on cal tenir presents els següents conceptes

C_i Temps d'execució de la tasca i

P_i Prioritat de la tasca i

T_i Període de la tasca i

S_i Semàfor binari i coordinador una zona crítica d'un recurs

$Z_{i,j}$ Zona crítica d'una tasca j coordinada per una semàfor i

$D_{i,j}$ Duració de l'execució de la zona crítica

$B^*_{i,j}$ Secció crítica més gran de la tasca J_j que pot bloquejar la tasca J_i

B_i* Conjunt de seccions crítiques més grans que poden bloquejar la tasca J_i $\left(B^*_i = U_j \geq_i B^*_{i,j}\right)$

Protocol de prioritat heretada

Les tasques es planifiquen en funció de les seves prioritats actives, i en el cas que estiguin en un mateix nivell de prioritat, utilitzant el protocol FIFO(first input first output).

Quan una tasca J_i intenta accedir a una zona crítica $Z_{i,j}$ i el recurs $R_{i,j}$ (S_j) ja està controlat per una tasca d'inferior prioritat, la tasca J_i es bloqueja. Es diu que la la tasca J_i està bloquejada per la tasca que controla el recurs, si no és el cas, la tasca J_i accedeix a la zona crítica $Z_{i,j}$.

Quan a una tasca J_i està bloquejada per un semàfor, transmet la seva prioritat a la tasca que el controla. Aquesta tasca, s'executa durant la resta de la secció crítica amb la prioritat de la bloquejada. La tasca que bloqueja hereta la prioritat de la tasca bloquejada amb la prioritat més gran.

Quan una tasca que controla un recurs surt de la secció crítica, deixa de controlar el semàfor; si hi ha alguna tasca bloquejada pel semàfor de prioritat superior, passa a executar-se. A la tasca sortint s'assignarà la prioritat més alta de les tasques bloquejades.

El bloqueig es pot deure principalment a dues raons:

- Bloqueig directe
- Bloqueig indirecte

Si hi ha n tasques d'inferior prioritat que poden bloquejar una tasca J_i, llavors J_i pot estar bloquejada durant n zones crítiques, independentment del número de semàfors que utilitzi J_i. Si hi ha m semàfors diferents que poden bloquejar la tasca J_i, llavors J_i pot estar bloquejada durant m zones crítiques, una per cada m semàfor.

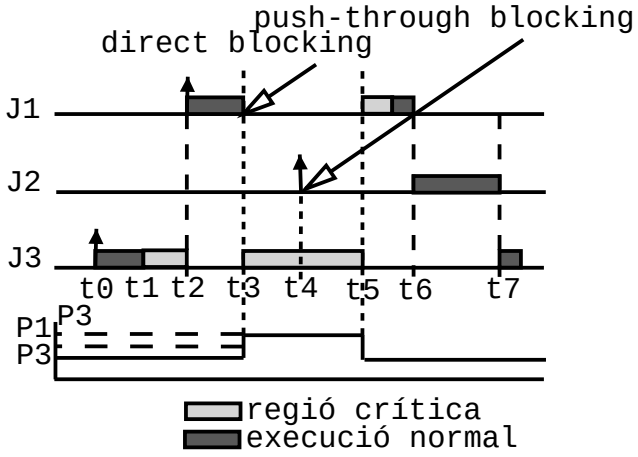
En el protocol de prioritat heretada, una tasca J pot estar bloquejada durant $min(n, m)$

$$\forall i, 1 \leq i \leq n \sum_{k=1}^{1} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq i \left(2^{\frac{1}{i}} - 1\right) \tag{3.4}$$

Per cada tasca J_j amb una prioritat més petita que P_i, les zones crítiques que poden bloquejar la tasca J_i s'identifiquen com $B_{i,j}$. Per cada semàfor S_k, tots les zones crítiques que coordinada i que poden bloquejar la tasca J_i s'identifiquen com a $\gamma_{i,k}$.

La suma de la duració de les seccions crítiques més llargues en cada $B_{i,j}$ per cada tasca J_j amb una prioritat més petita que P_im, s'identifica com a B_i^l. La suma de la duració de les zones crítiques més llargues en cada $\gamma_{i,k}$ per qualsevol semàfor S_k, s'identifica com B_i^s

El temps de bloqueig B_i s'obté del mínim entre B_i^l i B_i^s



Protocol amb sostre de prioritat

El protocol amb sostre de prioritat s'utilitza per prevenir bloqueig transitiu i deadlocking. Si una tasca J_k s'expulsa de l'estat d'execució en una regió crítica Z_a per una tasca J_i que entra a una regió crítica J_b, llavors la tasca J_k no pot heretar una prioritat superior o igual a la tasca J_i fins que la tasca J_i no finalitzi la seva execució. Una tasca J_i pot estar bloquejat fins la duració d'una regió crítica, i una regió crítica $Z_{j,k}$ pot bloquejar una tasca J_i només si $P_j < P_i$ i $C\left(S_k\right) \geq P_i$.

Cada semàfor S_k té assignada una prioritat sostre $C\left(S_k\right)$, calculada prèviament a l'execució, igual a la prioritat de la tasca amb més prioritat de la pot bloquejar.

Considerem que la tasca J_i és la tasca amb la prioritat més alta de totes les disponibles, S^* el semàfor amb la prioritat de sostre més alta entre tot els semàfors controlats per tasques diferents a J_i i $C\left(S^*\right)$ la prioritat sostre.

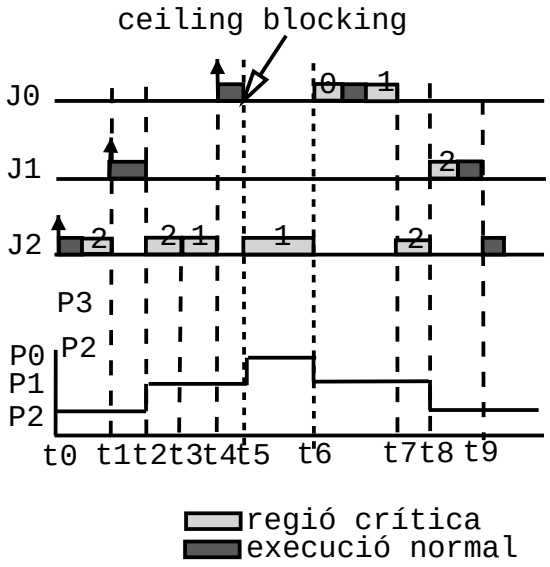
Per entrar a a una regió crítica controlada pel semàfor S_k, la tasca J_i ha de tenir una prioritat superior a $C\left(S^*\right)$. Si la $P_i \leq C\left(S^*\right)$, es denega el control de S_k i la tasca J_i queda bloquejada pel semàfor S^* per la tasca que controla el semàfor S^*.

Quan una tasca J_i queda bloquejada en un semàfor, transmet la seva prioritat a la tasca J_k, que controla em semàfor. La tasca J_k s'executa durant la resta de regió crítica amb la prioritat de la tasca J_i. Es diu que la tasca J_k hereta la prioritat de J_i. En general, una tasca hereta la prioritat més alta de les tasques que bloqueja.

Quan la tasca J_k surt d'una regió crítica, deixa de controlar el semàfor i permet l'execució de la tasca de més prioritat bloquejada pel semàfor. Un cop abandona la regió crítica, la prioritat de la tasca J_k pot ser: si no hi ha cap altra tasca bloquejada per J_k, la seva prioritat és la nominal, en cas contrari, la seva prioritat és la de la tasca amb més prioritat bloquejada per J_k.

L'herència de prioritat és transitiva, i per tant si una tasca J_3 bloqueja una tasca J_2, i la tasca J_2

bloqueja la tasca J_1, llavors la tasca J_3 hereta la prioritat de J_1 via J_2.



Anàlisi de planificació

Tenint present les limitacions exposades fins aquest punt, l'anàlisi bàsic de l'algorisme RMA s'obté de la següent equació:

$$\forall n, 1 \leq n \leq m \quad \frac{C_1}{T_1} + ... + \frac{C_n}{T_n} \leq n\left(2^{\frac{1}{i}} - 1\right) \tag{3.5}$$

Equació 3.5

$$\forall i, 1 \leq i \leq n \quad min_{(k,l)\in R_i} \sum_{j=i}^{i} C_j \frac{1}{lT_k} \left[\frac{lT_k}{T_j}\right] = min_{(k,l)\in R_i} \sum_{j=i}^{i} U_j \frac{T_j}{lT_k} \left[\frac{lT_k}{T_j}\right] \quad \leq \quad 1 \tag{3.6}$$

C_j Temps d'execució

T_j Període

U_j Utilització de la tasca T_j

$R_i = \left\{(k,l)\,|1 \leq k \leq i, l = 1, ..., \left\lceil \frac{T_i}{T_k} \right\rceil \right\}$

El bloqueig de qualsevol tasca J_i pot ser directe, push-through, però mai excedirà a B_i.

$$\forall i, 1 \leq n \leq n \quad \frac{C_1}{T_1} + ... + \frac{C_n}{T_n} + \frac{B_i}{T_i} \leq n\left(2^{\frac{1}{i}} - 1\right) \tag{3.7}$$

Suposem que per cada tasca J_i es compleix l'equació, per la qual cosa es compleix la primera equació, també es complirà en cas complirà si $n = i$ i es substitueix C_i per $C_i^* = (C_i + B_i)$. És a dir, en absència de qualsevol bloqueig, qualsevol tasca J_i complirà els seus límits d'execució encara que s'executi $(C_i + B_i)$ unitats de temps, i per tan si s'executa durant C_i unitats de temps, es pot retrassar un interval B_i i continuarà complint les limitacions.

$$\frac{C_1}{T_1} + ... + \frac{C_n}{T_n} + max\left(\frac{B_1}{T_1}, ..., \frac{B_{n-1}}{T_{n-1}}\right) \quad \leq \quad n\left(2^{\frac{1}{n}} - 1\right) \tag{3.8}$$

Un conjunt de n tasques periòdiques utilitzant el protocol amb sostre de prioritat es poden planificar utilitzant RMA si es compleix la següent condició:

$$\forall i, \quad 1 \leq i \leq n \qquad min_{(k,l)\in R_i} \quad \left[\sum_{j=1}^{i-1} U_j \frac{T_j}{lT_k}\left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{C_i}{lT_k} + \frac{B_i}{lT_k}\right] \quad \leq \quad 1 \tag{3.9}$$

C_j Temps d'execució

T_j Període

U_j Utilització de la tasca T_j $U_j = \frac{C_i}{T_i}$

B_i Màxim temps de bloqueig de la tasca

$R_i = \left\{(k,l)\,|1 \leq k \leq i, l = 1, ..., \left\lceil \frac{T_i}{T_k} \right\rceil \right\}$

Per més informació vegeu [9]

Capítol 4

Implementació proposada

4.1 Estructura de dades

La implementació del planificador de processos, utilitza un conjunt d'estructures que caracteritzen el sistema i aporten la informació necessària a les funcions que realitzaran les diferent subtasques del planificador.

Les diferents estructures segueixen el següent model

nom estructura
-atribut 1: tipus atribut 1 -atribut 2: tipus atribut 2 -...: ...

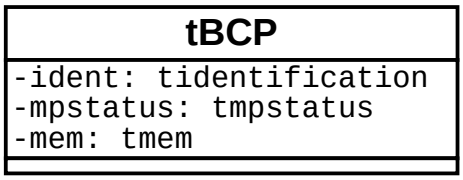
4.1.1 Procés

txPtab
-bcp: tBCP -sched: tsched -resources: tresources -xustack[XUSTACK_SIZE]: txustack -xuimem[XUIMEM_SIZE]: txuimem

Un procés està descrit pels següents elements:

- **BCP** Bloc de control de procés
- **Planificació** Característiques temporal del procés
- **Recursos** Característiques d'accés a recursos
- **Pila** Memòria de pila del procés
- **Memòria** Memòria local del procés

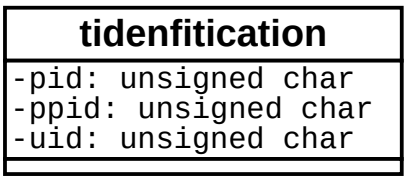
4.1.2 BCP



El bloc de control de procés és l'estructura de dades que descriu l'estat del procés en un instant determinat i conté el següent subapartats.

- Identificació
- Estat del processador
- Memòria Atributs que defineixen la posició de memòria en una instant determinat

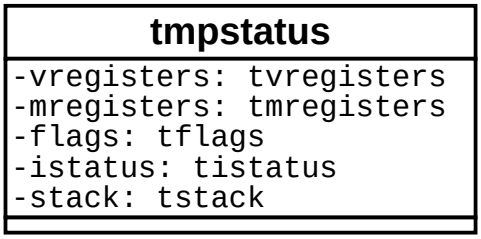
4.1.3 Identificació



L'estructura d'identificació conté els atributs que identifiquen cada procés

- **pid** Identificador únic del procés que en aquest cas és un número de 1..NUMTASKS_MAX.
- **ppid** Identificador únic que correspon a l'identificador del procés pare
- **uid** Identificador únic d'usuari al qual s'associa el procés, en aquest cas és un número de 1..NUMUSERS_MAX

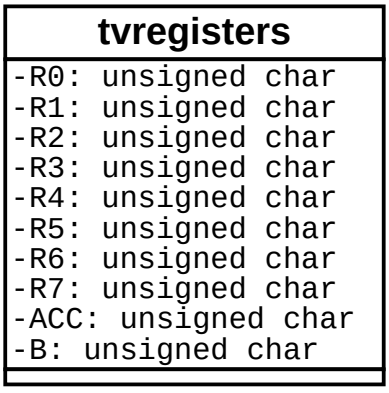
4.1.4 Estat del processador



L'estructura conté els atributs que defineixen l'estat del processador en un instant determinat i es descriu a través del següents subapartats:

- Registres visibles
- Registres no-visibles
- Banderes
- Estat interrupcions
- Estat pila

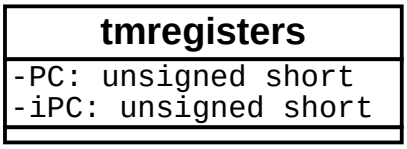
Registres accessibles



L'estructura de registres accessibles conté els registres visibles no referents a interrupcions, ni memòria del processador, en un instant determinat.

- **R0..R7** Registres universals de memòria interna situats en el banc de registres seleccionat
- **ACC** Registre amb funció especial representant a l'acumulador
- **B** Registre amb funció especial representant al registre B

Registres no-accessibles



L'estructura de registres no-accessibles conté els registres als quals no es pot accedir a través del direccionament directe/indirecte

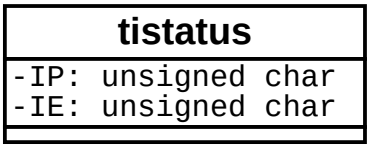
- **PC** Comptador de programa, registre que indica la posició de memòria de programa en cada instant del fil d'execució
- **iPC** Comptador de programa a inici de la tasca

Banderes

L'estructura de banderes conté la informació referent als indicadors d'estat

- **PSW** Registre amb funció especial que conté indicadors d'estat com poden ser els de carry, overflow, paritat o selecció de banc de registres universals

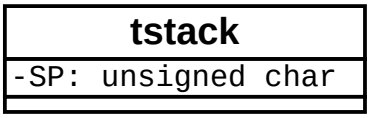
Estat d'interrupcions



L'estructura d'estat d'interrupcions conté informació referent als registres que gestionen les interrupcions

- **IP** Registre de nivell de prioritat de les interrupcions
- **IE** Registre d'habilitació de les interrupcions

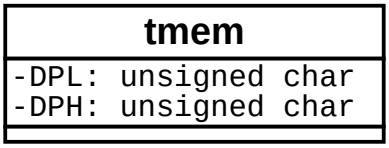
Estat de la pila



L'estructura d'estat de la pila conté informació referent a la situació de la pila en memòria interna

- **SP** Registre amb funció especial de punter a la situació de la pila en la memòria interna

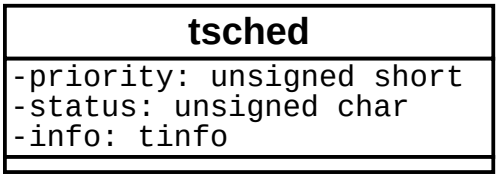
4.1.5 Memòria



L'estructura de memòria conté informació referent a la situació de la memòria de la tasca en memòria externa

- **DPH** Part alta del registre amb funció especial de punter a la zona de memòria externa de dades
- **DPL** Part baixa del registre amb funció especial de punter a la zona de memòria externa de dades

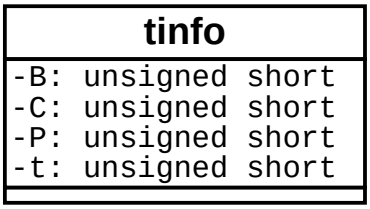
4.1.6 Planificació



L'estructura de planificació conté tots els atributs referents a qüestions temporals i de prioritat de l'execució d'una tasca

- **priority** Prioritat de la tasca que estarà directament relacionada amb el model de planificació escollit
- **status** Estat de la tasca
- **info** Atributs referents a qüestions temporals

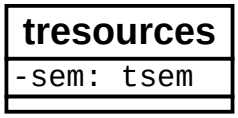
Informació temps



L'estructura d'informació de temps conté els atributs referents a qüestions de temps

- **B** Atribut de temps de bloqueig supeditat a la utilització de recursos compartits
- **C** Interval de temps de computació d'una tasca
- **P** Període d'execució d'una tasca
- **t** Comptador de temps d'execució d'una tasca

4.1.7 Recursos



L'estructura de recursos conté tota la informació referent a la gestió de recursos compartits i es descriu a través d'aquest subapartat

- Semàfors

Semàfors

tsem
-sem[NUMSEMS_MAX]: unsigned short -sem_used[NUMSEMS_MAX]: unsigned short -sem_flag[NUMSEMS_MAX]: tsemreg

L'estructura d'informació de semàfors conté tota la informació referent a la gestió de recursos compartits utilitzant semàfors. Cada un dels semàfors disposarà d'un dels següents atributs, amb la qual cosa es gestionaran un nombre NUMSEMS_MAX de semàfors

- **sem** Interval de temps d'accés a una recurs compartit necessari
- **sem_used** Interval de temps d'accés a una recurs compartit utilitzat. Sempre inferior a l'atribut anterior
- **sem_flag** Bandera utilitzada en la gestió d'accés al semàfor

4.1.8 Pila

pila
-txustack[XUSTACK_SIZE]: unsigned char

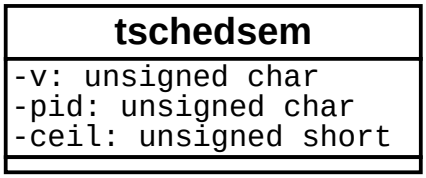
La zona de pila conté una imatge a memòria externa de la pila utilitzada durant els interval d'execució i situada a memòria interna

4.1.9 Memòria

memòria de dades
-txuimem[XUSTACK_SIZE]: unsigned char

La zona de memòria conté el conjunt de dades locals dels diferents processos situats permanentment en memòria externa

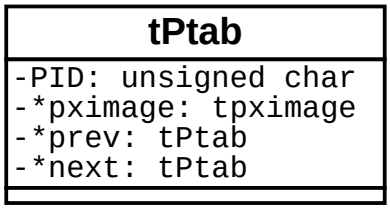
4.1.10 Semàfor



Un semàfor està descrit pel següent atributs

- **v** Atribut binari de restricció d'accés al propi semàfor
- **pid** PID del procés que controla el semàfor
- **ceil** Prioritat sostre del semàfor

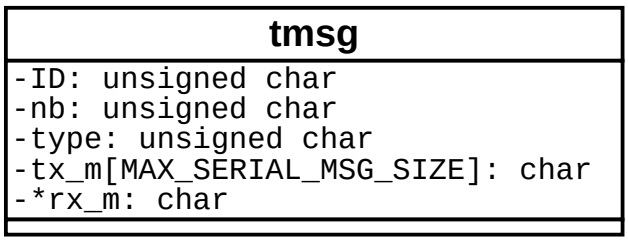
4.1.11 Taula interna de processos



Estructura que conté una taula interna de processos utilitzada en la gestió de cua i d'accés a la taula externa i descriptiva de processos

- **PID** Identificador de procés
- **pximage** Punter a l'estructura de descripció de la tasca amb pid = PID situada a memòria externa
- **next** Punter al següent element de la cua en que es troba en un instant determinat del seu interval d'execució
- **prev** Punter a l'element anterior de la cua en que es troba en un instant determinat del seu interval d'execució

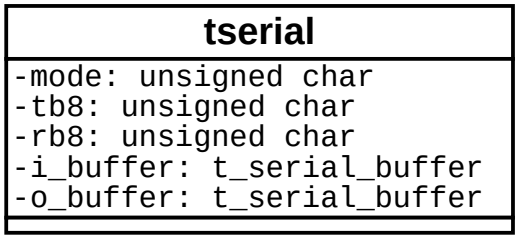
4.1.12 Missatge

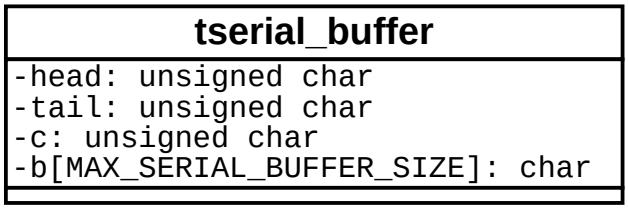


Estructura que descriu les característiques d'un missatge entre processos

- **ID** ID del procés orígen
- **nb** Número de bytes del missatge
- **type** Tipus de missatge
- **tx_m** Buffer d'emissió
- **rx_m** Punter a la zona de memòria de recepció

4.1.13 Port sèrie



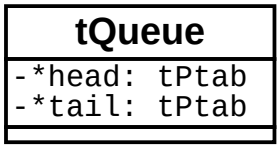


Estructura que descriu les característiques del port sèrie

- **mode** Mode de transmissió/recepció del port sèrie

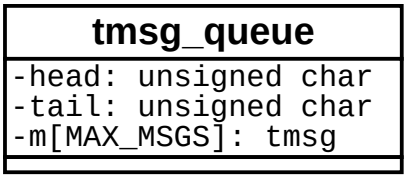
- **tb8** Novè bit de transmissió en el mode 2 o 3
- **rb8** Novè bit de recepció en el mode 2 o 3
- **i_buffer** Buffer de recepció
- **o_buffer** Buffer de transmissió

4.1.14 Cua



Estructura que descriu una cua que defineix un estat de la tasca durant la vida del sistema

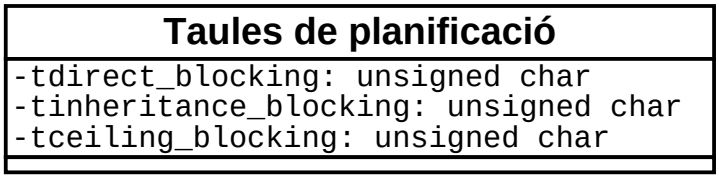
- **head** Punter al primer element de la cua
- **tail** Punter a l'últim element de la cua



Estructura que descriu una cua que gestionarà els missatges entre processos

- **head** Índex al primer element de la cua
- **tail** Índex a l'últim element de la cua
- **m** Llistat de missatges

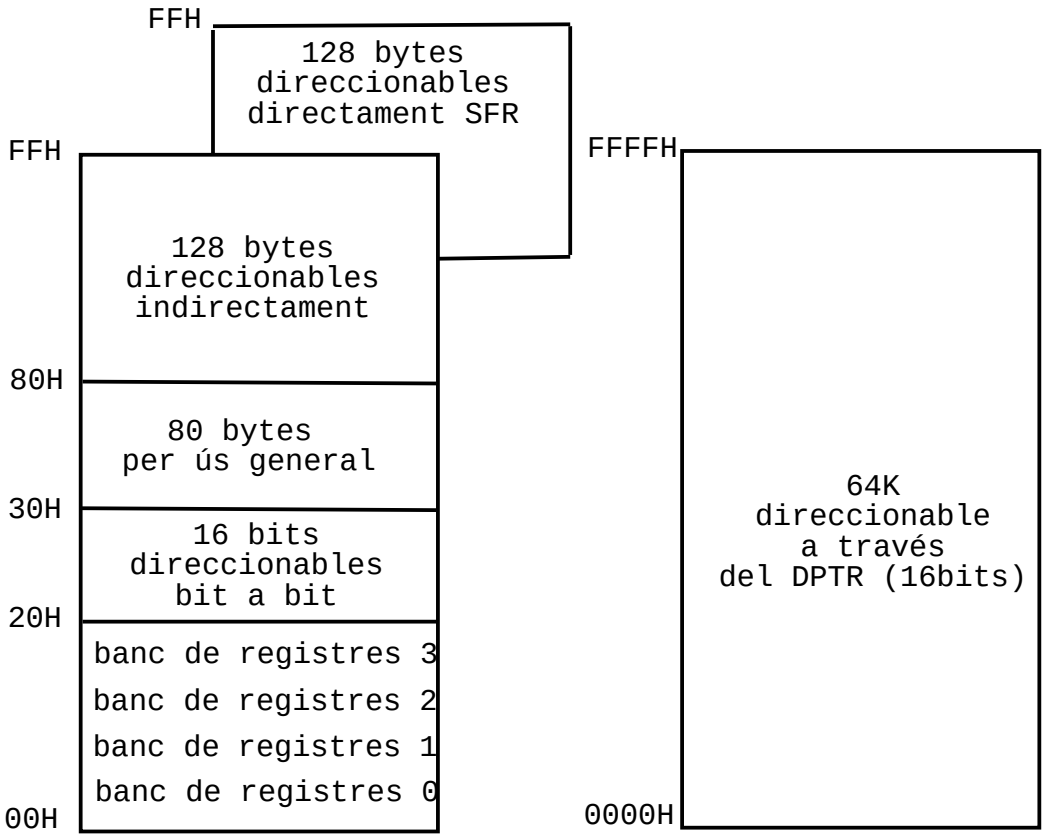
4.1.15 Taules de planificació



Les taules de planificació conten informació referent a l'estudi de planificabilitat del conjunt de tasques i les corresponents característiques que defineixen el sistema a estudiar

4.2 Projecció a memòria

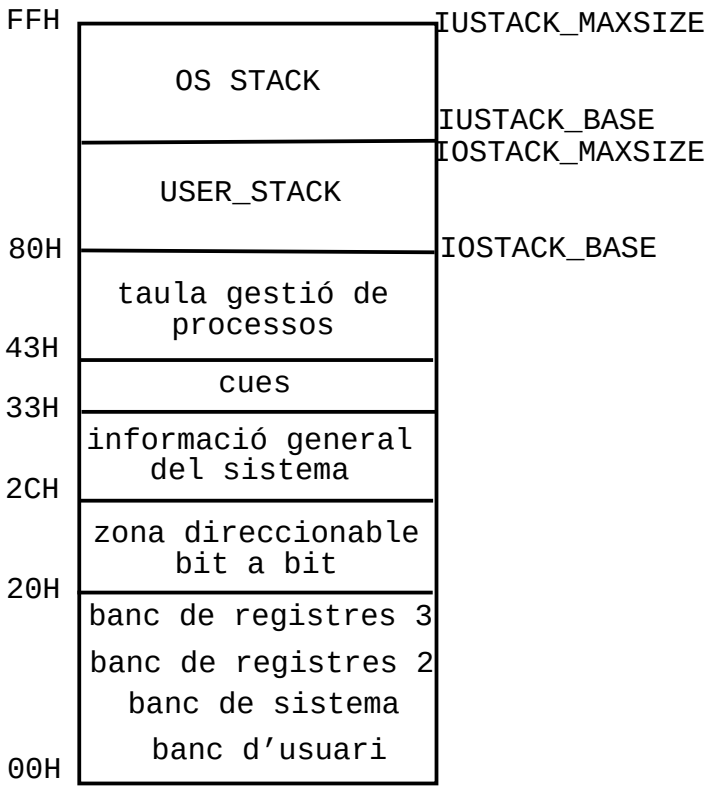
Tal i com s'exposa en l'apartat de descripció de l'arquitectura del microcontrolador 8051, aquest disposa de dos espais de memòria de dades: la memòria interna i la memòria externa. En el cas de la memòria interna, disposa d'un conjunt de registres universals distribuïts en diferents bancs utilitzables pel programador, d'altra banda, disposa d'un espai de lliure utilització direccionable indirectament i un conjunt de registre amb funció especial que representen diferents paràmetres del microcontrolador.



Ambdues memòries comparteixen dades del sistema de control i de les tasques de l'usuari

4.2.1 Memòria interna

En la memòria interna conviuen dades del sistema i de les tasques de l'usuari.



Bancs de registres universals

El banc de registres universals 0, conté els registres universals de la tasca que s'està executant en un moment determinat. El banc 1 conté els registres del sistema de gestió i, per tant, mantindrà els valors independentment de l'intercanvi de tasques que es pugui realitzar

80 bytes d'ús general

Aquest espai de la memòria interna es reserva per la taula interna de gestió de tasques del sistema, informació referent a les cues i informació general del sistema

Informació general del sistema

Informació general
del sistema

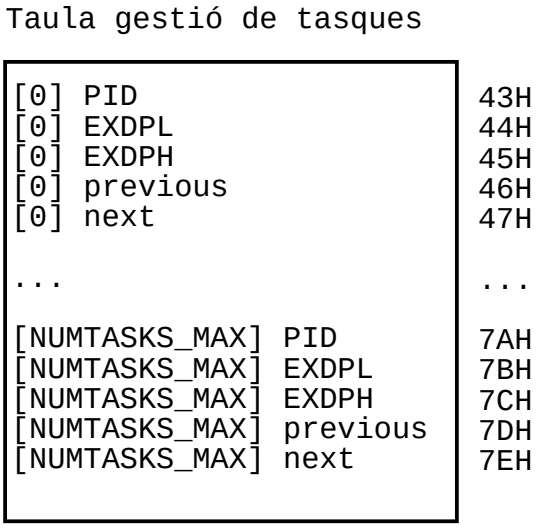
spsched idle	2CH
spsched	2EH
exec task	2FH
num sems	31H
num tasks	32H

Taula interna de cues representada per l'estructura tQueue

Cues

QBlk[0] head	33H
QBlk[0] tail	34H
QBlk[1] head	35H
QBlk[1] tail	36H
QBlk[2] head	37H
QBlk[2] tail	38H
QBlk[3] head	39H
QBlk[3] tail	3AH
QBlk[4] head	3BH
QBlk[4] tail	3CH
QDone head	3DH
QDone tail	3EH
QRdy head	3FH
QRdy tail	40H
QExec head	41H
QExec tail	42H

Taula interna de processos representada per l'estructura tPtab

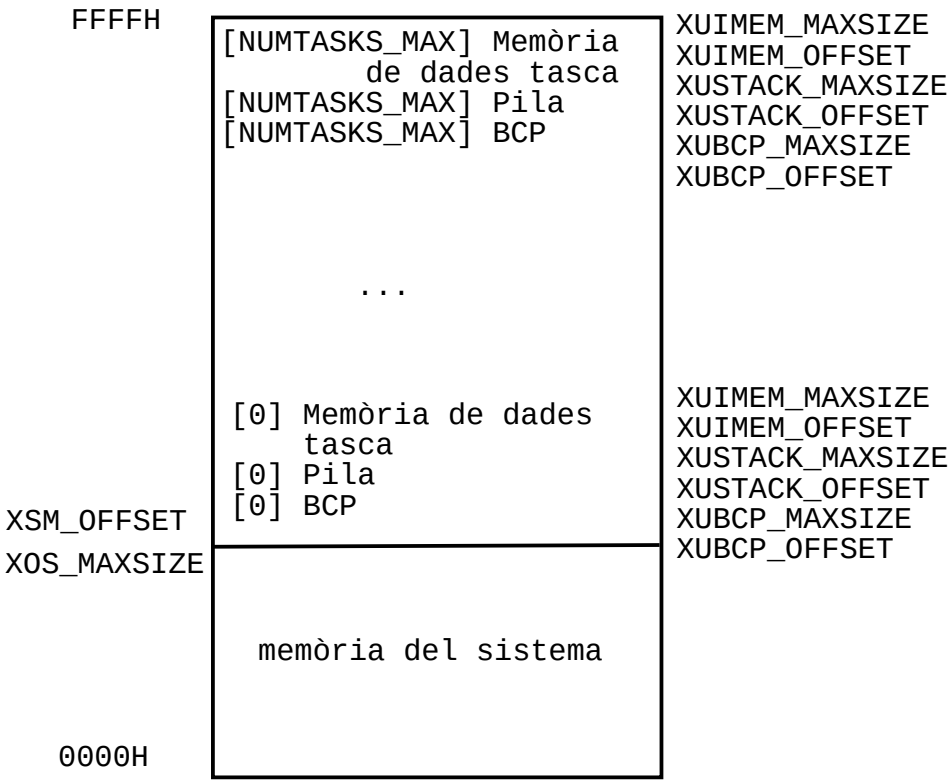


128 bytes direccionables indirectament

Aquest espai de la memòria externa té una zona reservada per la pila de la tasca que s'està executant en un moment determinat i una zona reservada pel sistema

4.2.2 Memòria externa

En la memòria externa conviuen dades del sistema i de les tasques de l'usuari.

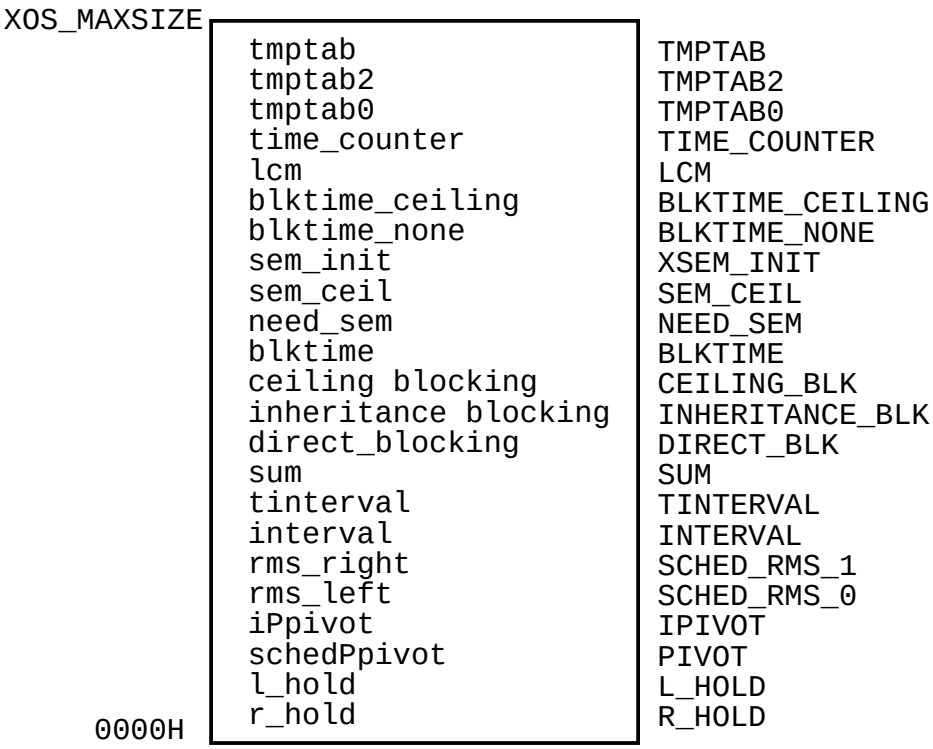


Zona de memòria de gestió del sistema

Aquest espai és una zona reservada a les taules i variables utilitzades internament pel sistema en la gestió del conjunt de processos i recursos

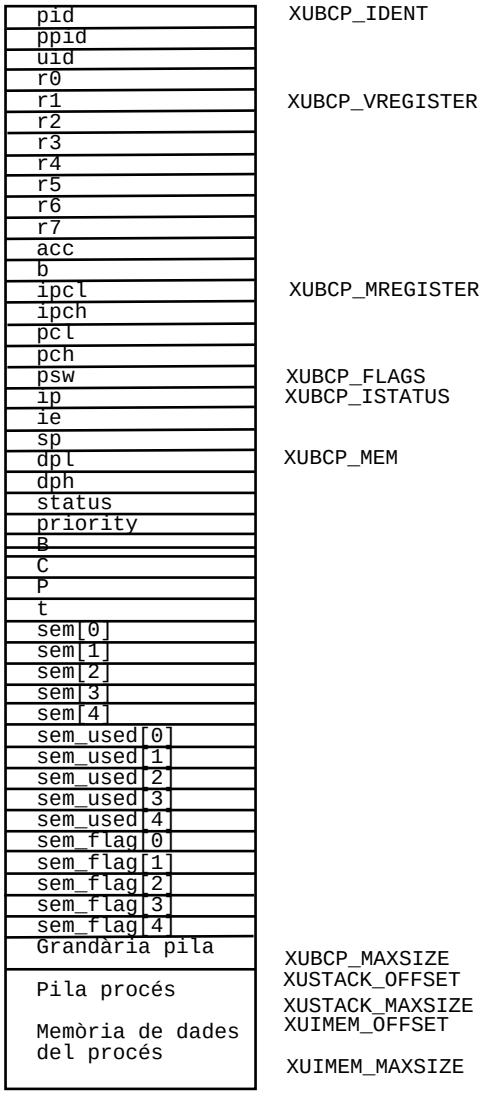
- Dades d'ordenació

- Dades temporals de càlcul de planificabilitat i planificació basat en l'algorisme RMS


XOS_MAXSIZE
tmptab
tmptab2
tmptab0
time_counter
lcm
blktime_ceiling
blktime_none
sem_init
sem_ceil
need_sem
blktime
ceiling blocking
inheritance blocking
direct_blocking
sum
tinterval
interval
rms_right
rms_left
iPpivot
schedPpivot
l_hold
r_hold
0000H
TMPTAB
TMPTAB2
TMPTAB0
TIME_COUNTER
LCM
BLKTIME_CEILING
BLKTIME_NONE
XSEM_INIT
SEM_CEIL
NEED_SEM
BLKTIME
CEILING_BLK
INHERITANCE_BLK
DIRECT_BLK
SUM
TINTERVAL
INTERVAL
SCHED_RMS_1
SCHED_RMS_0
IPIVOT
PIVOT
L_HOLD
R_HOLD

Zona de memòria de la taula de tasques

Aquest espai és una zona reservada a la taula de tasques. Tindrà la imatge del procés en execució i la zona de memòria de dades



4.3 Operacions amb les estructures

4.3.1 Procés

La gestió dels diferents processos que defineixen una sistema es realitza treballant amb el conjunt d'esctructures que caracteritzen una tasca. Les operacions que es realitzen són:

- Inicialització/finalització
- Ordenació
- Càlcul de bloquejos
- Càlcul planificabilitat
- Extracció LCM
- Càlcul RMS
- RMS (workload)
- Planificació
- Canvis de context
  - Salvar context
  - Restaurar context

Inicialització

La inicialització de qualsevol procés es realitza introduint un nou element a la taula de processos. El valor dels atributs del nou procés vindràs donats per les especificacions que pugui realitzar el programador. En la tasca d'inicialització, hi interven bàsicament dues funcions.

```
void init_task(unsigned char ntask,void *func)
```

Descripció	Inicialitza l'atribut del comptador de programa associat a l'execució d'una tasca
Precondició	El paràmetre *ntask* ha de ser de tipus *unsigned char* i expressar número de tasca a inicialitzar. El paràmetre *func* ha de ser el punter a la funció que representa una de les tasques a planificar.
Postcondició	Situa el punter a la funció que representa la tasca a l'atribut comptador de programa de l'estructura que descriu el context
Implementació	La primera part de la inicialització del procés recull el punter al comptador de programa de la funció *func* programada per l'usuari i l'introdueix a l'atribut *unsigned short PC* de l'estructura *tmregisters mregisters* que defineix el procés *ntask* de la taula de processos *txPtab xPtab*

```
void init_t_task(void)
```

Descripció	Inicialitza els atributs de la tasca definits pel programador del procés
Precondició	
Postcondició	Inicialitza els atributs d'identificació, requisits temporals, semàfor, pila i les referències a la taula interna de processos
Implementació	Inicialitza els atributs d'identificació de l'estructura *tidentification ident*, requisits temporals *tinfo info*, semàfors *tresources resource* i pila *tstack stak* de l'estructura que defineix el procés *unsigned char ntask* de la taula de processos *txPtab xPtab*. En aquesta mateixa funció s'inicialitza el nou procés en la taula interna de processos *tPtab Ptab* que apunta a l'estructura de la taula externa, i inicialitza els punters de gestió de cua *tPtab *prev* i *tPtab *next* apuntant a NULL

```
void end_t_task(void)
```

Descripció	Finalitza l'execució d'una tasca i cedeix l'execució a una de nova
Precondició	
Postcondició	Afegeix la tasca que finalitza la seva execució a la llista de tasques executades i cerca una nova tasca per accedir al processador
Implementació	Extreu la tasca de la cua de tasques en execució *tqueue QExec* i l'afegeix a la cua de tasques executades *tqueue QDone*. Cerca la següent tasca que accedeix a l'execució de la cua *tqueue QRdy* segons el criteri de planificació

A partir d'una taula de processos *txPtab xPtab* inicialitzada, és possible obtenir l'interval de temps mínim comú múltiple entre les diferents tasques.

```
char extract_lcm(void)
```

Descripció	Calcula l'interval de temps mínim comú múltiple entre les diferents tasques que defineix la planificació que es reproduirà indefinidament en l'interval de temps
Precondició	Les diferents tasques han d'haver inicialitzat les respectives estructures amb la definició temporal de les tasques
Postcondició	Actualitza el valor de la variable *unsigned short lcm* amb el valor del mcm calculat
Implementació	Utilitzant una implementació de l'obtenció del mínim comú múltiple i estudiant de l'atribut *unsigned short P* de l'estructura *tinfo info* dels processos de la taula *txPtab xPtab* s'obté el valor de l'interval mínim comú múltiple

Ordenació

L'ordenació dels processos es realitza a partir d'algun criteri d'ordenació basat en un o més atributs que defineixen un procés descrit a través de l'estructura *txPtab xPtab*.

```
char order_tasks(unsigned char left, unsigned char right)
```

Descripció	Calcula l'interval de temps mínim comú múltiple entre les diferents tasques que defineix la planificació que es reproduirà indefinidament en l'interval de temps
Precondició	Les diferents tasques han d'haver inicialitzat les respectives estructures amb la definició dels paràmetres de temporalitat *tinfo info*. Els paràmetres *left* i *right* han de ser de tipus *unsigned char* i defineixen els límits en l'ordenació
Postcondició	L'ordenació de les tasques modifica la projecció a memòria de les tasques, resituant les diferents estructures *txPtab Ptab* en una nova posició de memòria que ja està contemplat en el punter *txPtab *pximage* de la taula interna de processos *tPtab Ptab*
Implementació	A partir de la taula d'estructures de processos *txPtab xPtab* es realitza una ordenació basada en l'algorisme quick sort en el que l'element pivot és inicialment *unsigned char left = 0* i *unsigned char right = num_tasks*. L'atribut que s'utilitza en el criteri d'ordenació és el període de cada tasca *unsigned short P*, que és el que es tindrà en compte en el càlcul de planificabilitat i la planificació de les tasques. Aquestes s'ordenen de més a menys període que és inversament proporcional a la prioritat

Càlcul de bloquejos

```
char blk_times(void)
```

Descripció	A partir dels atributs dels recursos compartits especificats pel programador del procés, es calcula els bloquejos en funció dels criteris de bloqueig directe, bloqueig per herència de prioritat i bloqueig per sostre de prioritat
Precondició	Han d'estar inicialitzades l'estructura *tresources resources* i el nivell de prioritat especificat a *tinfo info* present a la taula de processos *txPtab Ptab*
Postcondició	
Implementació	En el cas del criteri de bloqueig directe, la funció recorre tots els processos ordenats buscant el temps de bloqueig màxim en la relació entre una tasca i la resta d'inferior prioritat. Per cada tasca *i*, guarda el valor màxim d'interval de temps d'accés a un recurs compartit *j* que comparteix amb una tasca de prioritat inferior *k*. A partir de les relacions de bloqueig que observa, inicialitza la taula *tdirect_blocking direct_blocking[NUMTASKS_MAX-1][NUMTASKS_MAX]*. En el cas del criteri de bloqueig per herència, la funció recorre tots els processos ordenats buscant situacions en les que una tasca *i* de prioritat inferior a una tasca *j* pugui bloquejar a aquesta última heretant la seva prioritat i bloquejant totes les tasques de prioritat superior a *i* però inferior a *j*. A partir de les relacions de bloqueig que observa, inicialitza la taula *tinheritance_blocking inheritance_blocking[NUMTASKS_MAX-1][NUMTASKS_MAX]*. El criteri de bloqueig per sostre de prioritat es basa en el criteri de bloqueig per herència alliberant les tasques que no precisin de semàfor en funció de la comparació entre la seva prioritat i la pioritat sostre.

Càlcul de planificabilitat

```
char rms_schedule_checking(void)
```

Descripció	Realitza el càlcul de planificabilitat segons les especificacions del mètode RMS.
Precondició	Les diferents tasques han d'haver inicialitzat les respectives estructures amb la definició dels paràmetres de temporalitat *tinfo info*
Postcondició	Retorna un *char* amb valor *SCHED_ERROR* en cas que les tasques definides no siguin planificables segons el mètode RMS i retorna *SCHED_OK* en cas que l'anàlisi sigui positiu.
Implementació	A partir dels atributs d'informació temporal especificats en l'estructura *tinfo info* presents en cada procés *txPtab xPtab* de la taula de processos, s'implementa l'agorisme basat en la funció $\forall i, 1 \leq i \leq n \sum_{k=1}^{1} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq i \left(2^{\frac{1}{i}} - 1\right)$ La variable *C* correspon a l'atribut *unsigned short C* de l'estructura *tinfo info*, la variable *T* a l'atribut *unsigned short T* de la mateixa estructura. En el cas de la variable *B* correspon *unsigned short B* obtingut en a través dels càlculs de bloqueig i també present a l'estructura *tinfo info* de cada procés de la taula *txPtab xPtab*.

```
char rms_schedule_eworkload(void)
```

Descripció	Realitza el càlcul de planificabilitat segons les especificacions del mètode de la càrrega de treball.
Precondició	Les diferents tasques han d'haver inicialitzat les respectives estructures amb la definició dels paràmetres de temporalitat *tinfo info*
Postcondició	Retorna un *char* amb valor *SCHED_ERROR* en cas que les tasques definides no siguin planificables segons el mètode RMS i retorna *SCHED_OK* en cas que l'anàlisi sigui positiu.
Implementació	A partir dels atributs d'informació temporal especificats en l'estructura *tinfo info* presents en cada procés *txPtab xPtab* de la taula de processos, s'implementa l'agorisme basat en la funció $W_i(t) = \sum_{j=1}^{n} C_i \left\lceil \frac{t}{p_j} \right\rceil$ Per comprovar la viabilitat de la planificació s'ha de complir que $W_i(t) \leq d_i$ Cada interval de temps de càrrega de cada tasca es compara amb les restriccions d'aquest. La càrrega de treball s'extreu de la primera funció aplicada a totes les tasques de prioritat igual o inferior.

Planificació

La planificació de les tasques es basa en la implementació de l'algorisme de planificació RMS en la funció

```
void oschedule(void)
```

Descripció	La funció de planificació selecciona la tasca que s'executarà en el pròxim canvi de context basant-se en la premissa proposada per l'algorisme RMS de seleccionar la tasca amb inferior període
Precondició	Les diferents tasques definides pel programador han d'estar inicialitzades i situades en alguna de les diferents cues *tQueue *queue* que expressen els possibles estats de les tasques
Postcondició	La tasca preparada per executar-se amb la prioritat més alta, es situa a la cua *tQueue *queue* de tasques executables *tQueue QExec*.
Implementació	La funció utilitza el conjunt d'operacions de tractament de cues per localitzar la tasca amb l'atribut prioritat *unsigned short priority* de l'estructura *tsched sched* més petita del conjunt de tasques present en la cua *QRdy*. Les operacions de gestió de cues que s'utilitzen són *struct tPtab *hpriority_task(struct tQueue *queue)* i *void queue_out(struct tPtab *pptab)*. A partir de l'execució d'aquesta funció, l'atribut *tPtab *tail* de l'estructura de la cua *tQueue QExec* apunta a l'estructura *tPtab Ptab* de la tasca a executar en el pròxim canvi de context.

Canvi de context en l'interrupció temporal de replanificació

L'intercanvi en l'execució de les tasques implica salvar l'estat del microcontrolador com també la pila i memòria de dades, l'anomenat context, de la tasca que deixar d'executar-se i recuperar el context de la nova tasca que passarà a executar-se.

```
void save_timer_context (struct tPtab *TTask,unsigned char sp)
```

Descripció	Salva l'estat del microcontrolador als respectius atributs de l'estructra *txPtab xPtab* de la taula de processos del sistema
Precondició	El paràmetre *TTask* ha de ser de un punter a la posició de memòria de la taula interna de processos de tipus *tPtab* que relaciona el procés a salvar amb la taula El paràmetre *sp* ha de ser de tipus *unsigned char* i indica la direcció de memòria interna de dades que correspon al punter de pila, registre *SP*

Postcondició	El context de la tasca en execució està salvat a l'estructura que defineix l'entorn en la taula de processos *txPtab xPtab*

Implementació

Salvar el context del procés que abandona l'execució implica plasmar aquest context sobre les estructures que defineixen el context d'un procés. L'estructura que defineix la tasca és *txPtab Ptab* i específicament en el l'estat del processador és *tBCP BCP*, en el cas de la informació de planificació *tsched sched*, en el cas dels recursos *tresources resources*, en el cas de la pila *txustack xustack* i en el cas de la memòria de dades locals *txuimem xuimem*.

La condició d'executar-se en la rutina de servei d'interrupció del timer0 implica que la informació que hi ha a pila en el moment de l'execució és diferent que en el cas que fos una rutina normal i corrent. En aquest cas, a l'executar-se la RSI compilador inclou el salvament a pila de dels registres ACC,B,DPH,DPL i PSW.

Salva a la pila de sistema els diferents registres que es poden modificar en el procés de salvar el context de la tasca i inicia el procés de salvar una imatge del context a *txPtab xPtab*. A partir de la direcció del procés a salvar de la taula interna, s'utilitza l'apuntador *txPtab *pximage* de l'estructura *tPtab Ptab* per situar el punter de memòria de dades externa a l'inici de l'estructura *txPtab xPtab* del procés a salvar de la taula de processos. A través de la definició dels diferents offsets de cada un dels subapartats que configuren la descripció del procés, va recorrent els diferents registres que en termes absoluts seria *&xPtab[num_task] + OFFSET_REGISTRE*.

Salva els registres universals que continuen accessibles al banc de registres 0, tot i que no a través del símbols, sinó directament a través de les posicions de memòria i utilitzant el punter a memòria externa a través de l'offset de l'inici de descripció del procés específic *XUBCP_VREGISTER* que correspon a l'estructura *tvregisters vregisters* del procés *txPtab xPtab*. Salva el comptador de programa que obté a partir de la lectura de l'adreça de retorn situada a la pila arran de la interrupció a l'offset *XUBCP_MREGISTER* que correspon a l'estructura *tmregisters mregisters* del procés *txPtab xPtab*. Salva el punter a memòria externa de dades obtingut de la pila de sistema i el situa a l'offset *XUBCP_MEM* del procés *txPtab xPtab*. Salva el registre d'estat (banderes) obtingut de la pila de sistema i el situa a l'offset *XUBCP_FLAGS* del procés *txPtab xPtab*. Salva els registres referents a interrupcions i el situa a l'offset *XUBCP_ISTATUS* del procés *txPtab xPtab*. Salva el registre d'informació de pila i el situa a l'offset *XUBCP_STACK* del procés *txPtab xPtab*.

A partir del valor del punter de pila abans de produïr-se la interrupció, calcula la quantitat d'informació afegida a pila per part del procés, n'extreu les adreces de retorn i les dades salvades per la interrupció deixant únicament les dades pròpies del procés. Recorre tota la pila de procés situant la informació a l'offset *XUSTACK_OFFSET* del procés *txPtab xPtab*.

void restore_timer_context (struct tPtab *TTask)

Descripció	Restarua l'estat del microntrolador a partir dels respectius atributs de l'estructura *txPtab xPtab* de la taula de processos del sistema
Precondició	El paràmetre *TTask* ha de ser de un punter a la posició de memòria de la taula interna de processos de tipus *tPtab* que relaciona el procés a restaurar amb la taula
Postcondició	El context de la tasca en execució és el restaurat de l'estructura que defineix l'entorn en la taula de processos *txPtab xPtab*
Implementació	Restaurar el context del procés que recupera l'execució implica plasmar les estructures que defineixen el context d'un procés sobre el microcontrolador. L'estructura que defineix la tasca és *txPtab Ptab* i específicament en el l'estat del processador és *tBCP BCP*, en el cas de la informació de planificació *tsched sched*, en el cas dels recursos *tresources resources*, en el cas de la pila *txustack xustack* i en el cas de la memòria de dades locals *txuimem xuimem*. En aquest cas, es realitza el procés invers que en la funció *save_timer_context(struct tPtab *TTask* i tant es continua utilitzant la combinació *&xPtab[num_task] + OFFSET_REGISTRE* per accedir als registres que descriuen el context a *txPtab xPtab*. A partir d'aquest recorregut pels diferents atributs de l'estructura *txPtab xPtab* es van restaurant els registres i estat del mirocontrolador. Es recupera la pila del procés emmagatzemada a *txustack xustack* i es prepara introduint atributs com el comptador de programa per tal que salti correctament a la línia d'execució tornant de l'interrupció. Abans de retornar a l'execució normal sortint de la interrupció, cal restaurar la configuració del timer0 per tal que es torni a produir una interrupció en l'interval predefinit.

Canvi de context fruit de les característiques d'execució d'una tasca

El canvi de context degut a factors aliens a l'interrupció del timer per la replanificació es realitza a través de les funcions

void save_context (struct tPtab *TTask,unsigned char sp)

Descripció	Aquesta funció té un comportament similar a *void save_timer_context (struct tPtab *TTask,unsigned char sp)* amb la diferència que al no tractar-se d'una funció que s'executa dins d'una interrupció, l'accés als registres del microcontrolador es realitza diferent.
Precondició	
Postcondició	
Implementació	

```
void restore_context (struct tPtab *TTask)
```

Descripció	Aquesta funció té un comportament similar a *void restore_timer_context (struct tPtab *TTask)* amb la diferència que al no tractar-se d'una funció que s'executa dins d'una interrupció, la cessió del control d'execució a la nova tasca es realitza diferent i no val reconfigurar el temporitzador ja que encara s'està executant en un interval entre interrupcions.
Precondició	
Postcondició	
Implementació	

4.3.2 Timer

La gestió del timer del microcontrolador es realitza treballant amb el conjunt de registres que el caracteritzen. El conjunt d'operacions que es realitzen són

- Inicialització
- Interrupció

Inicialització

La inicialització de qualsevol del temporitzador únicament precisa de la correcte configuració dels registres de funció especial referents al timer. En el cas del microcontrolador 8051 aquest registres són el *TMOD* i *TL-TH*.

```
void init_timer()
```

Descripció	Inicialitza els paràmetres del TIMER 0
Precondició	
Postcondició	
Implementació	

Interrupció

La interrupció que es produeix en l'interval de temps definit en la configuració del timer serveix per executar de forma periòdica la planificació de la seqüència d'execució de les tasques.

```
void timer0_isr (void) interrupt 1
```

Descripció	Planifica i realitza el canvi de control de l'execució per part d'un procés
Precondició	S'ha d'estar executant algun procés o estar en bucle infinit del sistema
Postcondició	La tasca de més prioritat de la cua de preparats té el control del processador
Implementació	La interrupció comprova si hi ha alguna tasca executant-se estudiant l'element *tPtab TExec*. En cas d'haver-hi una tasca executant-se anteriorment, la funció salva el context d'aquest cridant la funció *void save_timer_context (struct tPtab *TTask,unsigned char sp)* i la introdueix a la cua *tQueue QRdy*. En cas de no haver-hi cap tasca de l'usuari, salva el context de la tasca d'execució indefinida *while(1)* del sistema. En cada interval de planificació, cal comprovar si s'ha assolit el període d'una tasca que ha finalitzat la seva execució, és a dir, cal observar si en el temps d'execució es correspon a l'atribut *unsigned short P* de l'estructura *tinfo info* i si la tasca està en la cua de tasques realitzades *tQueue QDone*. En cas de coincidir amb el període, reinicialitza el comptador de temps d'execució *unsigned short t* de l'estructura *txPtab xPtab* de la tasca, i l'extreu de la cua *tQueue Qdone* i l'introdueix a la cua *tQueue QRdy*. La funció crida la funció *void oschedule(void)* esperant que seleccioni la tasca de més prioritat per la posterior execució. En cas que el comptador d'execució *usigned short t* sigui el mateix que el temps de computació definit pel programador de la tasca a *unsigned short C* de *tinfo info*, l'extreu de les cues *tQueue QRdy* i *tQueue QExec* per introduir-la a la cua *tQueue QDone*. A partir de la tasca executable obtinguda dels procediments anteriors, es recupera el context d'execució a través de la funció *void restore_timer_context (struct tPtab *TTask)* que cedeix l'execució en el microcontrolador a la nova tasca.

4.3.3 Semàfor

La gestió dels diferents semàfors que defineixen una sistema es realitza treballant amb el conjunt d'estructures referents a semàfors que caracteritzen una tasca. El conjunt d'operacions que es realitzen són

- Inicialització
- Up
- Down

Inicialització

La inicialització de qualsevol semàfor es realitza inicialitzant els atributs de les estructures de gestió de semàfors i relació entre tasca-semàfor.

```
void init_sems(void)
```

Descripció	Inicialitza els semàfors que gestionen el recursos compartits entre diferents tasques
Precondició	
Postcondició	Les estructures que defineixen els semàfors estan inicialitzades *tschedsem schedsem*
Implementació	Inicialitza l'atribut *unsigned char v* de l'estructura *tschedsem schedsem* de cada semàfor que el sistema utilitza per gestionar-hi l'accés

Up

La gestió del semàfor a nivell d'usuari es realitza a través de dues funcions a les quals el programador té accés.

```
void sem_up(tsemreg xdata *semreg, unsigned char nsem)
```

Descripció	A través d'aquest funció s'intenta aconseguir l'accés a un recurs compartit a partir de la gestió del corresponent semàfor associat i representat per l'estructura *tschedsem schedsem*
Precondició	El paràmetre *semreg* ha de ser de un punter a la posició de memòria de l'atribut *unsigned char sem_flag[num_sem]* de la taula de processos de tipus *txPtab xPtab* El paràmetre *nsem* ha de ser de tipus *unsigned char* i el semàfor del qual s'intenta obtenir el control
Postcondició	En cas que el semàfor no estigui controlat per cap altra tasca, accedirà a la gestió del semàfor i l'accés al recurs compartit, en cas contrari, quedarà bloquejat, abandonant la línia d'execució *tQueue QExec* i passant a la cua de tasques bloquejades *tQueue QBlk[num_sem]* pel semàfor *num_sem*
Implementació	Inicialment, i independentment de l'estat del semàfor, salvarà el context d'execució actual de la tasca a través de la funció *void save_context (struct tPtab *TTask,unsigned char sp)* d'aquesta manera, en cas de quedar bloquejat, quan torni a l'estat d'execució ho reprendrà des d'aquest punt, repetint les operacions de comprovació d'estat del semàfor que es realitzen a continuació. Comprova l'estat del semàfor *tschedsem schedsem[num_sem]* a través de la funció *void tsl(tsemreg xdata *semreg,struct tschedsem *lock)* per decidir sobre el futur de la tasca en execució. En cas d'observar que una altra tasca ja té el control del semàfor afegeix la tasca a la cua de tasques bloquejades *tQueue Qblk[num_sem]* a través de la funció *void t_blk(unsigned char nsem)*. Al quedar bloquejada la tasca en execució, busca una nova tasca per tal que reprengui l'execució replanificant l'execució a través de la funció *void oschedule(void)* que selecciona la tasca de més prioritat de la cua *tQueue QRdy*. Un cop seleccionada, restaura el context de la tasca que entra en execució a través de la funció *extern void restore_context (struct tPtab *TTask)*.

Down

La gestió del semàfor a nivell d'usuari es realitza a través de dues funcions a les quals el programador té accés.

```
void sem_down(tsemreg xdata *semreg, unsigned char nsem)
```

Descripció	A través d'aquest funció s'allibera el control d'un recurs compartit a partir de la gestió del corresponent semàfor associat i representat per l'estructura *tschedsem schedsem*

Precondició	El paràmetre *semreg* ha de ser de un punter a la posició de memòria de l'atribut *unsigned char sem_flag[num_sem]* de la taula de processos de tipus *txPtab xPtab* El paràmetre *nsem* ha de ser de tipus *unsigned char* i el semàfor del qual s'intenta obtenir el control
Postcondició	A l'alliberar el semàfor, restaura els valors inicials de la prioritat de la tasca i la treu de la cua d'execució *tQueue QExec* a través de la funció *void queue_qout(struct tPtab *pptab,struct tQueue *queue)*; introdueix la tasca sortint a la cua de preparats *tQueue QRdy* a través de la funció *void queue_in(struct tPtab *pptab,struct tQueue *queue)*.
Implementació	El fet de disposar del control sobre un semàfor i el corresponent recurs, pot implicar que durant el període entre l'obtenció del control i l'alliberament, alguna tasca hagi quedat bloquejada a l'intentar accedir al semàfor a través de la funció *void sem_up(tsemreg xdata *semreg, unsigned char nsem)*. Alliberat el semàfor, situa la tasca bloquejada de més prioritat presents a la cua *tQueue QBlk[num_sem]* del semàfor *num_sem* a la cua *tQueue QRdy* Restaurades les cues *tQueue QBlk[num_sem]* i *tQueue QRdy* executa el planificador de nou cridant la funció *void oschedule(void)* i restaurant el context de la nova tasca a executar a través de la funció *void restore_context (struct tPtab *TTask)*. El context que restaura és el salvat anteriorment en la funció *void sem_up(tsemreg xdata *semreg, unsigned char nsem)* i que situa l'execució abans de la comprovació de l'estat del semàfor.

4.3.4 Port sèrie

La gestió del port sèrie es realitza treballant amb l'estructura que el caracteritza. El conjunt d'operacions que es realitzen són

- Tasca de gestió
- Llegir
- Escriure
- Interrupció

Tasca de gestió

La inicialització del port sèrie es realitza gestionant els atributs de l'estructura, inicialitzant el mode de funcionament i inicialitzant el temporitzador relacionat.

```
void init_serial(unsigned char mode, unsigned short tr_speed)
```

Descripció	Inicialitza els paràmetres de configuració del port sèrie
Precondició	El paràmetre mode ha de ser de tipus unsigned char i correspon al mode funcionament del port sèrie MODE0,MODE1,MODE2,MODE3. El paràmetre tr_speed ha de ser de tipus unsigned short i correspon al paràmetre de configuració del timer1 associat al port sèrie
Postcondició	S'inicialitzen els atribut referents a la configuració del port sèrie de l'estructura tserial serial
Implementació	

```
void t_task_serial(void)
```

Descripció	Procés del sistema que gestiona la el port sèrie i que interactua amb la resta de processos a través de missatges
Precondició	Inicialització del sistema
Postcondició	En cas de rebre una petició d'escriptura, copia els bytes del contingut *char *tx_m* del missatge *tmsg msg* al buffer del port sèrie. En cas de rebre una petició de lectura, copia els bytes del buffer del port sèrie al punter *char *rx_m* de *tmsg msg*
Implementació	

Llegir

```
void read_serial(char xdata *info, unsigned char count)
```

Descripció	Genera un missatge de lectura del port sèrie
Precondició	El paràmetre info és un punter a una posició de memòria externa, molt probablement una variable local de la tasca. El paràmetre count ha de ser de tipus unsigned char i correspon al número de bytes que del buffer del port sèrie volem passar a la posició de memòria apuntada per char xdata *info
Postcondició	S'ha generat un missatge amb la petició de llegir *count* bytes del buffer i escriure'ls a *char *info* i s'ha afegit a la cua de missatges de la tasca que gestiona el port sèrie
Implementació	

Escriure

```
void write_serial(char xdata *info, unsigned char count)
```

Descripció	Genera un missatge d'escriptura al port sèrie
Precondició	El paràmetre info és un punter a una posició de memòria externa, molt probablement una variable local de la tasca. El paràmetre count ha de ser de tipus unsigned char i correspon al número de bytes que de l'orígen de dades apuntat per char xdata *info que volem passar al buffer del port sèrie
Postcondició	S'ha generat un missatge amb la petició d'escriure *count* bytes de *char *info* i s'ha afegit a la cua de missatges de la tasca que gestiona el port sèrie
Implementació	

Interrupció

```
static void serial_isr (void) interrupt 4
```

Descripció	Interrupció del port sèrie
Precondició	
Postcondició	En cas de tractar-se d'una interrupció per recepció d'un byte pel port sèrie el byte s'afegirà a l'última posició del buffer d'entrada. En cas de tractar-se d'una interrupció per transmissió d'un byte pel port sèrie, el byte s'extraurà de l'última posició del buffer de sortida
Implementació	

4.3.5 Cua

La gestió de les diferents cues que gestionen una sistema es realitza treballant amb el conjunt d'estructures referents a cues. El conjunt d'operacions que es realitzen són

- Introduir element
- Extreure element
- Selecció element amb més prioritat dins una cua

Introduir element

L'inserció d'un nou element a la cua és independent de l'estat que defineix cada cua, ja que en tots els casos el tipus de cua és el mateix *tQueue queue*.

```
void queue_in(struct tPtab *pptab,struct tQueue *queue)
```

Descripció	Introdueix un nou procés a una cua
Precondició	El paràmetre *pptab* ha de ser de un punter a la posició de memòria del procés en la taula interna de processos *tPtab Ptab* El paràmetre *queue* ha de ser de tipus *tQueue* i correspon a la cua a la qual volem introduir el procés
Postcondició	El procés representat a *tPtab *pptab* està siutat a la cua *tQueue *queue*
Implementació	La funció modifica l'atribut *tPtab *next* de l'element *tPtab *tail* de la cua *tQueue *queue* perquè apunt a la direcció de memòria de la nova tasca a introduir a la cua *tPtab *pptab*. Seguidament, modifica l'atribut *tPtab *prev* de l'element *tPtab *pptab* perquè apunti a la direcció de memòria de l'element *tPtab *tail*. El nou element és l'últim passa ser l'últim element de la cua, ja que la funció apunta l'atribut *tPtab *tail* de l'estructura *tQueue *queue* a la direcció de memòria de l'element *tPtab *pptab*

Extreure element

L'extracció d'un element de la cua, és independent de l'estat que defineix cada cua, ja que en tots els casos el tipus de cua és el mateix *tQueue queue*.

```
void queue_qout(struct tPtab *pptab,struct tQueue *queue)
```

Descripció	Extreu un nou procés d'una cua
Precondició	El paràmetre *pptab* ha de ser de un punter a la posició de memòria del procés en la taula interna de processos *tPtab Ptab* El paràmetre *queue* ha de ser de tipus *tQueue* i correspon a la cua de la qual volem extreure el procés
Postcondició	El procés representat a *tPtab *pptab* està extret de la cua *tQueue *queue*
Implementació	La funció busca si l'element a extreure és la direcció de memòria d'algun dels punters *tPtab *head* o *tPtab *tail* de la cua *tQueue *queue*, si és dóna el primer cas ,situa el punter *tPtab *next* de *tPtab *pptab* a *tPtab *head* a la cua, en el segon cas situa el punter *tPtab *prev* de *tPtab *pptab* a *tPtab *tail*. Modifica l'atribut *tPtab *next* de l'element *tPtab *Ptab* anterior perquè apunti a la direcció de memòria de l'element *tPtab *Ptab* següent obtingut de l'atribut *tPtab *next* de la tasca sortint. Modifica l'atribut *tPtab *prev* de l'element *tPtab *ptab* següent perquè apunt a la direcció de memòria de l'element *tPtab *Ptab* anterior obtingut de l'atribut *tPtab *prev* de la tasca sortint.

Selecció element amb més prioritat dins d'una cua

La selecció de l'element de la cua és independent de l'estat que defineix cada cua, ja que en tots els casos el tipus de cua és el mateix *tQueue queue*.

```
tPtab *get_hpriority_task(struct tQueue *queue)
```

Descripció	Busca el procés de més prioritat en una cua de tasques
Precondició	El paràmetre *queue* ha de ser de tipus *tQueue* i correspon a la cua en la qual volem buscar el procés de més prioritat
Postcondició	Retorna un punter de la posició de memòria de la taula de memòria interna ocupada pel procés de més prioritat i representat per una estructura de tipus *tPtab*.
Implementació	La funció recorre tots els elements de la taula *txPtab xPtab* situats a la cua *tQueue *queue* a partir dels punters a direcció de memòria externa *txPtab *pximage* de *tPtab Ptab* on hi ha les estructures d'informació de les tasques. Compara l'atribut *tpriority priority* de tots els elements buscant el de més prioritat, que en aquest cas, és el que té el valor més petit de l'atribut *tpriority priority*.

Capítol 5

Manual d'usuari

En la utilització del planificador, el programador de les diferents tasques que s'executaran ha de disposar d'informació respecte al sistema, les seves tasques i la relació entre ambdós. A partir d'aquesta informació, es realitza el càlcul de planificació i s'inicia l'execució de la tasca.

5.1 Característiques de la tasca

El conjunt de característiques que ha de tenir el programador són:

- Característiques d'identificació
- Característiques temporals
- Característiques d'utilització de recursos compartits

A part, haurà de disposar de la tasca que vol executar al sistema amb les corresponents modificacions pel compartiment de recursos.

5.1.1 Característiques d'identificació

Les característiques d'identificació d'una tasca són:

pid	Identificador únic del procés que ha de ser inferior al número màxim tasques del sistema
ppid	Identificador únic que correspon a l'identificador del procés pare (inhabilitat)
uid	Identificador únic d'usuari al qual s'associa el procés que ha de ser inferior al número màxim d'usuaris del sistema (inhabilitat)

5.1.2 Característiques temporals

Les característiques temporals d'una tasca són:

C	Interval de temps de computació d'una tasca
P	Període d'execució d'una tasca

5.1.3 Característiques d'utilització de recursos compartits

Les característiques d'utilització de recursos compartits d'una tasca es defineixen a partir de l'interval de temps necessari d'accés a un recurs compartit gestionat per un semàfor. En l'anàlisi d'aquestes característiques cal tenir present aquest interval de temps per a cada un dels recursos compartits.

num_sem	Recursos compartits al qual es vol accedir
t_sem	Interval de temps d'accés a una recurs compartit necessari

5.2 Programació d'una tasca

A partir de les característiques obtingudes en l'anàlisi previ, cal representar-ho en la tasca ja programada per part de l'usuari. L'inclusió d'aquesta informació a la tasca, es realitza utilitzant un seguit de funcions que el sistema posa a servei del programador.

5.2.1 Inici definició d'una tasca i característiques d'identificació

Per introduir les característiques d'identificació a la tasca programada cal utilitzar la funció

```
rt_task_init(pid,ppid,uid)
```

Els paràmetres de la funció han de ser de tipus

- **pid** unsigned char
- **ppid** unsigned char
- **uid** unsigned char

5.2.2 Característiques temporals

Per introduir les característiques temporals a la tasca programada cal utilitzar la funció

```
rt_task_schedule(pid,C,P)
```

Els paràmetres de la funció han de ser de tipus

- **pid** unsigned char
- **C** unsigned short
- **P** unsigned short

5.2.3 Característiques d'utilització de recursos compartits

Per introduir les característiques temporals a la tasca programada cal utilitzar la funció

```
rt_sem(pid,semid,semt)
```

Els paràmetres de la funció han de ser de tipus

- **pid** unsigned char
- **semid** unsigned short (correspon al número de recurs compartit del qual es defineix l'interval de temps del paràmetre següent)
- **semt** unsigned short (correspon a l'interval de temps d'accés al número de recurs compartir que es defineix en el paràmetre anterior)

5.2.4 Fi definició d'una tasca

Per finalitzar la definició d'una tasca programada cal utilitzar la funció

```
rt_task_init_end()
```

5.2.5 Inici tasca de l'usuari

Per definir l'inici de la tasca programada de l'usuari cal utilitzar la funció

```
rt_task(pid)
```

- **pid** unsigned char

5.2.6 Fi tasca de l'usuari

Per definir la fi de la tasca programada de l'usuari cal utilitzar la funció

```
rt_task_end()
```

5.2.7 Adquirir el control d'un recurs compartit

Per definir l'intent d'adquisició de control d'un recurs compartir durant l'execució de la tasca programada per l'usuari, cal utilitzar la funció

```
rt_sem_up(semid)
```

- **semid** unsigned short (correspon al número de recurs compartit del qual volem obtenir el control)

5.2.8 Alliberar el control d'un recurs compartit

Per definir l'alliberament del control d'un recurs compartir durant l'execució de la tasca programada per l'usuari, cal utilitzar la funció

```
rt_sem_down(semid)
```

- **semid** unsigned short (correspon al número de recurs compartit del qual volem obtenir el control)

5.2.9 Escriure al port sèrie

Per escriure al port sèrie utilitzar la funció

```
void write_serial(char xdata *info, unsigned char count)
```

- **info** *char (correspon al punter a la informació)
- **count** unsigned char (correspon al número de bytes a transmetre)

5.2.10 Llegir del port sèrie

Per llegir del port sèrie cal haver-lo inicialitzat i utilitzar la funció

```
void read_serial(char xdata *info, unsigned char count)
```

- **info** *char (correspon al punter a la informació)
- **count** unsigned char (correspon al número de bytes a llegir)

5.3 Característiques de la tasca programada per l’usuari

Segons la definició de característiques de la tasca realitzada a partir de les funcions

- rt_task_init(pid,ppid,uid)
- rt_task_schedule(pid,C,P)
- rt_sem(pid,semid,semt)
- rt_task_init_end()

i la tasca programada per l’usuari amb l’inclusió de les funcions

- rt_task(pid)
- rt_task_end()

- rt_sem_up(semid)
- rt_sem_down(semid)

l'estructura d'una tasca programa per l'usuari és

```
rt_task_init(pid,ppid,uid)
rt_task_schedule(pid,C,P)
rt_sem(pid,semid,semt)
rt_task_init_end()
-----------------------------
rt_task(pid)
...
rt_sem_up(semid)
...
rt_sem_down(semid)
...
rt_task_end()
```

Figura 5.1: Estructura

5.4 Compilar i enllaçar

Per compilar i enllaçar el conjunt de tasques programades per l'usuari i el sistema, cal tenir els següents fitxers

- **process_adm.c** Fitxer principal del sistema. Inicialitza les diferents tasques i la gestió de recursos
- **rtsched.lib** Llibreria amb el conjunt de funcions del sistema
- **taskX.c** Tasca programada per l'usuari. X correspon al número de tasca, que va des de 0 fins el número màxim de tasques que suporta el sistema

Per la compilació i enllaç, són necessaris compilador i enllaçador, ambdos formen part del paquet de desenvolupament de l'empresa Keil http://www.keil.com.

Per tal d'executar el preprocessador cal disposar d'un intèrpret de Perl pel sistema operatiu del qual tenim les eines de compilació i enllaç, en el cas del paquet de Keil seria per MS Windows. A part, cal disposar dels fitxers

- **perl_macros.pl** Preprocessador
- **tags.def** Definició de tipus de macros
- **tags_contents.def** Definició de macros

Per iniciar la compilació i enllaç cal executar la següent comanda

```
perl perl_macros.pl rtsched.lib process_adm.c taskX.c
```

Fruit de l'execució de la comanda anterior, es genera el fitxer *proccess* que es pot utilitzar amb l'emulador per comprovar el correcte funcionament del sistema plantejat.

Capítol 6

Conclusions i línies de futur

6.1 Conclusions

A continuació s'especifica el conjunt de tasques que s'han realitzat en aquest projecte:

- S'ha realitzat un estudi teòric sobre els tres temes centrals en que es basa el projecte: *microcontrolador 8051*, *sistemes operatius* i *temps real*
- S'ha realitzat un anàlisi sobre la viabilitat de la implementació d'una combinació de *sistema operatiu* i *temps real* sobre la plataforma d'un *microcontrolador 8051*
- S'ha dissenyat i implementat un planificador de processos segons els requisits especificats en sistemes de *temps real*
- S'han analitzat i implementat funcions addicionals de *semàfors* i *gestió de dispositius E/S*.
- S'han analitzat i implementat les eines per desenvolupar tasques que funcionin sobre el planificador implementat *preprocessador*
- S'han analitzat els resultats obtinguts a nivell teòric i pràctic i s'han buscat noves línies futures per aquest projecte.

6.2 Línies de futur

Aquest projecte té vàries línies de futur enfocades bàsicament en els següents camps:

- Programari de desenvolupament

- Microcontrolador 8051
- Sistema operatiu
- Temps real

6.2.1 Programari de desenvolupament

Una de les primeres línies en les quals s'ha avançar, és en una recerca exhaustiva de programari lliure que permeti desenvolupar projectes similars a aquest. Inicialment es va fer una recerca d'aquest tipus de programari, on es va observar que complia les expectatives necessàries per realitzar les pràctiques de l'assignatura *Microcontroladors* de la carrera de *ET Electrònica Industrial*. Tot i això, precisava d'una anàlisi més profund per veure la viabilitat en la utilització d'un projecte com el que es presenta en aquesta memòria.

Una altra possibilitat és la millora de les eines ja existents per tal d'adaptar-les a les necessitats específiques de l'assignatura *Microcontroladors*. Un primer pas podria seria la integració en un sol entorn de desenvolupament de les diferents aplicacions individuals amb funcionalitats específiques: edició, compilació, enllaç, emulació, etc.

6.2.2 Microcontrolador 8051

En les línies referents al microcontrolador, seria interessant provar el planificador implementat en aquest projecte en entorns reals. També caldria realitzar *estudis de rendiment* del planificador en diferents microcontroladors i situacions.

En la definició de les especificacions del propi microcontrolador, es podria utilitzar XML per lligar-ho amb aplicacions externes.

En la relació amb el sistema operatiu, seria interessant desenvolupar del driver el màxim de dispositius d'entrada-sortida que puguin tenir els diferents models de l'arquitectura 8051.

En la relació amb les pròpies tasques, resultaria guany qualitatiu la possibilitat de carregar les tasques a partir dels fitxers objecte generats per un compilador. En en el desenvolupament d'aquest projecte es va intentar implementar un carregador segon les especificacions OMF-51, vegeu [8]. Degut a problemes amb la interpretació de la implementació del compilador *C51* de l'empresa *Keil* no va ser possible

6.2.3 Sistema operatiu

En les línies referents al sistema operatiu, seria interessant realitzar una implementació més modular, convertint la lògica del sistema operatiu transparent a les característiques específiques d'un model de microcontrolador.

Caldria replantejar la gestió de memòria que realitzar el planificador perquè fos més dinàmica, donant més llibertat al programador de tasques i no limitant el sistema.

Caldria replantejar els dispositius de comunicació entre processos que s'han implementat en aquest projecte. Al no ser l'objectiu inicial del projecte són una implementació bàsica i tampoc contemplen d'altres possibilitats com poden ser els monitors

Seria molt interessant desenvolupar les crides de sistema més comunes en el sistemes operatius (POSIX,etc.), sempre tenint present les restriccions dels sistemes en temps real.

Com a salt qualitatiu seria molt interessant desenvolupar algun model de sistema que permetés treballar en paral·lel a diferents microcontroladors i balancejar la càrrega de treball d'un sistema encastat.

6.2.4 Temps real

En les línies referents al temps real, seria interessant realitzar un estudi dels diferents algorismes que s'han plantejat a l'introducció teòrica i realitzar-ne una implementació. Segurament caldria modificar parcialment aquest planificador per tal que l'algorisme de planificació i planificabilitat fos transparent al propi sistema. Cal tenir present les característiques i restriccions del propi microcontrolador i dels compiladors eines de desenvolupament

Seria interessant provar l'algorisme implementat en aquest projecte en entorns reals. També caldria realitzar *estudis de rendiment* dels diferents algorismes possibles, com també de situacions no-acotables, tasques aperiòdiques, etc

Capítol 7

Agraïments

Segurament hauria de recórrer bona part de la meva vida per poder donar les gràcies a qui, directament o indirecta, han col·laborat en la gestació i elaboració d'aquest projecte.

Les limitacions d'espai i memòria personal m'obliguen a restringir-me als últims tres mesos.

Gràcies a la Fàtima, Ramon i Josep Maria pel suport en els múltiples àmbits que intervenen en l'elaboració d'un projecte i en les qüestions quotidianes de la pròpia existència

Gràcies al personal de la biblioteca per la col·laboració en les tasques de recerca d'informació.

Gràcies al professor Jesús Vicente (Departament d'Enginyeria Electrònica de l'EUPM) per la col·laboració en les etapes inicials del projecte i per l'assignatura *Microcontroladors* . . . d'allí va sorgir la idea.

Gràcies al tutor del projecte Sebastià Vila (Departament de Llenguatges i Sistemes informàtics de l'EUPM) pel suport ètic, estètic, sociològic i, sobretot i en escreix, tècnic/tecnològic.

Salut i records pel Sr. Winston(1984-) i Sr. Ferrer i Guàrdia(1901-)

Apèndix A

Codi font

Els codi font de bona part de la implementació realitzada es troba descrit segons el següent índex:

perl_macros.pl
Preprocessador

process_adm.c
Funció principal del programa

init_tasks.c
void init_task(unsigned char ntask,void *func)
void reset_task(struct txPtab xdata *lxp)

end_tasks.c
void end_t_task(void)

scheduler_r1.c
char order_tasks(unsigned char left, unsigned char right)
char blk_times(void)
unsigned short workload(unsihed char l, unsigned short t)
char rms_schedule_eworkload(void)
char extract_lcm(void)

scheduler_r2.c void t_blk(unsigned char nsem)
void oschedule(void)

save_context.a51
void save_context (struct tPtab *TTask,unsigned char sp)

restore_context.a51

void restore_context (struct tPtab *TTask)

save_idle.a51
void save_idle (unsigned char sp)

restore_idle.a51
void restore_idle (unsigned char sp)

timer.c
void timer0_isr (void)
void init_timer(void)

sem.c void init_sems(void)
void tsl(tsemreg xdata *semreg,tschedsem xdata *lock)
void sem_up(tsemreg xdata *semreg, unsigned char nsem)
void sem_down(tsemreg xdata *semreg, unsigned char nsem)

serie.c void serial_isr (void)
void write_serial(char xdata *info, unsigned char count)
void read_serial(char xdata *info, unsigned char count)
void init_serial(unsigned char mode, unsigned short tr_speed)

task_serial.c
void init_t_task_serial(void)
void t_task_serial(void)

queues.c
void queue_in(struct tPtab *pptab,struct tQueue *queue)
void queue_qout(struct tPtab *pptab,struct tQueue *queue)
char inside_queue(unsigned char PID, struct tQueue *queue)
tPtab *get_hpriority_task(struct tQueue *queue)
void mqueue_qout(struct tms *mmsg, struct tmsg_queue *queue)

Listing A.1: perl_macros.pl

```
1
2  $base_address = hex 7C00;
3   $offset  = hex C00;
4
5  @tags =   initialize_tags  ("tags.def");
6
7  foreach $arg_num  (0 .. $# ARGV) {
8       $fitxers [$arg_num] = $ARGV[$arg_num];
9  }
10
11 $nt  = 0;  $num_c = 0; $num_lib = 0;
12
13 foreach $f( @fitxers ) {
14    if  ( $f  =~ /^*. c$ /) {
```

```
15        $fitxers_compile [$num_c]->{source}=$f;
16        if ($f =~ /^task[0-9]{1,2}.c$/) {
17          $nt++;
18           $fitxers_compile [$num_c]->{type}="t";
19        }else{
20           $fitxers_compile [$num_c]->{type}="n";
21        }
22        $num_c++;
23     } elsif ($f =~ /^*.lib$/) {
24        $num_lib++;
25         $fitxers_lib [$num_lib]=$f;
26     }
27  }
28
29  $num_c--;
30  $nt--;
31
32
33   rt_preprocessor ();
34   rt_init ();
35
36  @LINK_OPT = rt_init_files(@fitxers_compile);
37
38  $msg = rt_compile_files (@LINK_OPT);
39  print $msg;
40
41  @LINK_OPT = rt_add_libs(@LINK_OPT);
42
43  $msg = rt_link_files (@LINK_OPT);
44  print $msg;
45
46  $msg = execute_linker("command.L51");
47  print $msg;
48
49  sub analyze_file {
50     local($tags,$f) = @_;
51     local($i,$j,$descr,$c,$cs);
52
53     open(FILE,$f);
54     @fc = <FILE>;
55     close(FILE);
56
57     for ($i=0;$i<=$#fc;$i++) {
58        if ($fc[$i] =~ /(rt_\w*\((\w+,*)*\))/) {
59           $descr= extract_description ($1);
60           for ($j=0;$j<= $#tags;$j++) {
61              if ($descr->{name} eq $tags[$j]->{name}) {
62                 $tags[$j]->{contents}=get_tags_contents($tags[$j],"tags_contents.def");
63                 $cs .= replace_arguments($descr,$tags[$j]);
64              }
65           }
66        }else{
67           $cs .= $fc[$i];
68        }
```

```
69      }
70      return $cs;
71  }
72
73  sub replace_arguments {
74      local($d,$t)=@_;
75      local($l);
76
77      for ($l=0;$l<=$t->{num_arg};$l++) {
78          while (($t->{contents} =~ s/$t->{args}[$l]{name}/$d->{arg}[$l]{name}/)){};
79      }
80      return $t->{contents};
81  }
82
83
84  sub get_tags_contents {
85
86      local($tag,$f) = @_;
87      local $contents,$flg,$k,@tag_contents;
88
89      open(FILE,$f);
90      @tags_contents = <FILE>;
91      close(FILE);
92
93      $flg = 0;
94      for ($k=0;$k<=$#tags_contents;$k++) {
95          if ($tags_contents[$k] =~ /^\[$tag->{file}\]/) {
96              $flg = 1;
97          } elsif ($tags_contents[$k] =~ /^\[\w*\]/) {
98              $flg = 0;
99          } elsif ($flg == 1) {
100             $contents .= $tags_contents[$k];
101         }
102     }
103     return $contents;
104 }
105
106
107 sub initialize_tags {
108
109     local($f) = @_;
110
111     open(FILE,$f);
112     @ftags = <FILE>;
113     close(FILE);
114
115     for($num_tags=0;$num_tags <= $#ftags;$num_tags++) {
116
117         if ($ftags[$num_tags] =~ /(.*)\:\:(\w*)/) {
118             $tagdescr = $1;
119             $tags[$num_tags]->{file} = $2;
120         }
121
122         $descr= extract_description($tagdescr);
```

```
123       $tags[$num_tags]->{name}=$descr->{name};
124       $tags[$num_tags]->{num_arg}=$descr->{num_arg};
125       $tags[$num_tags]->{args}=$descr->{arg};
126     }
127     return @tags;
128 }
129
130 sub  extract_description  {
131
132     local ($tagdescr)=@_;
133
134     if  ($tagdescr     =~ /(.*)\((.*)\)/) {
135       $descr->{name} = $1;
136       $tagdescr  =  $2;
137
138       $num_arg = 0;
139       while  ($tagdescr  =~ /(\w+),?(.*)/) {
140         $descr->{arg}[$num_arg]{name}=$1;
141         $tagdescr  =  $2;
142         $num_arg++;
143       }
144       $descr->{num_arg}=$num_arg;
145     }
146     return  $descr;
147 }
148
149 sub  rt_preprocessor  {
150     open(FILE,"» preprocessor_global.h");
151       print  FILE "#define␣PREP_NUM_TASKS␣" . $nt;
152     close(FILE);
153 }
154
155 sub  rt_init  {
156     local  $contents, $flg, $k,$m,@tag_contents;
157
158     open(FILE,"process_adm.c");
159     @tags_contents = <FILE>;
160     close(FILE);
161
162     open(FILE,">process_adm.c");
163
164     for  ($k=0;$k<=$#tags_contents;$k++) {
165       if  ($tags_contents[$k] =~ /  os_init_tasks  ()/) {
166         print  FILE "␣␣num_tasks␣=␣PREP_NUM_TASKS;\n\n";
167         for  ($m=0;$m<=$nt;$m++) {
168           print  FILE "␣␣init_task(" . $m . ",&t_task" . $m . ");\n";
169           print  FILE "␣␣init_t_task" . $m . "();\n";
170         }
171       } elsif  ($tags_contents[$k] =~ /  os_define_tasks  ()/) {
172         for  ($m=0;$m<=$nt;$m++) {
173           print  FILE "extern␣void␣t_task" . $m . "(void);\n";
174           print  FILE "extern␣void␣init_t_task" . $m . "(void);\n\n";
175         }
176       } else {
```

```
177         print FILE $tags_contents[$k];
178       }
179     }
180     close(FILE);
181 }
182
183 sub   rt_init_files   {
184
185     local(@fitxers_compile)=@_;
186     local($x, $ft, $fc, @OPT);
187     local  ($cr)=0;
188
189     for  ($x=0;$x<=$#fitxers_compile;$x++) {
190
191       $ft  =  $fitxers_compile [$x]->{source};
192       $ft  =~ s /\./ _C51./;
193
194       $OPT[$x]->{source}=$ft;
195
196       $ft  =~ s /\. c //;
197       $OPT[$x]->{module}=$ft;
198
199       $ft  .=  ".obj";
200       $OPT[$x]->{object}=$ft;
201
202       $OPT[$x]->{type}= $fitxers_compile[$x]->{type};
203
204       if  ($OPT[$x]->{type} eq "t") {
205         $OPT[$x]->{address}=$base_address + ($cr*$offset);
206         $cr++;
207       }
208
209       $fc  =  analyze_file ($tags, $fitxers_compile [$x]->{source});
210       open(FILE,">$OPT[$x]->{source}");
211         print  FILE $fc;
212       close(FILE);
213     }
214     return  @OPT;
215 }
216
217 sub  rt_compile_files  {
218
219     local(@OPT) = @_;
220     local($x,$fname,@output);
221
222     for  ($x=0;$x<=$#OPT;$x++){
223       $fname = $OPT[$x]->{module} . ".cl";
224       open(FILE,">$fname");
225       print  FILE $OPT[$x]->{source};
226       print  FILE " LARGE BROWSE DEBUG OBJECTEXTEND ";
227       close(FILE);
228       @output = execute_compiler($fname,$OPT[$x]->{source});
229     }
230     return  $output;
```

```
231 }
232
233 sub  rt_link_files  {
234    local (@OPT) = @_;
235    local ($x, $cr );
236
237    open(FILE,">command.L51");
238
239    for  ( $x=0;$x<=$#OPT;$x++){
240       print  FILE "\"" .  $OPT[$x]->{object} . "\"";
241       if  ( $x  != ($# OPT)) {
242          print  FILE ",";
243       }
244       print  FILE "\n";
245    }
246
247    print  FILE "TO␣\"process\"\n";
248    print  FILE "RAMSIZE(256)\n";
249    print  FILE "XDATA";
250
251    $cr=0;
252    print  FILE "(␣";
253    print  FILE "?XD?t_task_serial? task_serial (7000h),";
254    for  ( $x=0;$x<=$#LINK_OPT;$x++){
255       if  ( $OPT[$x]->{type} eq "t") {
256          print  FILE "␣?XD?t_task" . $cr  . "?"  .  $OPT[$x]->{module} . "(";
257          printf  FILE ("%x",$OPT[$x]->{address});
258          print  FILE "h)";
259          if  ( $cr  == $nt ) {
260             print  FILE "␣)";
261          } else  {
262             print  FILE ",";
263          }
264          $cr++;
265       }
266    }
267    print  FILE "\nIDATA(␣?STACK(80h)␣)";
268    close (FILE);
269 }
270
271 sub dump_tags {
272
273    local (@tags)=@_;
274    local ($i );
275
276    foreach  $t(@tags) {
277       print  "\n\nFile:␣" .  $t->{file};
278       print  "\nName:␣" . $t->{name};
279
280       $i=0;
281       while  ( $t->{args}[$i ]) {
282          print  "\n␣␣Argument:" . $t->{args}[$i]{name};
283          $i++;
284       }
```

```
285         print "\nContents: " . $t->{contents};
286     }
287 }
288
289 sub execute_compiler {
290     local ($f1,$f2) = @_;
291     local @output,$cmd;
292
293     $cmd = "C51 @" .$f1;
294     print "\n\n!! Compilant fitxer $f2:  $cmd\n";
295     @output = system($cmd);
296     return $output;
297
298 }
299
300 sub execute_linker {
301     local ($f1) = @_;
302     local @output,$cmd;
303
304     $cmd = "BL51 @" .$f1;
305     print "\n\n!! Linker $f1:  $cmd\n";
306     @output = system($cmd);
307     return $output;
308
309 }
310
311 sub rt_add_libs {
312     local (@OPT) = @_;
313     local ($x,$cr);
314
315     $cr = $#OPT+1;
316
317     for ($x=0;$x<$# fitxers_lib ;$x++) {
318         $OPT[$cr+$x]->{object} = $fitxers_lib [$x-1];
319         $OPT[$cr+$x]->{type} = "l";
320     }
321
322     return @OPT;
323 }
```

Listing A.2: process_adm.c

```
1  void main (void) {
2
3   // funcions generades pel preprocessador
4  // ######################
5      os_init_tasks ()
6  // ######################
7
8
9      init_task (T_SERIAL_ID,t_task_serial);
10      init_t_task_serial ();
11
12     order_tasks (0,num_tasks);
13
```

```
14    num_sems = NUMSEMS_MAX;
15    init_sems ();
16
17    blk_times ();
18    if (rms_schedule_checking() == SCHED_ERROR) {
19      if (rms_schedule_eworkload() == SCHED_ERROR) {
20        // no compleix els requisits de planificabilitat
21        // exit ();
22      }
23    }
24    // compleix els requisits de planificabilitat
25
26    extract_lcm ();
27
28    for (l=0;l<=num_tasks;l++) {
29      queue_in(&Ptab[l],&QDone);
30    }
31
32    init_timer ();
33    TExec = NULL;
34
35    while(1);
36  }
```

Listing A.3: init_tasks.c

```
1   void init_task (unsigned char ntask,void *func) {
2     xPtab[ntask].bcp.mpstatus.mregisters.iPC = func;
3   }
4
5
6   void reset_task (struct txPtab xdata *lxp) {
7
8     lxp->bcp.sched.info.t=0;
9     lxp->bcp.sched.priority = lxp->bcp.sched.info.P;
10
11    lxp->bcp.mpstatus.mregisters.PC = lxp->bcp.mpstatus.mregisters.iPC;
12    lxp->bcp.mpstatus.stack.SP = IUSTACK_BASE;
13
14    lxp->bcp.mpstatus.flags.PSW = 0;
15  }
```

Listing A.4: end_tasks.c

```
1   void end_t_task(void) {
2
3     EA = 0;
4
5     queue_qout(TExec,&QExec);
6     queue_in(TExec,&QDone);
7
8   //  EA = 0;
9
10    SP = IOSTACK_BASE;
11    oschedule();
```

```
12    xp = QExec.tail->pximage;
13    TExec = QExec.tail;
14     restore_context (TExec);
15
16    EA = 1;
17  }
```

Listing A.5: scheduler_r1.c

```
1
2   char order_tasks(unsigned char left, unsigned char right) reentrant {
3
4     l_hold=left;
5     r_hold=right;
6
7     schedPpivot = xPtab[ left ];
8
9     while ( left < right ) {
10
11      while ((xPtab[ right ].bcp.sched.info.P >= schedPpivot.bcp.sched.info.P) && (left < right )) right--;
12
13
14      if ( left != right ) {
15        xPtab[ left ] = xPtab[ right ];
16        Ptab[ right ].PID=xPtab[left ].bcp.ident.pid;
17
18         left ++;
19      }
20
21
22      while ((xPtab[ left ].bcp.sched.info.P <= schedPpivot.bcp.sched.info.P) && (left < right )) left ++;
23      if ( left != right ) {
24        xPtab[ right ] = xPtab[ left ];
25        Ptab[ left ].PID=xPtab[right ].bcp.ident.pid;
26         right--;
27      }
28    }
29
30    xPtab[ left ] = schedPpivot;
31    Ptab[l_hold ].PID=xPtab[left ].bcp.ident.pid;
32
33    iPpivot = left;
34    left = l_hold;
35    right = r_hold;
36
37    if ( left < iPpivot ) {
38      order_tasks ( left , iPpivot-1);
39    }
40    if ( right > iPpivot ) {
41      order_tasks ( iPpivot+1, right );
42    }
43
44    return ORDER_OK;
45  }
46
```

```
47
48
49
50  char blk_times(void) {
51
52   // ## Bloqueig directe
53
54   // calcula els temps de bloqueig de cada
55   // tasca en funcio dels semafors corresponents
56
57   // mante una taula per la saber el temps de bloqueig
58   // maxim que hi pot haver entre dues tasques
59
60   // recorre totes les tasques comprovant el maxim
61   // temps de bloqueig (es a dir comprovant cada
62   // semafor) entre una tasca i la resta de
63   // tasques i emmagatzema sempre la mes gran.
64   // Per cada tasca i, guarda el valor maxim del semafor j
65   // que pot estar relacionat amb una tasca de menor
66   // prioritat, es a dir una tasca k.
67   // d_b[i][k] semafor j de i mes gran
68
69     for (i=0;i<=(num_tasks-1);i++) {
70       blktime=0;
71       for (j=0;j<num_sems;j++) {
72             if (xPtab[i].bcp.resources.sem.sem[j] != 0){
73               for (k=i+1;k<=num_tasks;k++) {
74                     if (direct_blocking[i][k] < xPtab[k].bcp.resources.sem.sem[j]) {
75                       direct_blocking[i][k] = xPtab[k].bcp.resources.sem.sem[j];
76                     }
77                     if (blktime < direct_blocking[i][k]) {
78                       blktime = direct_blocking[i][k];
79                     }
80                   }
81             }
82           }
83           blktime_none[i] = blktime;
84     }
85     blktime_none[num_tasks]=0;
86
87   // ## Bloqueig per herencia
88
89   // inheritance protocol
90   // comprova els possibles bloquejos per herencia,
91   // es a dir, en cas que una tasca Ti d' inferior
92   // pes que Tj, pot bloquejar Tj (directament), en
93   // aquest cas, Ti assumiria el pes (heredaria)
94   // de la tasca Tj, i podria bloquejar les tasques
95   // que hi ha entre Ti i Tj
96
97     for (j=1;j<=num_tasks;j++) {
98       blktime=0;
99       for (i=1;i<=j && i<=num_tasks-1;i++) {
100            if (blktime < direct_blocking[i-1][j]) {
```

```
101                 blktime = direct_blocking [i-1][j];
102               }
103               inheritance_blocking [i][j] = blktime;
104             }
105     }
106
107     for (i=0;i<=(num_tasks-1);i++) {
108       blktime=0;
109             for (j=i+1;j<=num_tasks;j++) {
110               need_sem=0;
111               for (k=0;k<num_sems;k++){
112                 if (xPtab[i].bcp.resources.sem.sem[k] != 0 && (sem_ceil[k] \\
113                 <= xPtab[i].bcp.sched.info.P)) {
114                         need_sem = 1;
115                         break;
116                       }
117               }
118               if (need_sem) {
119                 ceiling_blocking [i][j] = inheritance_blocking [i][j];
120               }else{
121                 ceiling_blocking [i][j] = 0;
122               }
123               if (blktime < ceiling_blocking [i][j]) {
124                 blktime = ceiling_blocking [i][j];
125               }
126             }
127             blktime_ceiling [i] = blktime;
128     }
129      blktime_ceiling [num_tasks]=0;
130
131     for (i=0;i<=(num_tasks-1);i++) {
132       if (blktime_none[i] > 0) {
133               blktime = blktime_none[i];
134             }else{
135               blktime=0;
136       }
137
138             if (blktime < blktime_ceiling [i]) {
139               blktime = blktime_ceiling [i];
140             }
141       xPtab[i].bcp.sched.info.B = blktime;
142     }
143     xPtab[i].bcp.sched.info.B = 0;
144
145     return OK;
146   }
147
148   char rms_schedule_checking(void) {
149
150    // implementacio de l'algorisme de comprovacio
151    // de planificacio de temps real RMS
152
153    //    per tot i, on 1<=i<=n c1/p1 + c2/p2 + (ci+bi)/pi ha
154    // de ser <= i(s^(1/i)-1)
```

```
155 // on c es el temps d'execucio␣i␣p␣el␣íperode
156
157   for ( i=0;i<=num_tasks;i++) {
158
159     rms_left  = 0.0;
160         rms_right=( float )( i +1.0)*((pow (2.0,1.0/( float )( i +1.0)))-1.0);
161
162         for ( j=0;j<=i;j++) {
163
164       xp=Ptab[j ].pximage;
165       if ( j  !=  i ) {
166                   rms_left  += ( float )xp->bcp.sched.info.C/( float )xp->bcp.sched.info.P;
167           }else{
168             rms_left  += ((( float )xp->bcp.sched.info.C ) + \\
169             (( float )xp->bcp.sched.info.B ))/(( float )xp->bcp.sched.info.P);
170           }
171         }
172     if ( rms_right  <  rms_left ) {
173       return SCHED_ERROR;
174     }
175   }
176   return SCHED_OK;
177 }
178
179
180 unsigned short  workload(unsigned char l,unsigned short  t ) {
181
182   sum=0;
183   for ( k=0;k<=l;k++) {
184     xp = Ptab[k ].pximage;
185     if ( k == l ) {
186           sum += (xp->bcp.sched.info.C + xp->bcp.sched.info.B ) *  \\
187           (unsigned short) ceil ((( float )t )/(( float )xp->bcp.sched.info.P));
188         }else{
189           sum += xp->bcp.sched.info.C *(unsigned short) ceil (( float )t /( float )xp->bcp.sched.info.P);
190         }
191   }
192   return sum;
193 }
194
195
196 char rms_schedule_eworkload(void) {
197
198
199   for ( i=0;i<=num_tasks;i++) {
200     interval =0;
201     for ( j=0;j<=i;j++) {
202           xp=Ptab[j ].pximage;
203       interval  += xp->bcp.sched.info.C;
204         }
205
206
207         while (( tinterval  = workload(i, interval )) !=  interval ) {
208           interval = tinterval ;
```

```
209             xp= Ptab[ i ]. pximage;
210             if ( interval > ( xp−>bcp.sched.info.P )) {
211               return SCHED_ERROR;
212             }
213           }
214
215   }
216   return SCHED_OK;
217 }
218
219 char extract_lcm(void) {
220
221   lcm = xPtab[num_tasks].bcp.sched. info .P;
222   k = 1;
223
224   for ( i=(num_tasks−1);((i>=0)&&(i<255));i−−) {
225     if ( xPtab[ i ]. bcp.sched. info .P != xPtab[num_tasks].bcp.sched. info .P) {
226           l=1;
227           while (( xPtab[ i ]. bcp.sched. info .P∗(k+l )) < lcm) l++;
228       if ( xPtab[ i ]. bcp.sched. info .P∗(k+l ) != lcm) {
229             i=num_tasks;
230                 lcm = xPtab[num_tasks].bcp.sched. info .P∗(++k);
231           }
232     }
233   }
234
235   return OK;
236 }
```

Listing A.6: scheduler_r2.c

```
1  void t_blk(unsigned char nsem) {
2    queue_in(tmpPtab0,&QBlk[nsem]);
3  }
4
5
6  void oschedule(void) using 2{
7
8    if ( QExec. tail = get_hpriority_task (&QRdy)) {
9      queue_qout(QExec. tail,&QRdy);
10   }
11 }
```

Listing A.7: save_context.a51

```
1  NAME SAVE_CONTEXT
2
3  $include ( constants .a51)
4
5    ?PR?_save_context?SAVE_CONTEXT    SEGMENT CODE
6      PUBLIC _save_context
7      RSEG ?PR?_save_context?SAVE_CONTEXT
8      USING 2
9
10   _save_context:
```

```
11
12    mov acc,r1
13    mov r7,acc
14
15    mov acc,r5
16    mov r1,acc
17
18    push ip       ; ip
19    push ie       ; ie
20
21    mov a,@r1
22    push acc      ; psw
23    dec r1
24    mov a,@r1
25    push acc      ; dpl
26    dec r1
27    mov a,@r1
28    push acc      ; dph
29    dec r1
30    mov a,@r1
31    push acc      ; b
32    dec r1
33    mov a,@r1
34    push acc      ; acc
35    dec r1
36
37    mov a,r1
38     clr cy
39    subb a,#3
40
41    mov r1,a
42    mov a,@r1
43    mov r5,a      ; salvar PC -> R5
44    dec r1
45    mov a,@r1
46    mov r6,a      ; salvar PC -> R6
47
48    dec r1
49
50    mov a,r1
51    mov r4,a      ; salvar direccio pila -> R4
52
53
54  ; el punter DPTR apunta al principi de la taula BCP
55  ; de la memoria externa
56
57  ; passar cada un dels registres seguint els offsets
58  ; de les constats definides XUBCP_*. Recorrent
59  ; les posicions de memoria a traves del DPTR
60
61  ; registres visibles
62
63    mov r0,#XUBCP_VREGISTER
64     call XMEM_POSITION
```

```
65
66    mov a,0x00
67    movx @dptr,a
68    inc  dptr
69    mov a,0x01
70    movx @dptr,a
71    inc  dptr
72    mov a,0x02
73    movx @dptr,a
74    inc  dptr
75    mov a,0x03
76    movx @dptr,a
77    inc  dptr
78    mov a,0x04
79    movx @dptr,a
80    inc  dptr
81    mov a,0x05
82    movx @dptr,a
83    inc  dptr
84    mov a,0x06
85    movx @dptr,a
86    inc  dptr
87    mov a,0x07
88    movx @dptr,a
89    inc  dptr
90    pop acc ; recupera el acc salvat a pila
91    movx @dptr,a
92    inc  dptr
93    pop acc
94    movx @dptr,a
95
96  ;  registres  d'execucio
97
98    mov r0,#XUBCP_MREGISTER
99     call  XMEM_POSITION
100
101   mov acc,r5
102   movx @dptr,a
103   inc  dptr
104   mov acc,r6
105   movx @dptr,a
106
107    ; informacio de memoria
108
109   mov r0,#XUBCP_MEM
110    call  XMEM_POSITION
111
112   pop acc ; recupera el dph salvat a pila
113   movx @dptr,a
114   inc  dptr
115   pop acc ; recupera el dpl salvat a pila
116   movx @dptr,a
117
118 ; banderes d' estat
```

```
119
120     mov r0,#XUBCP_FLAGS
121     call  XMEM_POSITION
122
123     pop acc    ; recupera el PSW salvat a pila
124   ;   modificar PSW pq tingui en compte que el banc de registres correcte
125   ; PSW
126   ; | CY|AC|F0|RS1|RS0|OV|-|P|
127   ;     1   1   1   0     0     1 1 1    -> E7
128     movx @dptr,a
129
130   ; informacio d' estat
131
132     mov r0,#XUBCP_ISTATUS
133     call  XMEM_POSITION
134
135     pop acc
136     movx @dptr,a
137     inc dptr
138     pop acc
139     movx @dptr,a
140
141
142   ; pila
143
144     mov r0,#XUBCP_STACK
145     call  XMEM_POSITION
146
147     mov acc,r4
148     clr cy
149     subb a,#2
150     movx @dptr,a
151
152   ; salvar la pila
153
154     call  XMEM_STACK_POSITION
155
156   ; obtenir la mida de la pila
157   ; i guardar-la a primera posicio
158   ; de memoria externa de la zona
159   ; de la pila
160
161     mov a,r4
162     clr cy
163     subb a,#2
164     clr cy
165     subb a,#IUSTACK_BASE
166     movx @dptr,a
167
168     jz END_RESTORE_STACK
169   ; recuperar el stack pointer
170
171     mov a,r4
172   ; situar al principi del que cal gravar
```

```
173
174     mov r1,a
175     mov r0,#IUSTACK_BASE+3
176
177   ; bucles que recorre la zona de pila salvant
178   ; cada byte del stack a la memoria externa
179
180   SAVE_STACK:
181     inc dptr
182     mov a,@r0
183     movx @dptr,a
184
185     inc r0
186     dec r1
187
188   ; compara el valor del punter de stack
189   ; al direccio base de la pila d'usuari
190   ; a memoria interna
191
192     cjne r1,#IUSTACK_BASE+2,SAVE_STACK
193   END_RESTORE_STACK:
194
195   ; procedure per fer l'operacio de sumar l'offset
196   ; í especfic del registre i mantenir la consistencia
197   ; del DPTR
198
199     RET
200
201   XMEM_POSITION:
202
203   ; a partir de la direccio del proces a salvar
204   ; de la taula de processos, apuntar el DPTR
205   ; l'çadrea base de la seva zona de memoria externa
206   ; #PTAB_LXIMAGE fa referencia a l'offset dins
207   ; la taula de processos de la memoria interna
208
209     mov a,r7
210     add a,#PTAB_LXIMAGE
211     mov r1,a
212     mov dph,@r1
213     inc r1
214     mov dpl,@r1
215
216   ; gravar cada un dels registres a la corresponent
217   ; posicio de memoria externa. En funcio de la taula
218   ; descrita al document process_xdata_bcp.txt
219
220     mov a,dpl
221     add a,#XUBCP_OFFSET
222     mov dpl,a
223      clr a
224     addc a,dph
225     mov dph,a
226
```

```
227    mov a,dpl
228    add a,r0
229    mov dpl,a
230     clr a
231    addc a,dph
232    mov dph,a
233
234  RET
235
236
237  XMEM_STACK_POSITION:
238
239  ; a partir de la direccio del proces a salvar
240  ; de la taula de processos, apuntar el DPTR
241  ; l'çadrea␣base␣de␣la␣seva␣zona␣de␣memoria␣externa
242  ; #PTAB_LXIMAGE fa referencia a l'offset␣dins
243  ; la taula de processos de la memoria interna
244
245    mov a,r7
246    add a,#PTAB_LXIMAGE
247    mov r1,a
248    mov dph,@r1
249     inc r1
250    mov dpl,@r1
251
252  ; gravar la pila corresponent
253  ; posicio de memoria externa. En funcio de la taula
254  ; descrita al document process_mem.txt
255
256    mov a,dpl
257    add a,#XUSTACK_OFFSET
258    mov dpl,a
259     clr a
260    addc a,dph
261    mov dph,a
262
263  ?C0001:
264  RET
265
266  END
```

Listing A.8: restore_context.a51

```
1   NAME RESTORE_CONTEXT
2
3
4   $include ( constants.a51)
5
6     ?PR?_restore_context?RESTORE_CONTEXT SEGMENT CODE
7     PUBLIC _restore_context
8     RSEG ?PR?_restore_context?RESTORE_CONTEXT
9     USING 2
10
11     _restore_context :
12
```

```
13    mov a,r1
14    mov r7,a
15
16     call  XMEM_STACK_POSITION
17    movx a,@dptr
18    mov r1,a
19    mov r0,#IUSTACK_BASE
20
21    inc  r0
22    mov @r0,SPSCHED_IDLEL
23    inc  r0
24    mov @r0,SPSCHED_IDLEH
25
26  ;  passar  cada  byte  de  memoria externa  a  pila
27
28    jz  END_RESTORE_STACK
29
30  RESTORE_STACK:
31    inc  dptr
32    inc  r0
33    movx a,@dptr
34    mov @r0,a
35    djnz  r1 , RESTORE_STACK
36  END_RESTORE_STACK:
37
38    mov a,r0
39    mov r1,a
40    inc  r1
41
42    mov r2,#XUBCP_MREGISTER
43     call  XMEM_POSITION
44    movx a,@dptr
45    mov r3,a
46    inc  dptr
47    movx a,@dptr
48    mov @r1,a
49    inc  r1
50    mov a,r3
51    mov @r1,a
52
53  ;  situar   el  ACC i el  B sobre  la   pila
54
55    mov r2,#XUBCP_ABREGISTER
56     call  XMEM_POSITION
57    inc  r1
58    movx a,@dptr
59    mov @r1,a
60    inc  r1
61    inc  dptr
62    movx a,@dptr
63    mov @r1,a
64
65  ;  situar   DPH i DPL a sobre  la   pila
66    mov r2,#XUBCP_MEM
```

```
67      call  XMEM_POSITION
68      inc  r1
69      movx a,@dptr
70      mov @r1,a
71      inc  r1
72      inc  dptr
73      movx a,@dptr
74      mov @r1,a
75
76    ;  situar  PSW a sobre la  pila
77      mov r2,#XUBCP_FLAGS
78      call  XMEM_POSITION
79      inc  r1
80      movx a,@dptr
81      mov @r1,a
82
83    ;  informacio  d' estat
84
85      mov r2,#XUBCP_ISTATUS
86      call  XMEM_POSITION
87
88      movx a,@dptr
89      mov ip,a
90      inc  dptr
91      movx a,@dptr
92      mov ie,a
93
94    ;  restaurar  les  dades  de  la  pila
95
96      mov r2,#XUBCP_STACK
97      call  XMEM_POSITION
98
99      movx a,@dptr
100     cjne  a,#0x80,END_RESTORE_SP
101     add a,#0x07
102   END_RESTORE_SP:
103     add a,#0x02
104     mov sp,a
105
106     mov r2,#XUBCP_VREGISTER
107     call  XMEM_POSITION
108
109   ;  restaurar  tots  els  registres  Rn
110
111     movx a,@dptr
112     mov 0x00,a
113     inc  dptr
114     movx a,@dptr
115     mov 0x01,a
116     inc  dptr
117     movx a,@dptr
118     mov 0x02,a
119     inc  dptr
120     movx a,@dptr
```

```
121     mov 0x03,a
122     inc  dptr
123     movx a,@dptr
124     mov 0x04,a
125     inc  dptr
126     movx a,@dptr
127     mov 0x05,a
128     inc  dptr
129     movx a,@dptr
130     mov 0x06,a
131     inc  dptr
132     movx a,@dptr
133     mov 0x07,a
134
135   ; restaurar  els  valors  de  DPTR i PSW
136   ; emmagatzemats a pila  per ü qestions  de
137   ;  consistencia  de  les  dades
138
139     pop psw
140     pop dpl
141     pop dph
142     pop b
143     pop acc
144
145     setb  ET0
146     setb  TR0
147     setb  EA
148
149   RETI
150
151   XMEM_POSITION:
152
153   ; a  partir  de  la  direccio  del  proces a  restaurar
154   ; de  la  taula  de  processos , apuntar  el  DPTR
155   ; l'çadrea␣base␣de␣la␣seva␣zona␣de␣memoria␣externa
156   ; #PTAB_LXIMAGE fa referencia a l'offset␣dins
157   ;  la  taula  de  processos  de  la  memoria interna
158
159     mov a,r7
160     add a,#PTAB_LXIMAGE
161     mov r0,a
162     mov dph,@r0
163     inc  r0
164     mov dpl,@r0
165
166   ;  restaurar  cada un dels  registres  a  la  corresponent
167   ;  posicio  de  memoria externa . En funcio  de  la  taula
168   ;  descrita  al  document process_xdata_bcp. txt
169
170     mov a,dpl
171     add a,#XUBCP_OFFSET
172     mov dpl,a
173     clr  a
174     addc a , dph
```

```
175    mov dph,a
176
177    mov a,dpl
178    add a,r2
179    mov dpl,a
180     clr a
181    addc a,dph
182    mov dph,a
183
184  RET
185
186
187  XMEM_STACK_POSITION:
188
189  ; a partir de la direccio del proces a restaurar
190  ; de la taula de processos, apuntar el DPTR
191  ; l'çadrea␣base␣de␣la␣seva␣zona␣de␣memoria␣externa
192  ; #PTAB_LXIMAGE fa referencia a l'offset␣dins
193  ; la taula de processos de la memoria interna
194
195    mov a,r7
196    add a,#PTAB_LXIMAGE
197    mov r1,a
198    mov dph,@r1
199    inc r1
200    mov dpl,@r1
201
202  ; restaurar la pila corresponent
203  ; posicio de memoria externa. En funcio de la taula
204  ; descrita al document process_mem.txt
205
206    mov a,dpl
207    add a,#XUSTACK_OFFSET
208    mov dpl,a
209     clr a
210    addc a,dph
211    mov dph,a
212
213  RET
214
215  END
```

Listing A.9: save_idle.a51

```
1   NAME SAVE_IDLE
2
3   $include (constants.a51)
4
5     ?PR?_save_idle?SAVE_IDLE     SEGMENT CODE
6       PUBLIC _save_idle
7       RSEG ?PR?_save_idle?SAVE_IDLE
8       USING 2
9
10    _save_idle:
11
```

```
12    mov a,r7
13    mov r1,a
14    mov a,@r1
15    mov SPSCHED_IDLEL,a
16    inc  r1
17    mov a,@r1
18    mov SPSCHED_IDLEH,a
19
20  ?C0001:
21  RET
22
23  END
```

Listing A.10: restore_idle.a51

```
 1  NAME RESTORE_IDLE
 2
 3  $include  ( constants .a51)
 4
 5    ?PR? _restore_idle ?RESTORE_IDLE       SEGMENT CODE
 6       PUBLIC _restore_idle
 7       RSEG ?PR?_restore_idle?RESTORE_IDLE
 8       USING 2
 9
10     _restore_idle :
11
12    mov a,r7
13    mov r1,a
14    mov a,SPSCHED_IDLEL
15    mov @r1,a
16    inc  r1
17    mov a,SPSCHED_IDLEH
18    mov @r1,a
19    mov a,r1
20    mov sp,a
21
22    anl  TMOD,#0xF0
23    orl  TMOD,#0x01
24    mov TL0,#0xE0
25    mov TH0,#0xFF
26    setb  ET0
27    setb  TR0
28    setb  EA
29
30  ?C0001:
31  RETI
32
33  END
```

Listing A.11: timer.c

```
1
2  static  void  timer0_isr  (void)  interrupt  1  using  2 {
3
4    EA = 0;
```

```
5     TR0 = 0;
6
7     sp_user = SP;
8     SP = IOSTACK_BASE;
9
10    if (TExec) {
11       save_timer_context(TExec,sp_user);
12       queue_in(TExec,&QRdy);
13    } else {
14       save_idle(sp_user-6);
15    }
16
17    if (time_counter == lcm) time_counter = 0;
18
19    for (i=0;i<=num_tasks;i++) {
20       xp = Ptab[i].pximage;
21
22       if ((( time_counter%xp->bcp.sched.info.P) == 0) && (inside_queue(Ptab[i].PID,&QDone))) {
23          reset_task(xp);
24          queue_qout(&Ptab[i],&QDone);
25                queue_in(&Ptab[i],&QRdy);
26       }
27    }
28
29    QExec.tail=NULL;
30    while (QExec.tail == NULL) {
31       oschedule();
32       xp = QExec.tail->pximage;
33    }
34    TExec = QExec.tail;
35
36    xp->bcp.sched.info.t++;
37    time_counter++;
38
39       restore_timer_context(TExec);
40 }
41
42
43
44 void init_timer() {
45
46    EA = 0;
47    TR0 = 0;
48    TMOD &= ~0x0F;
49    TMOD |= 0x01;
50    TL0 = 0xC0;
51    TH0 = 0xFF;
52    ET0 = 1;
53    TR0 = 1;
54    EA = 1;
55
56 }
```

Listing A.12: sem.c

```
1
2  void  init_sems (void) {
3     for  ( i=0; i<num_sems;i++){
4       sem[i ]. v=0;
5
6       for  ( j=0;j<=num_tasks;j++){
7         xPtab[ j ]. bcp. resources . sem.sem_flag[ i ]=1;
8
9         if  ( xPtab[ j ]. bcp. resources . sem.sem[i] && ((xPtab[ j ]. bcp.sched. info . P  \\
10        < sem_ceil[ i   ]) || (  sem_ceil[ i  ] == 0))){
11          sem_ceil[ i ] =  xPtab[ j ]. bcp.sched. info . P;
12          sem[i ]. ceil  =  xPtab[ j ]. bcp.sched. info . P;
13        }
14      }
15    }
16 }
17
18
19 void  tsl (tsemreg  xdata  ∗semreg,tschedsem  xdata  ∗lock ) {
20  //      EA = 0;
21       ∗semreg = lock−>v;
22       lock−>v = 1;
23  //      EA = 1;
24 }
25
26
27 void  sem_up(tsemreg xdata ∗semreg , unsigned char nsem) using  2 {
28
29       EA = 0;
30       SP =  save_registers  ();
31       sp_user  =  SP;
32       SP = IOSTACK_BASE;
33       save_sem_context(TExec,sp_user );
34  //      EA = 1;
35
36       semreg =  get_semregister  ();
37       nsem = get_nsems();
38
39        tsl (semreg,&sem[nsem]);
40
41             if (∗semreg  == 1) {
42
43               tmpPtab0 = TExec;
44               t_blk (nsem);
45
46  //           EA = 0;
47               oschedule ();
48               xp  =  QExec. tail −>pximage;
49               TExec = QExec. tail ;
50                restore_context  (TExec);
51
52  //           EA = 1;
53            }
```

```
54        sem[nsem].pid=TExec->PID;
55        xp->bcp.sched.priority = sem[nsem].ceil;
56        SP = sp_user-7;
57        EA = 1;
58  }
59
60
61
62  void sem_down(tsemreg xdata *semreg, unsigned char nsem) using 2{
63
64    EA = 0;
65
66    SP = save_registers ();
67    sp_user = SP;
68    SP = IOSTACK_BASE;
69
70    semreg = get_semregister ();
71    nsem = get_nsems();
72
73    save_context(TExec,sp_user);
74    *semreg=1;
75
76    xp = QExec.tail->pximage;
77    xp->bcp.sched.priority = xp->bcp.sched.info.P;
78
79    tmpPtab0 = QExec.tail;
80    queue_qout(QExec.tail,&QExec);
81    queue_in(tmpPtab0,&QRdy);
82    QExec.tail = NULL;
83
84    if ((tmpPtab0 = get_hpriority_task (&QBlk[nsem])) != NULL) {
85      queue_qout(tmpPtab0,&QBlk[nsem]);
86      queue_in(tmpPtab0,&QRdy);
87    }
88
89    if (QBlk[nsem].head == NULL) {
90      sem[nsem].v=0;
91      sem[nsem].pid=0;
92    }
93    oschedule();
94    xp = QExec.tail->pximage;
95    TExec = QExec.tail;
96     restore_context (TExec);
97
98    EA = 1;
99
100 }
```

Listing A.13: serie.c

```
1  static void serial_isr (void) interrupt 4 using 2 {
2
3      unsigned char STAT_RI;
4
5          STAT_RI=RI;
```

```
6
7           switch(STAT_RI) {
8
9             case 0:
10                  if ( serial . o_buffer .c == 0) {
11                    TI=0;
12                    break;
13                  }
14                  SBUF = serial . o_buffer .b[ serial . o_buffer .head];
15                   serial . o_buffer .c−−;
16                   serial . o_buffer .head++;
17                  if ( serial . o_buffer .head == MAX_SERIAL_BUFFER_SIZE) {
18                     serial . o_buffer .head = 0;
19                  }
20                  TI = 1;
21                  break;
22
23            case 1:
24               serial . i_buffer .b[ serial . i_buffer . tail ]=SBUF;
25                    serial . i_buffer .c++;
26                    serial . i_buffer . tail ++;
27                  if ( serial . i_buffer . tail == MAX_SERIAL_BUFFER_SIZE) {
28                     serial . i_buffer . tail = 0;
29                  }
30                  RI = 0;
31            break;
32     }
33 }
34
35 void write_serial (char xdata ∗info , unsigned char count) {
36
37   unsigned char ib;
38   struct tmsg msg;
39
40   msg.ID = xp−>bcp.ident.pid;
41   msg.nb = count;
42   msg.type = TX_MSG;
43
44   for ( ib=0;ib<=count;ib++){
45     msg.tx_m[ib] = ∗info;
46          info++;
47   }
48
49   sem_up(&(xp−>bcp.resources.sem.sem_flag[1]),1);
50   mqueue_in(&msg,&QmSerial);
51   sem_down(&(xp−>bcp.resources.sem.sem_flag[1]),1);
52 }
53
54
55
56 void read_serial (char xdata ∗info , unsigned char count) {
57
58   struct tmsg msg;
59
```

```
60    msg.ID = xp->bcp.ident.pid;
61    msg.nb = count;
62    msg.type = RX_MSG;
63    msg.rx_m = info;
64
65    sem_up(&(xp->bcp.resources.sem.sem_flag[1]),1);
66    mqueue_in(&msg,&QmSerial);
67    sem_down(&(xp->bcp.resources.sem.sem_flag[1]),1);
68  }
69
70
71  void   init_serial  (unsigned char mode, unsigned short  tr_speed ) {
72
73    ES=0;
74
75     serial .mode = mode;
76
77     serial . i_buffer .head = NULL;
78     serial . i_buffer . tail  = NULL;
79     serial . i_buffer .c = NULL;
80
81     serial . o_buffer .head = NULL;
82     serial . o_buffer . tail  = NULL;
83     serial . o_buffer .c = NULL;
84
85    switch ( mode) {
86       case  MODE0:
87                SM0 =0;
88                SM1= 0;
89                break;
90       case  MODE1:
91                SM0 =0;
92                SM1= 1;
93                break;
94       case  MODE2:
95                SM0 =1;
96                SM1= 0;
97                break;
98       case  MODE3:
99                SM0 =1;
100               SM1= 1;
101               break;
102   }
103
104   TMOD = 32;
105   TH1 = tr_speed & 0xFF00;
106   TH1 = tr_speed & 0x00FF;
107
108   SBUF = NULL;
109   REN = 1;
110
111   ES = 1;
112   TR1 = 1;
113 }
```

Listing A.14: task_serial.c

```
1  void   init_t_task_serial  (void) {
2
3    xPtab[T_SERIAL_ID].bcp.ident.pid = T_SERIAL_ID + 1;
4    xPtab[T_SERIAL_ID].bcp.ident.ppid = T_SERIAL_ID + 1;
5    xPtab[T_SERIAL_ID].bcp.ident.uid = T_SERIAL_ID + 1;
6
7    xPtab[T_SERIAL_ID].bcp.sched.info.C = EXEC_TIME;
8    xPtab[T_SERIAL_ID].bcp.sched.info.P = MAX_PERIOD;
9    xPtab[T_SERIAL_ID].bcp.resources.sem.sem[1] = 10;
10
11    reset_task (&xPtab[T_SERIAL_ID]);
12
13   Ptab[T_SERIAL_ID].PID = T_SERIAL_ID + 1 ;
14   Ptab[T_SERIAL_ID].pximage = &xPtab[T_SERIAL_ID];
15
16   Ptab[T_SERIAL_ID].prev = NULL;
17   Ptab[T_SERIAL_ID].next = NULL;
18 }
19
20
21 void   t_task_serial  (void) {
22
23   unsigned char ib=0;
24   struct  tmsg *msg;
25
26    init_serial  (SERIAL_MODE, SERIAL_SPEED);
27
28   while (1) {
29
30      if  ( msg = &(QmSerial.m[QmSerial.head])) {
31        switch  ( msg->type) {
32
33                case  TX_MSG:
34                  while(msg->nb != 0) {
35                     serial . o_buffer . b[ serial . o_buffer . tail  ] =  msg->tx_m[ib];
36                     if  ( serial . o_buffer . tail  == MAX_SERIAL_BUFFER_SIZE) {
37                        serial . o_buffer . tail  =0;
38                     } else {
39                        serial . o_buffer . tail ++;
40                     }
41                     msg->nb--;
42                      serial . o_buffer . c++;
43                     ib++;
44                       }
45                   ib=0;
46                   sem_up(&(xp->bcp.resources.sem.sem_flag[1]),1);
47                   mqueue_qout(msg,&QmSerial);
48                   sem_down(&(xp->bcp.resources.sem.sem_flag[1]),1);
49
50                   TI=1;
51                   ES=1;
52                   break;
53
```

```
54              case RX_MSG:
55
56                while(( serial . i_buffer .head !=  serial . i_buffer . tail ) && (msg->nb >= 0)) {
57                  *(msg->rx_m) = serial. i_buffer .b[ serial . i_buffer .head];
58
59                  if  ( serial . i_buffer .head == MAX_SERIAL_BUFFER_SIZE) {
60                    serial . i_buffer .head =0;
61                  }else{
62                    serial . i_buffer .head++;
63                      }
64                  msg->nb--;
65                   serial . i_buffer .c--;
66                  msg->rx_m++;
67                }
68
69                if  (msg->nb !=0){
70                  sem_up(&(xp->bcp.resources.sem.sem_flag[1]),1);
71                  mqueue_qout(msg,&QmSerial);
72                  sem_up(&(xp->bcp.resources.sem.sem_flag[1]),1);
73                }
74                ES=1;
75                break;
76          }
77        }
78    }
79    end_t_task ();
80  }
```

Listing A.15: queues.c

```
1   void queue_in(struct tPtab *pptab, struct tQueue *queue) {
2
3     tmpPtab = NULL;
4     if (queue->tail) {
5       tmpPtab = queue->tail;
6       tmpPtab->next = pptab;
7     }
8     pptab->prev = tmpPtab;
9     pptab->next = NULL;
10
11    if  (!( queue->head && queue->tail)) {
12      queue->head = pptab;
13    }
14    queue->tail = pptab;
15  }
16
17
18
19  void queue_qout(struct tPtab *pptab, struct tQueue *queue) {
20
21    if (queue->head == pptab) {
22      queue->head = pptab->next;
23    }
24    if (queue->tail == pptab) {
25      queue->tail = pptab->prev;
```

```
26      }
27
28      tmpPtab = NULL;
29      if (pptab->prev) {
30          tmpPtab = pptab->prev;
31          tmpPtab->next = pptab->next;
32      }
33      if (pptab->next) {
34          tmpPtab2 = pptab->next;
35          tmpPtab2->prev = tmpPtab;
36      }
37
38      pptab->prev = NULL;
39      pptab->next = NULL;
40  }
41
42
43
44  char inside_queue(unsigned char PID,struct tQueue *queue) using 2{
45
46      tmpPtab2 = queue->head;
47      while (tmpPtab2 != NULL) {
48          if (tmpPtab2->PID == PID) {
49                  return OK;
50              }
51              tmpPtab2 = tmpPtab2->next;
52      }
53      return ERROR;
54  }
55
56
57
58  struct tPtab * get_hpriority_task (struct tQueue *queue) {
59
60      tmpPtab = queue->head;
61      tmpPtab2 = NULL;
62      sum = 0;
63
64      while (tmpPtab != NULL) {
65          xp = tmpPtab->pximage;
66
67          if ((sum == 0) || (xp->bcp.sched.priority < sum)) {
68                  sum = xp->bcp.sched.priority ;
69                  tmpPtab2 = tmpPtab;
70              }
71          tmpPtab = tmpPtab->next;
72      }
73      return tmpPtab2;
74  }
75
76
77
78  void mqueue_in(struct tmsg *mmsg,struct tmsg_queue *queue) {
79
```

```
80    if (queue->tail == MAX_MSGS) {
81      queue->tail = 0;
82    }
83    queue->m[queue->tail] = *mmsg;
84    queue->tail++;
85
86  }
87
88
89
90  void mqueue_qout(struct tmsg *mmsg,struct tmsg_queue *queue) {
91
92    if (queue->head == MAX_MSGS) {
93      queue->head = 0;
94    } else {
95      queue->head++;
96    }
97  }
```

Bibliografia

[1] Neil C. Audsley, Alan Burns, MF Richardson, and Andy J. Wellings. Hard real-time scheduling: The deadline-monothonic approach. Technical report, Department of Computer Science, University of York, -.

[2] Stuart Bennet. *Real-time computer control. An introduction*. Prentice Hall, 2ª edition, 1994.

[3] Alan Burns and Andy Wellings. *Real-time systems and programming languages*. Addison Wesley Longman, 2ª edition, 1997.

[4] TimeSys corporation. The concise handbook of real-time systems. Technical report, TimeSys Corporation, 2000.

[5] Ken Frazer. Scheduling algorithms for real-time operarting systems. -, 1997.

[6] Paul S. Heidmann. Rate montonic analysis. –.

[7] Hitex(UK). *C51 Primer. An Introduction To The Use Of The Keil C51 Compiler On The 8051 Family*. Hitex Development Tools. Hitex, 1996.

[8] Intel. External product specification for the mcs-51 object module format. Technical report, Intel Corporation, 1982.

[9] Andrew Jones. A rate-monotonic scheduler. 2001.

[10] David Kalinsky. Context switch. *Embedded Magazine*, 2001.

[11] Elektronitk keil gmbh. *8051 Utilities*. KEIL Software, 1ª edition, 2000.

[12] Elektronitk keil gmbh. *A51 Assembler. A251 Assembler*. KEIL Software, 1ª edition, 2000.

[13] Elektronitk keil gmbh. *C51 Compiler*. KEIL Software, 1ª edition, 2000.

[14] Elektronitk keil gmbh. *Getting Started and Creating Applications*. KEIL Software, 1ª edition, 2000.

[15] Jean Labrosse. *MicroCOSII*. R&D Books, 1ª edition, 1999.

[16] Phillip Laplante. *Real-time systems design and analysis. An Engineer's HandBook*. IEE Press, 2ª edition, 1997.

[17] John P. Lehoczky, Lui Sha, and Y. Ding. The rate monothonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time System Symposium, IEE*, 1989.

[18] CL Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery Vol. 20 No. 1*, 1973.

[19] Bernard Odant. *Microcontroladores 8051 y 8052*. Paraninfo, 1ª edition, 1997.

[20] Ismael Ripoll. Planificación en sistemas de tiempo real. –.

[21] Ismael Ripoll. *Planificación con prioridades dinámicas en sistemas de tiempo real crítico*. Tesis doctoral, Universidad Politécnica de Valencia, Departament de ingeniería de sistema, computadores y automática, 1996.

[22] Tom Schultz. *C and the 8051. Hardware, modular programming and multitasking*. Prentice Hall, 2ª edition, 1998.

[23] Lui Sha, Mark H. Klein, and John B. Goodenough. Rate monotonic analysis for real-time systems. Technical report, Software Engineering Institute, Carnegie Mellon University, 1991.

[24] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. In *Proceedings of the Real-Time System Symposium, IEE vol 39*, 1990.

[25] William Stallings. *Sistemas Operativos*. Prentice Hall, 2ª edition, 1997.

[26] William Stallings. *Organización y Arquitectura de Computadores*. Prentice Hall, 5ª edition, 2000.

[27] John A. Stankovic and Krithi Ramamritham. Scheduling algorithms and operating systems support for real-time systems. In *Proceedings of the Real-Time System Symposium, IEEE Vol. 82, No. 1*, 1994.

[28] Andrew Tanenbaum and Albert Woodhull. *Sistema Operativos. Diseño e implementación*. Prentice Hall. A Simon & Schuster Company, 2ª edition, 1998.

[29] Lonnie Vanzandt. Scheduling sporadic events. *Embedded Magazine*, 2002.

[30] Gabriel A. Wainer. Implementing real-time services in minix. Technical report, Departamento de Computación. Facultad de Ciencias Exactas y Naturales, –.