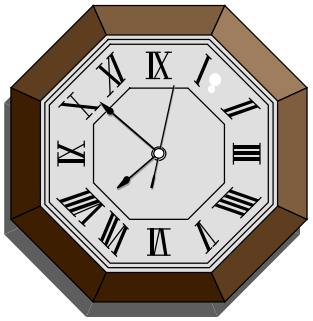
UPC Fundació Politècnica de Catalunya: la formació contínua
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Sistemes Informàtics en Temps Real



Ramon Sarrate Estruch

Departament d'Enginyeria de Sistemes,
Automàtica i Informàtica Industrial

1. Introducció als SITR
1.1. Objectius de l'assignatura

Desenvolupament d'aplicacions informàtiques de control i supervisió de processos industrials

- Identificar les *característiques* d'un sistema informàtic en temps real
- Comprendre com el *temps* intervé en aquest tipus d'aplicacions
- Saber estructurar l'aplicació en múltiples *seccions* concurrents de codi
- Veure diverses alternatives de *gestió* per aquestes seccions de codi

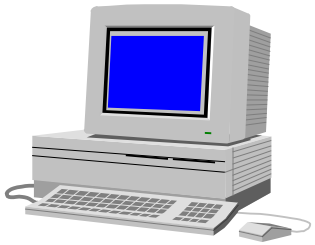
Més objectius

• Saber protegir adecuadament els *recursos* compartits per les diverses seccions de codi

• Practicar la programació d'aplicacions en temps real en el llenguatge *Ada 95*

• Conèixer tècniques d'anàlisi de *planificabilitat* per aplicacions en temps real

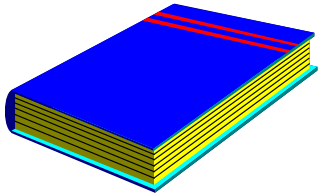
• Examinar les característiques de *xarxes* de comunicació en temps real



Pràctiques

Programació d'aplicacions de control en PC

- Maquinari: PC
- Programari: Windows NT + Ada 95



Referències

Sistemes en temps real

• A. Burns, A. Wellings, *Real-time systems and programming languages*, Addison-Wesley, 1997

• Ch. Bonnet, I. Demeure, *Introduction aux systèmes temps réel*, Hermes Science Publicactions, 1999

• Ph.A. Laplante, *Real-time systems design and analysis. An engineer's handbook*, IEEE Press, 1993

Sistemes concurrents

• A. Burns, G. Davies, *Concurrent programming*, Addison-Wesley, 1993

Més referències

Programació de co-rutines

• D.M. Auslander, *Mechatronics: a design and implementation methodology for real-time control software*

Programació en Ada 95

• J. Barnes, *Programming in Ada 95*, Addison-Wesley, 1996

• A. Burns, A. Wellings, *Concurrency in Ada*, Cambridge University Press, 1998

Programació en C

• D.M. Auslander, Ch.H. Tham, *Real-time software for control*, Prentice Hall, 1990

Referències a Internet

Sistemes en temps real

- *Real-Time Encyclopaedia* http://www.realtime-info.be/
- *IEEE-CS TC-RTS* http://cs-www.bu.edu/pub/ieee-rts/Home.html
- *RTOS* http://www.realtime-info.be/encyc/market/rtos/rtos.htm

Ada

- *Ada Home* http://www.adahome.com
- *Ada Power* http://www.adapower.com/index.html
- *Compilador GNAT* ftp://ftp.dit.upm.es/mirrors/cs.nyu.edu/pub/gnat/

Co-rutines i programació en temps real en C

- *Pàgina personal de D.M. Auslander http://www.me.berkeley.edu/faculty/auslander/*

Recursos de programació

- *Burks online* http://burks.brighton.ac.uk/

1.2. Definicions i conceptes

Què és un sistema en temps real (STR)?

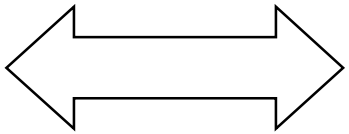
esdeveniments asíncrons externs i interns

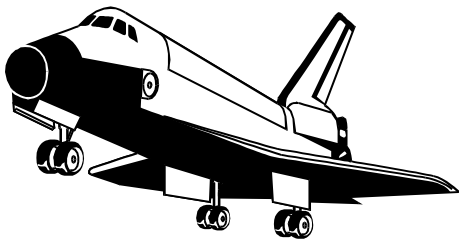
procés a controlar

temps resposta

Definició

Sistema que ha d'*interaccionar* amb l'*entorn* en intervals de *temps* especificables, finits i normalment breus. No només les accions empreses han de ser lògicament correctes sinó que s'han de produir en l'interval de temps especificat. En cas contrari perilla la seva missió.





Classificació de STR

STR crític

És imprescindible complir amb les especificacions temporals.
Exemples:control d'un avió, control d'una central nuclear.

STR acrític

Les especificacions temporals es poden violar ocasionalment.
Exemples: control de temperatura, control de nivell.

STR estricte

Sistema crític amb especificacions temporals molt curtes o breus.

Sistema interactiu (no és en temps real)

Sistema que no depèn d'especificacions temporals, o si ho fa són molt flexibles.
Un retard en les accions no resulta catastròfic.
Exemples: un editor de text, una aplicació de comptabilitat.

Concurrència

Programació seqüencial

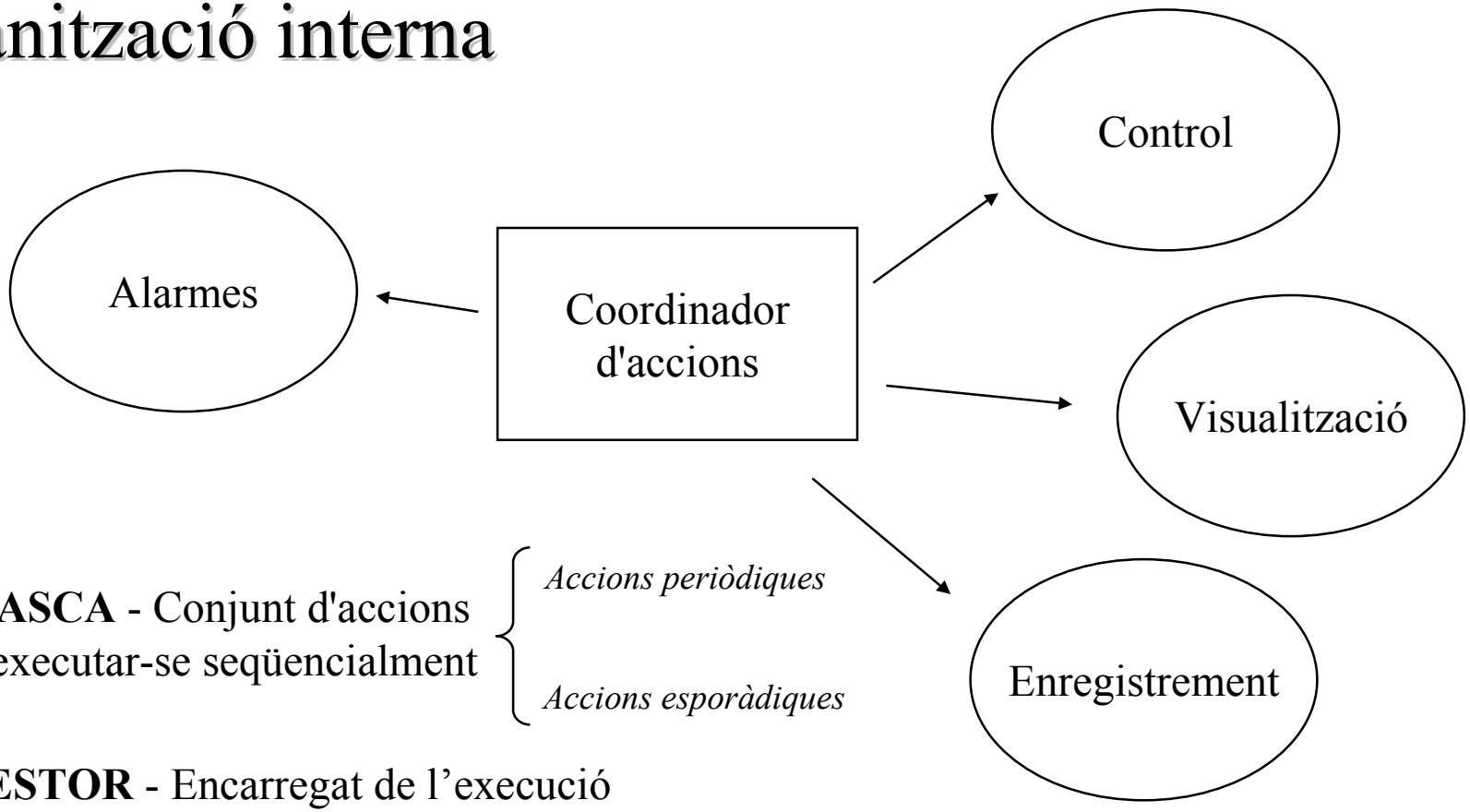
- El programa especifica textualment l'ordre d'execució de les instruccions
- Heretat de l'arquitectura Von Newmann
- No hi ha solapament en l'execució d'instruccions
- Imposa restriccions sobre una implementació lògica
- Exemple: una CPU programada en ensamblador

Programació concurrent

- L'ordre d'execució de les instruccions no queda especificat textualment
- És possible en arquitectures monoprocessador i multiprocessador
- El solapament d'intruccions està permès

1.3. Característiques d'un SITR

Organització interna



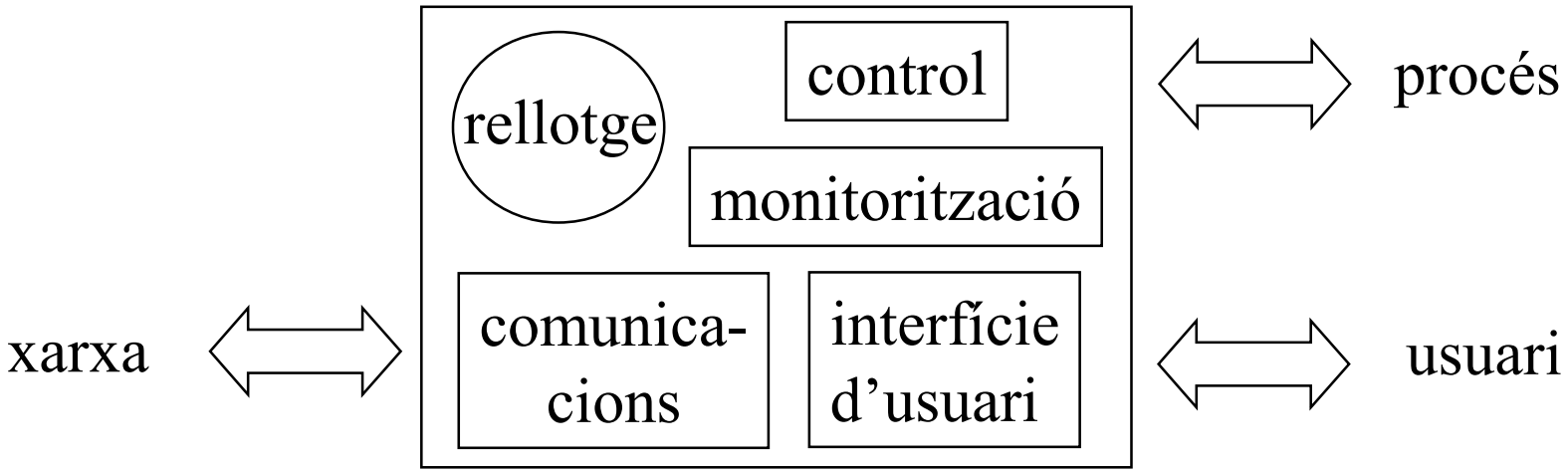
TASCA - Conjunt d'accions a executar-se seqüencialment

- *Accions periòdiques*
- *Accions esporàdiques*

GESTOR - Encarregat de l'execució concurrent de les tasques

Tipus d'esdeveniments

- Esdeveniments *síncrons*
- Esdeveniments *assíncrons*
  - *Periòdics* → control digital, visualització,enregistrament
  - *Esporàdics* → alarmes, interacció amb l'usuari



Sistemes multitasca

• Una tasca és un conjunt d'accions que s'executen seqüencialment. Normalment són bucles infinits

• Les tasques són concurrents entre elles

• En sistemes monoprocessador, el gestor se n'encarrega de l'execució virtualment paral·lela de les tasques. A nivell d'instrucció màquina l'execució és seqüencial

• En sistemes multiprocessadors o distribuïts l'execució és realment en paral·lel, fins i tot a nivell d'instrucció màquina

Més sobre sistemes multitasca

- La necessitat que les tasques interaccionin entre sí (per comunicar-se o sincronitzar-se) ocasiona noves problemàtiques

- Cal protegir recursos compartits per tal de garantir l'accés exclusiu

- Són sistemes no determinístics: executats repetidament davant les mateixes entrades produeixen resultats diferents

- Són sistemes difícilment testejables o depurables

Planificació de tasques

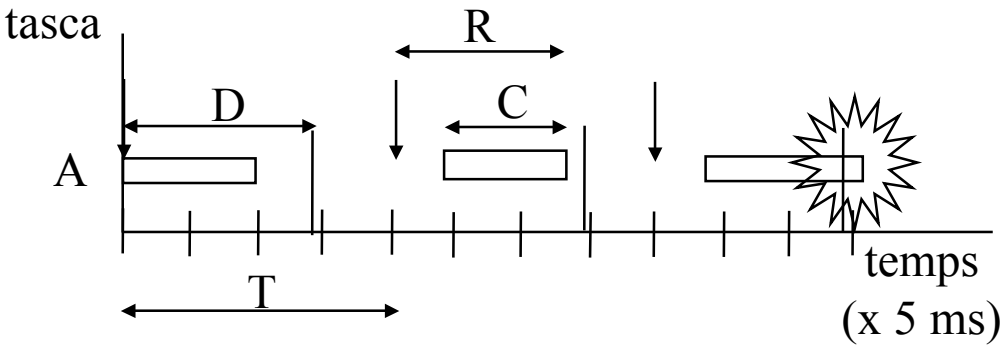
• Tot sistema en temps real ha de satisfer estrictament uns **requeriments temporals**

• Dissenyar un STR que compleixi aquests requeriments en temps real comporta dos aspectes:
  • Seleccionar un algorisme de planificació adequat
  • Disposar d'una metodologia d'anàlisi que permeti avaluar les prestacions en temps real que es poden esperar de la nostra aplicació

• L'**algorisme de planificació** especifica la forma de repartir a lo llarg del temps la/les CPU entre totes les tasques que composen una aplicació

Requeriments temporals

• Els requeriments temporals de STR s'especifiquen normalment en base als següents paràmetres:
  • el període, *T*
  • el termini, *D*
  • el temps de còmput, *C*
  • el temps de resposta, *R*



• Per tasques aperiòdiques, *T* representa el temps mínim entre dos activacions consecutives

• L'objectiu és aconseguir que totes les tasques s'executin *dins* el termini previst (cal que R ≤ D per totes les tasques)

Avaluació de prestacions en temps real

Existeixen dues tècniques d'anàlisi de *planificabilitat*:

- Avaluació gràfica sobre el diagrama temporal
  En aquest cas només cal estudiar què passa a partir de l'instant crític i durant un període de temps que ha de coincidir amb el període més llarg de totes les tasques involucrades

- Avaluació analítica del temps de resposta

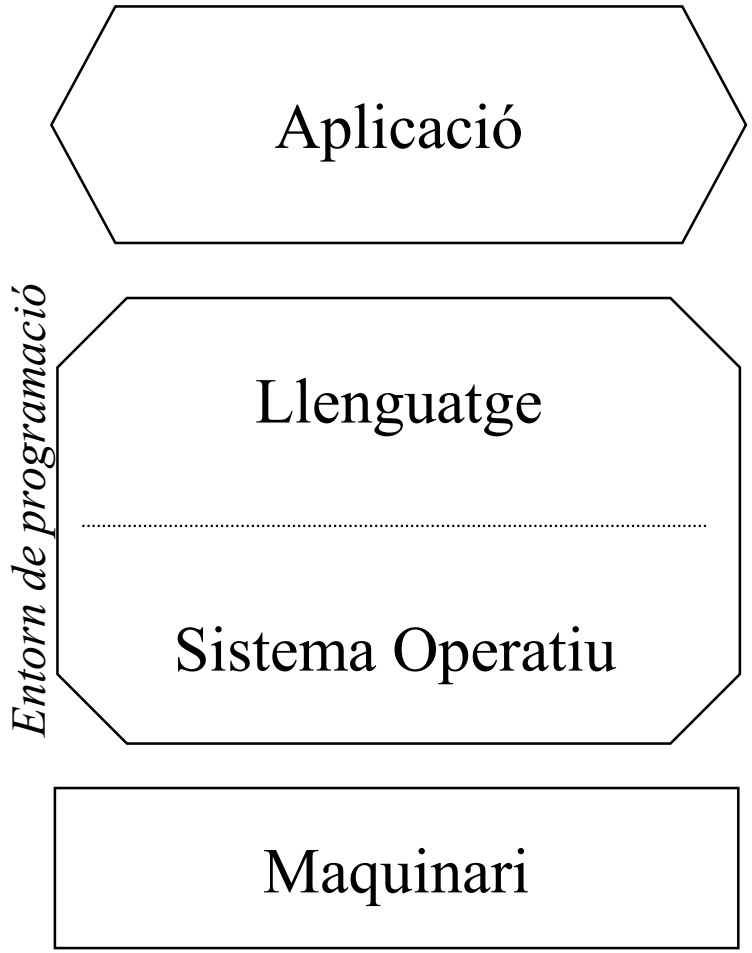
1.4. Programació de SITR: SO i llenguatges
Entorns de programació multitasca en temps real

A l'hora de programar aplicacions en temps real s'ha de poder:

- Accedir a dispositius de E/S a baix nivell
- Treballar amb interrupcions
- Mesurar i manipular el temps
- Organitzar l'aplicació en tasques d'execució concurrent
- Permetre escollir i parametritzar l'algorisme de planificació, preferentment basat en prioritats
- Disposar de mecanismes de comunicació i sincronisme eficients
- Disposar dels paràmetres temporals de l'entorn d'execució (e.g., canvis de contexte i servei d'interrupcions)

El poder tenir-ho més o menys tot dependrà del SO que s'utilitzi i/o del llenguatge de programació emprat.

Arquitectures de desenvolupament



Possibilitats:

- Llenguatge i SO no són en temps real
- Llenguatge en temps real però SO no
- SO en temps real però llenguatge no
- Llenguatge i SO en temps real
- Sistema empotrat (sense SO) amb llenguatge en temps real

Exemples d'entorns en temps real:

- Llenguatges: Ada, C, C++.
- SO: RTLinux, QNX, RTEMS, LynxOS, OS-9, RTX-Win NT.

El llenguatge de programació Ada 95 (1)

Història:

Fruit d'un concurs dins el DoD americà per tal d'unificar la programació en un sol llenguatge adequat a les necessitats.

Aplicacions:

Serveis financers, aviònica, aeronàutica, control de tràfic aeri, telecomunicacions, aparells mèdics, centrals elèctriques, ferrocarrils, astrofísica, satèl·lits i militars.

Característiques en temps real:

Estructuració en tasques
Gestió basada en prioritats
Mecanisme de comunicació i sincronisme: *rendezvous* i objectes protegits
Programació de retards no blocants i *timeouts*
Detecció de manca d'assoliment de terminis

El llenguatge de programació Ada 95 (2)

Altres característiques desitjables:

Fiabilitat: Comprovació d'errors en temps de compilació i en temps d'execució.

- Fort tipatge
- Es poden establir rangs de valors per a variables
- Cal declarar totes les variables
- Es detecten accessos a vectors incorrectes
- Excepcions del sistema i del programador

Programació estructurada: per a grans aplicacions i per la reutilització del codi.

- Organització en mòduls o *packages*
- Programació orientada a objectes: declaració de nous tipus de dades amb operacions restringides, herència i polimorfisme
- Aplicabilitat de disseny top-down i bottom-up.

Facilitat per a la programació:

- Vectors d'índex inicial no 0
- Valors per defecte a paràmetres de funcions i estructures
- Identificadors flexibles

2. Interacció amb el procés: programació de perifèrics
2.1. Tipus de perifèrics

Perifèric

Dispositiu de maquinari que permet a un sistema basat en microprocessador realitzar una sèrie d'accions no considerades de càlcul o de transferència amb memòria.

Exemples:

- Interfícies digitals d'entrada i sortida
- Interfícies analògiques d'entrada i sortida
- Interfícies de comunicacions
- Comptadors/temporitzadors
- Gestors d'interrupcions

2.2. Organització interna: registres de programació

Accions bàsiques amb perifèrics

• Configuració i comandament
Exemples: velocitat de transmissió (RS-232), número de canal (CAD), mode de funcionament (temporitzador), iniciar conversió (CAD)

• Estat
Exemples: fi de conversió (CAD), error en la recepció(RS-232)

• Transferència de dades
Exemples: dades adquirides (CAD), missatge rebut (RS-232)

Programació de perifèrics

Operacions → registres d'entrada/sortida

Registres d'Entrada/Sortida

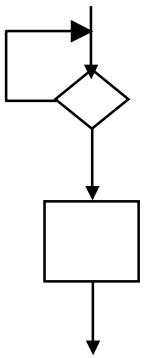
Segons com està configurat el sistema microprocessador, l'accés a aquests registres es pot fer mitjançant les mateixes instruccions que permeten accedir a registres de memòria o bé amb unes altres específiques. En el cas del PC, en C s'accedeix de forma diferent als registres de memòria (amb punters) que als d'E/S (amb inportb i outportb)

@ (adreça) base - normalment el conjunt de registres associats a un perifèric s'accedeixen de forma consecutiva a partir d'una adreça anomenada @ base. Aquesta adreça s'ha de configurar de forma que no hi hagi conflicte entre els diversos perifèrics de que disposem

2.3. Transferència de dades: enquesta i interrupció

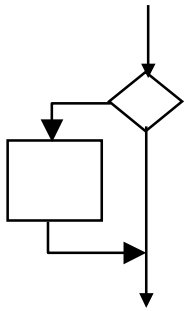
Perifèric ↔ CPU+memòria

- *Enquesta*: la CPU consulta al perifèric

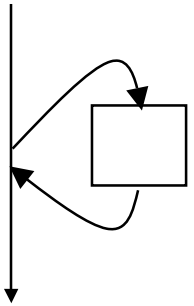


Activa: el perifèric és interrogat mitjançant un bucle fins que estigui llest. Comporta una CPU infrautilitzada

Passiva: el perifèric és interrogat periòdicament. Si no està llest no hi ha espera. Comporta un retard en la resposta



• *Interrupció*: el perifèric notifica a la CPU. És com una crida asíncrona a una subrutina. Comporta complexitat



Aspectes a considerar abans de treballar amb interrupcions

• Configurar el perifèric perquè utilitzi el canal adequat d'interrupció (hardware)

• Configurar el microprocessador associant la rutina de servei a l'interrupció (RSI) al canal d'interrupció corresponent (software)

Aspectes a considerar durant el servei d'una interrupció

• Abans de notificar-se la interrupció, la CPU finalitza la instrucció de codi màquina en execució

• Guardar temporalment l'estat dels diversos registres que s'utilitzen

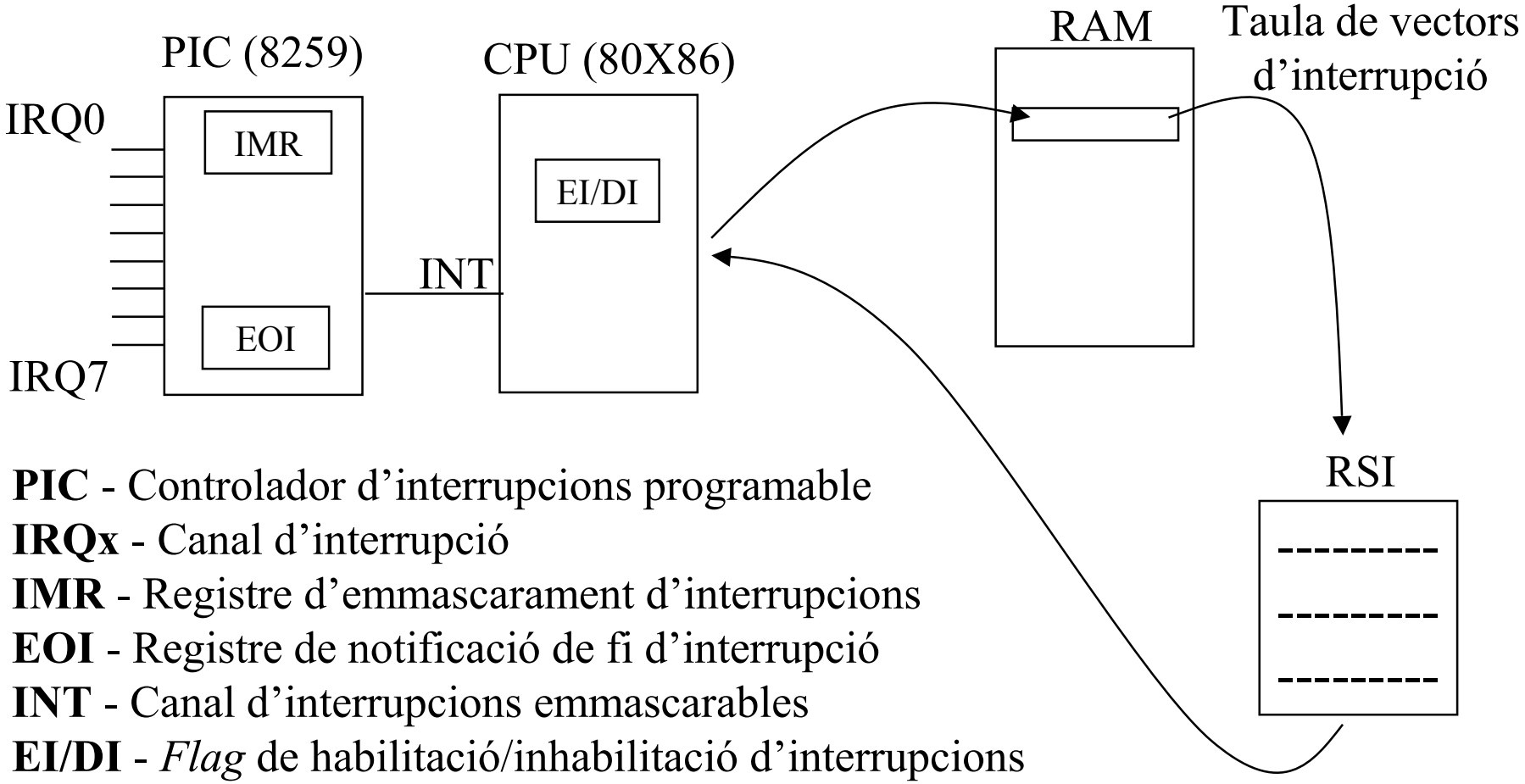
• Determinar quin perifèric ha generat la interrupció (si cal)

• Respondre adequada i breument

• Recuperar els registres

• Notificar la finalització del servei d'interrupció

Gestió de les interrupcions en el PC (maquinari)



PIC - Controlador d'interrupcions programable
IRQx - Canal d'interrupció
IMR - Registre d'emmascarament d'interrupcions
EOI - Registre de notificació de fi d'interrupció
INT - Canal d'interrupcions emmascarables
EI/DI - *Flag* de habilitació/inhabilitació d'interrupcions
RSI - Rutina de servei a la interrupció

Gestió de les interrupcions en el PC (programari)

Programa principal

• Guardar la configuració actual
• Associar el IRQ del perifèric amb la RSI corresponent
• Habilitar l'IRQ mitjançant l'IMR

Interrupcions actives

• Recuperar de la configuració anterior

Rutina de servei a la interrupció

Inhabilitar la INT de la CPU
Guardar registres

• Servir la interrupció
• Notificar al PIC d'EOI

Recuperar registres
Habilitar la INT de la CPU
Fi interrupció

3. El temps

3.1. Definicions i conceptes

La noció de temps

L'aplicació de control necessita coordinar la seva execució amb el 'temps' de l'entorn.

S'identifiquen les següents necessitats:

- Accés a un rellotge (calendari i monotònic) per fer-ne lectures
- Expressar retards
- Expressar *timeouts* per detectar esdeveniments no produïts
- Programar accions periòdiques
- Especificar terminis en l'execució de codi

Característiques del rellotge

Absolut. Ens interessa el temps respecte a una referència absoluta.

Relatiu. Ens interessa una diferència de temps.

Atributs del sistema de mesura

- *Precisió*: exactitud en la mesura del temps real
- *Granularitat*: mínima diferència de temps que es pot mesurar
- *Rang*: màxima diferència de temps que es pot mesurar

3.2. Sistemes de mesura de temps
Exemple d'un sistema existent en Ada 95

• *Packages*: *Calendar* i *Real_Time* (funció *Clock*, tipus *Time* i *Time_Span*)

• Retard relatiu (*delay*) i absolut (*delay until*)
  Comporten esperes no blocants
  S'assegura una espera del temps especificat com a mínim
  És la base per l'execució periòdica de codi

• Permet especificat *timeouts* en l'accés a recursos compartits i en la comunicació entre tasques (*select-or-delay* i *select-accept-or-delay*)

• Permet acotar el temps d'execució de codi (*select-delay-then abort*)

Concepció de sistemes de mesura de temps (1)

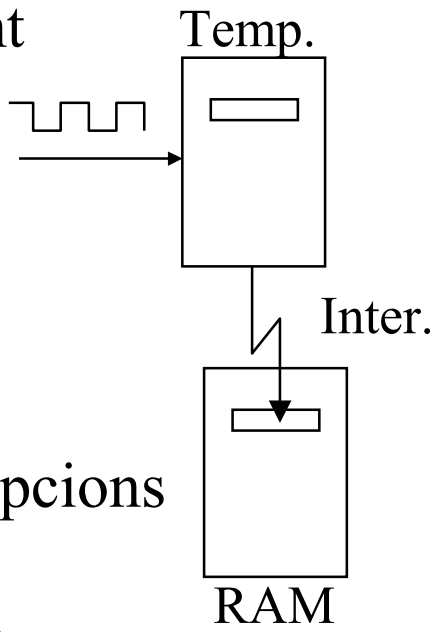
Exemple que no requereix maquinari addicional: per **calibratge**

- Un comptador s'incrementa regularment dins la nostra aplicació
- El temps s'obté multiplicant el valor del comptador per una constant que s'ha de calibrar prèviament
- Calibratge de la constant: durant un interval de temps prou gran es mesura, mitjançant una referència de temps fiable, el temps transcorregut i es divideix pel valor final del comptador

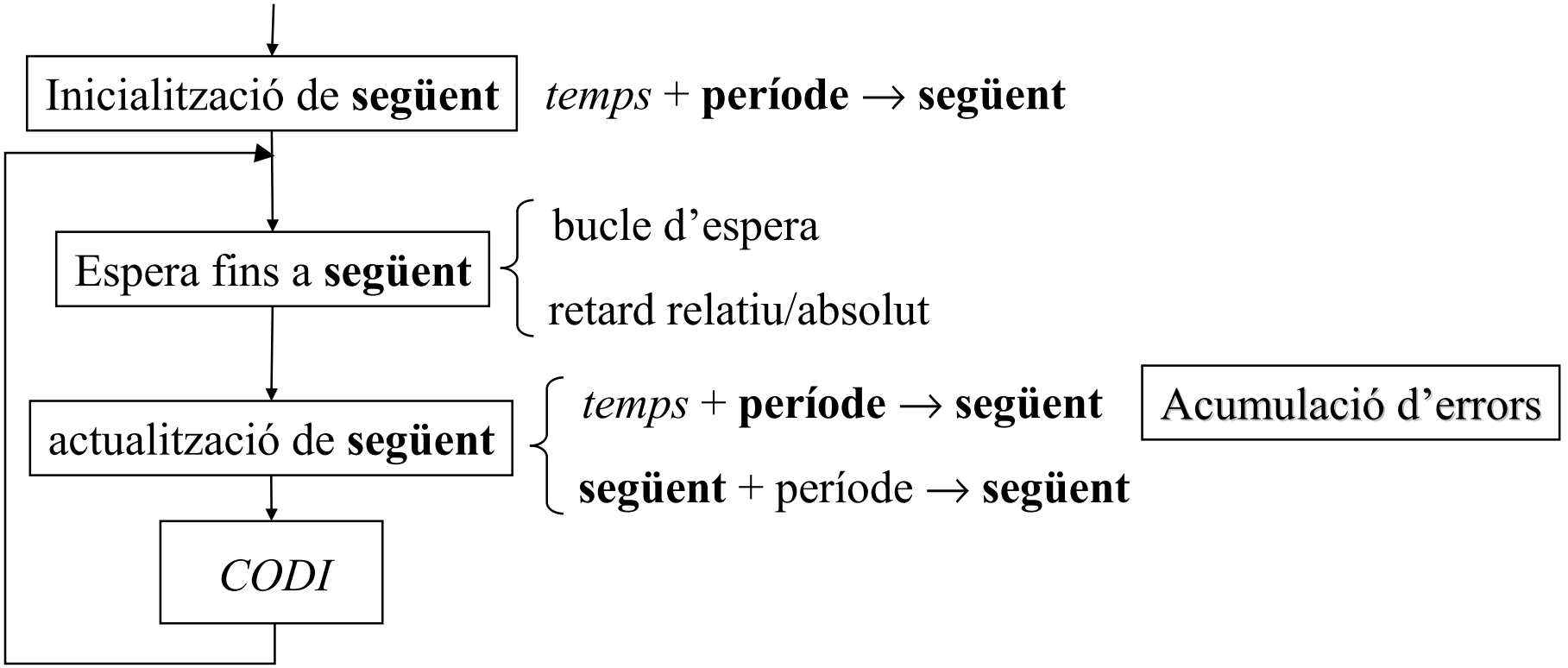
Concepció de sistemes de mesura de temps (2)

Exemple que requereix maquinari addicional: amb **temporitzador**

- Un temporitzador se n'encarrega d'anar comptabilitzant el pas del temps amb l'ajut d'un oscil·lador
- El comptador té un rang limitat pel nº de bits
- Cada vegada que arriba al màxim torna a començar automàticament el comptatge des de 0
- Aquest fet es senyalitza amb una interrupció
- Es manté una variable que comptabilitza el nº d'interrupcions
- El temps és proporcional al valor d'aquesta variable
- Per augmentar la granularitat es pot complementar amb la lectura parcial del temporitzador



Execució periòdica de codi



4. Sistemes multitasca
4.1. Introducció
4.1.1. Mecanismes d'invocació i de planificació

- Un sol processador és disputat per totes les tasques
- El gestor decideix quina tasca passa a tenir el control de la CPU
- Tipus de gestors segons el mecanisme d'invocació:
  - *Cooperatiu*: mentre la CPU està en possessió d'una tasca, el gestor no pot intervenir fins que la tasca finalitzi voluntàriament
  - *Apropiatiu*: en qualsevol moment el gestor és capaç de fer-se amb el control de la CPU, interrompent qualsevol tasca que s'estigui executant

• El gestor s'encarrega de oferir la sensació d'execució concurrent de les tasques

• Tipus de gestors segons el mecanisme de planificació:
  •*Arbitrari*: la següent tasca a executar-se s'escull a l'atzar, així com el temps d'assignació de CPU
  •*Time-slicing round robin*: s'assigna temps de CPU fix a totes les tasques de forma seqüencial
  •*Prioritari*: cada tasca té assignat un nivell de prioritat. En tot moment la tasca més prioritària és la que s'està executant

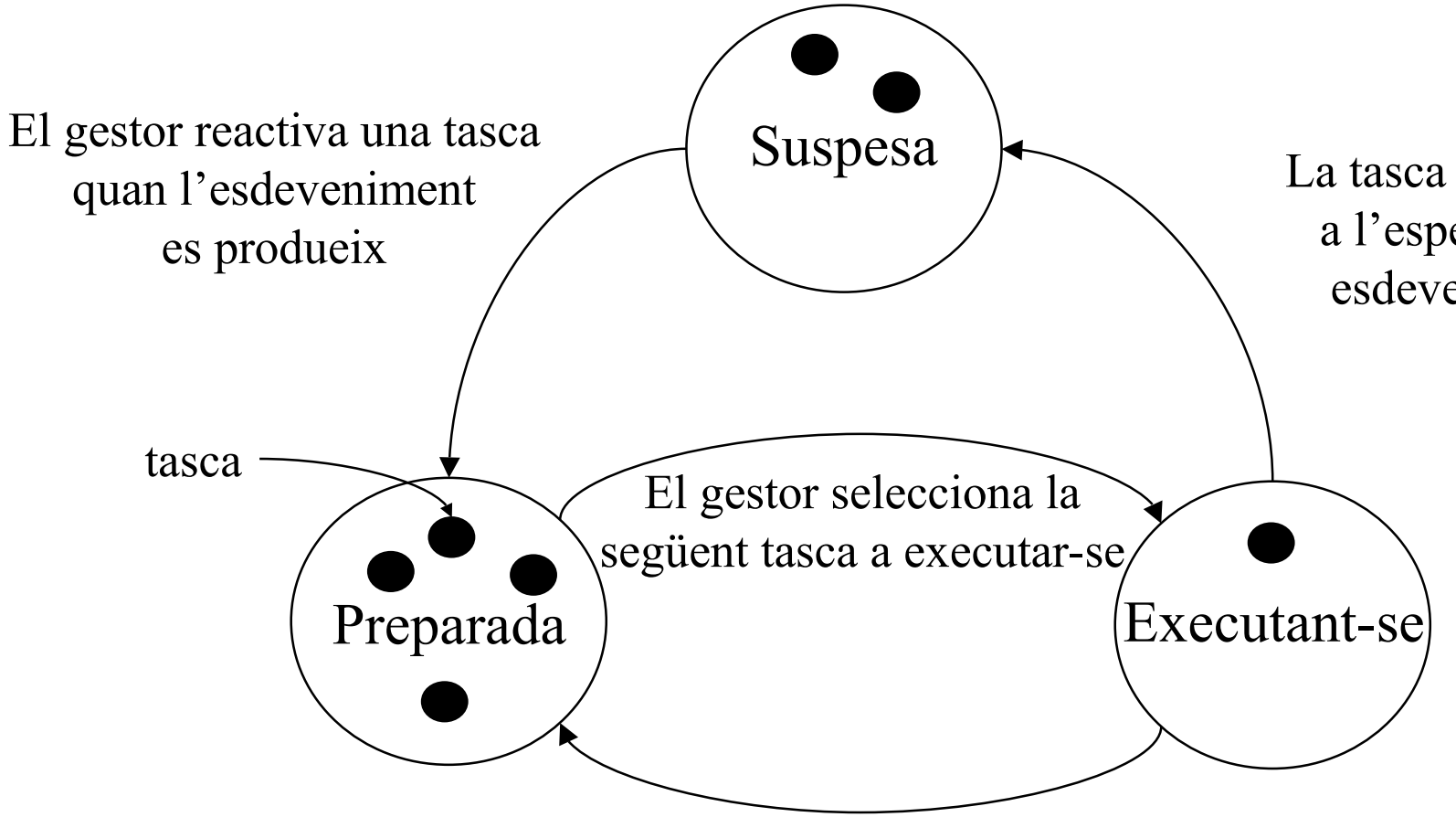
4.1.2. Cicle de vida d'una tasca

Observant el diagrama temporal d'assignació de CPU per a un sistema multitasca ens adonem del següent:

• En tot moment una sola tasca està *executant*-se.

• Algunes tasques estan *preparades* per executar-se; només esperen que el gestor els hi adjudiqui la CPU.

• Algunes tasques durant la seva execució, necessiten realitzar esperes per accedir a recursos compartits en ús o lents. Per tal d'optimitzar l'ús de la CPU, aquestes tasques es *suspenen*.



- El gestor interromp a la tasca en execució (gestor apropiatiu)
- La tasca en execució cedeix la CPU al gestor (gestor cooperatiu)

4.1.3. Interacció entre tasques

Segons el grau d'interacció, les tasques poden ser

- *Independents* - no hi ha interacció
  Ex: control de temperatura i de velocitat
- *Cooperatives* - interaccionen amb una finalitat comuna
  Ex: PID+PWM en el control de temperatura
- *Competitives* - interacionen disputant-se un recurs compartit
  Ex: targeta d'adquisició, memòria

Tipus d'interaccions:

- *Comunicació*: intercanvi d'informació entre tasques
- *Sincronisme*: coordinació en l'ordenació relativa d'accions entre tasques

Comunicació i sincronisme són dos conceptes molt lligats:

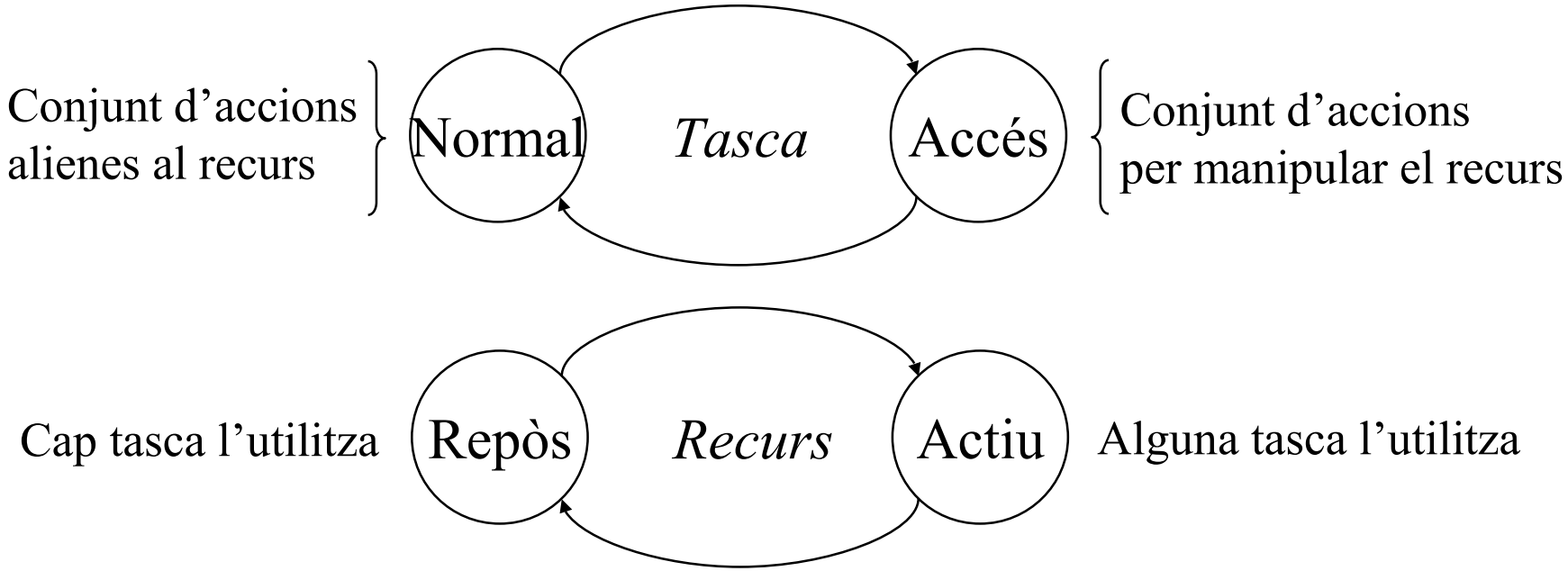
- Normalment les tasques que volen comunicar-se han de sincronitzar-se
- El sincronisme entre tasques es pot entendre com una comunicació sense contingut

Principalment existeixen dos mecanismes d'interacció:

- Mitjançant memòria compartida
- Mitjançant missatges

4.1.4. Protecció de recursos

- Per recurs s'entén memòria i qualsevol perifèric d'una CPU.
- Es pot considerar els següents diagrames d'estats d'accés:



- L'accés simultani de vàries tasques a un mateix recurs compartit pot comportar problemes d'integritat. Aquesta situació depèn del gestor.

• La protecció de recursos es garanteix accedint-hi de forma exclusiva
• **Secció crítica** - codi que s'ha d'accedir de forma exclusiva
• **Exclusió mútua** - sincronisme necessari per protegir un recurs compartit
• Una tasca accedint a un recurs compartit pot ser interrompuda pel gestor, però s'ha de garantir que la tasca entrant no accedeixi també al mateix recurs
• Amb un gestor *cooperatiu* el programador pot controlar fàcilment l'accés exclusiu ja que el gestor intervé en certs punts coneguts.
• Amb un gestor *apropiatiu* el control és més complexe ja que el gestor pot interrompre la tasca en possessió de CPU en qualsevol moment i de forma imprevisible.

Exemple: Integritat en l'accés a memòria compartida

Els requeriments d'integritat han de garantir-se en tres situacions:

- Accés a variables simples
- Accés consistent a variables compostes
- Execució consistent de codi

Accés a variables simples

Tasca A -
04FFh → V

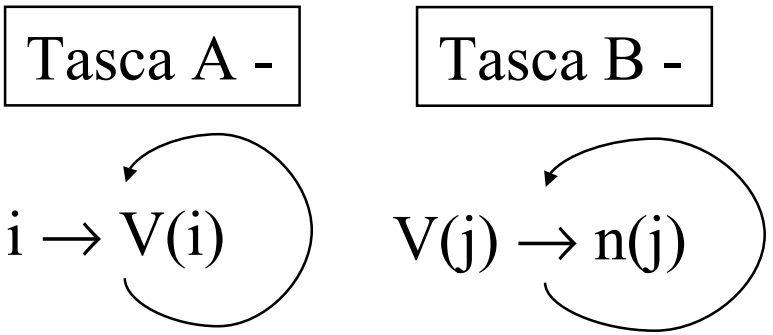
Tasca B -
V → n

Suposicions:
- Transferències de byte en byte
- Primer byte baix i després byte alt
- V inicialitzada a 0000h

Possible ordenació:
- Tasca B: 00h a *n*
- Gestor: Tasca B → Tasca A
- Tasca A: 04FFh → V
- Gestor: Tasca A → Tasca B
- Tasca B: 04h a *n*

n = 0400h !!

Accés consistent a variables compostes



Suposicions:
- V(i), vector de 10 elements inicialitzats a 0

Possible ordenació:
- Tasca B: 0 a *n(j)* (5 elem.)
- Gestor: Tasca B → Tasca A
- Tasca A: i → V(i) (10 elem.)
- Gestor: Tasca A → Tasca B
- Tasca B: i a *n(j)* (5 elem.)

n(j) : 0 0 0 0 0 6 7 8 9 10 !!

Execució consistent de codi

Tasca A -
PID → u
lim(u) → u

Tasca B -
u → *SortidaDA*

Possible ordenació:
- Tasca A: PID → u
- Gestor: Tasca A → Tasca B
- Tasca B: u → *SortidaDA*

u sense limitar !!

- Gestor: Tasca B → Tasca A
- Tasca A: lim(u) → u

Atomicitat

• **Instruccions atòmiques** són aquelles que un cop iniciades, el gestor no pot interrompre fins que hagin finalitzat

• Totes les instruccions en codi màquina (ensamblador) són atòmiques

• No és així amb totes les instruccions a alt nivell. Això farà necessari introduir mecanismes que garanteixin atomicitat en certes instruccions a alt nivell per evitar problemes en la protecció de recursos

4.2. Programació d'executius cíclics

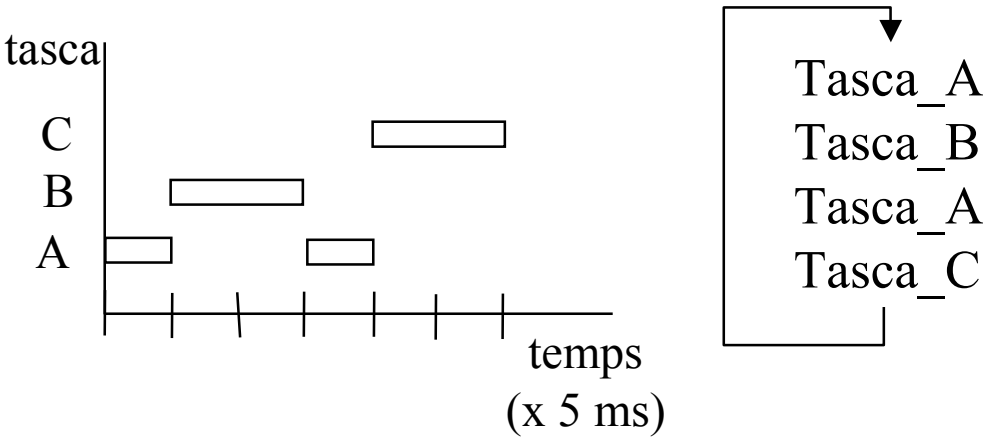
• Les tasques s'ordenen explícitament segons la corresponent seqüència d'execució planificada prèviament

• En cas que sigui necessari les tasques es subdivideixen en fragments de codi

• En aquest cas el sistema és totalment determinístic

Exemple:

Tasca	Temps Execució (ms)	Període Execució (ms)
Tasca_A	5	15
Tasca_B	10	30
Tasca_C	10	30

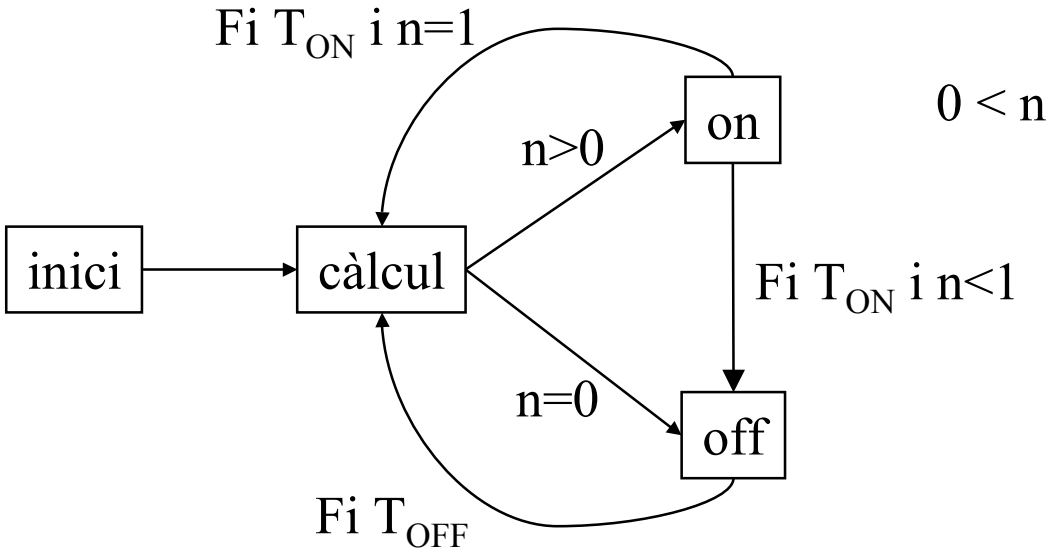


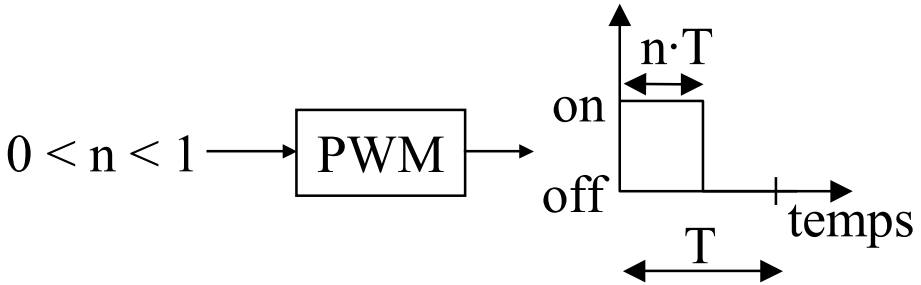
4.3. Exemples de programació de sistemes multitasca

4.3.1. Co-rutines. Exemple de multitasca cooperativa

Organització de les tasques

- Les tasques s'estructuren en estats i transicions
- S'associen accions als estats i condicions a les transicions
- Ni les accions ni la comprovació de condicions poden ser blocants





Càlcul:

T_{ON} = temps + n·T

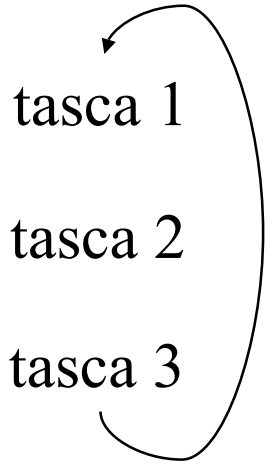
T_{OFF} = temps + T

Gestió per *scans* (I)

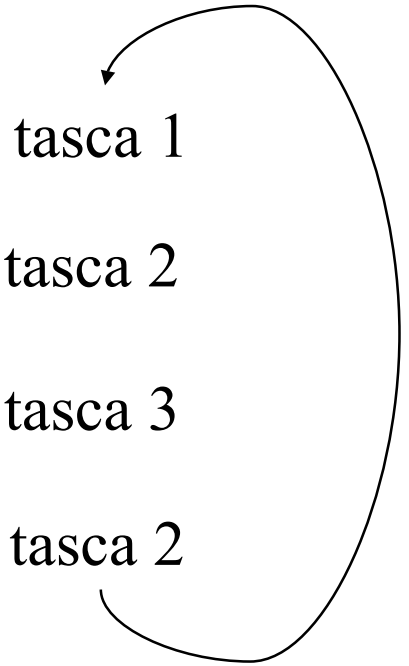
• L'execució es realitza per passades o *'scans'* (com els diagrames de contactes pels autòmats)
• Les tasques evolucionen mitjançant l'execució per *scans*
• En cada *scan* una tasca executa l'acció corresponent a l'estat actual i la comprovació de les condicions de sortida
• Si es compleix la condició d'una transició, la tasca canvia d'estat
• El gestor no gestiona les transicions. Només s'encarrega de determinar la següent tasca a dedicar-li un *scan*
• Segons la tipologia o importància de la tasca, el gestor pot decidir dedicar més *scans* a les més prioritàries

Gestió per *scans* (II)

Gestor seqüencial



Gestor amb prioritats

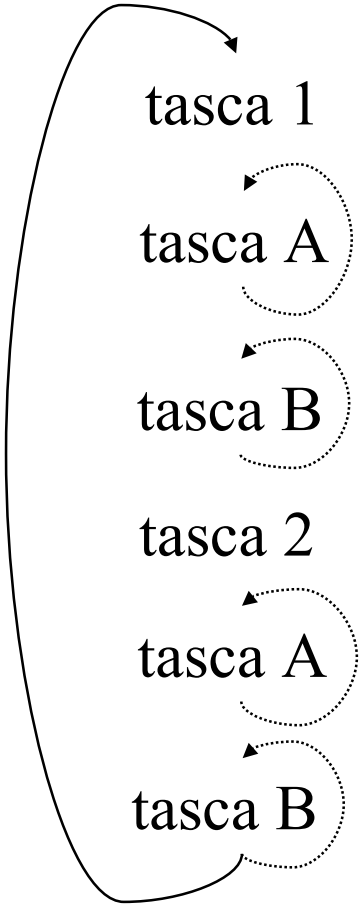


La tasca 2 és la més prioritària

Gestió per *scans* (III)

Gestor de mínima latència

- Les tasques que necessiten respondre a esdeveniments amb el mínim retard són gestionades amb preferència:
  - Reben un *scan* entre cada dues de les altres tasques
  - Si es detecta l'esdeveniment, el gestor els hi dedica tants *scans* com facin falta fins que finalitzin d'atendre'l
- Les altres tasques són gestionades seqüencialment



Les tasques A i B necessiten atenció especial per atendre a esdeveniments

Interacció de co-rutines i protecció de recursos

• Les tasques es comuniquen entre si mitjançant variables compartides.

• Les tasques es sincronitzen entre si mitjançant variables indicador compartides.

• Sempre que necessitin accedir a un recurs compartit durant més d'un *scan*, cal protegir-lo ja que pot ser que el gestor atorgui un *scan* a una altra tasca que vulgui accedir al mateix recurs. La protecció s'aconsegueix mitjançant variables indicador comaprtides.

Co-rutines: avantages i limitacions

• Permeten implementar un sistema multitasca de forma senzilla i ràpida.

• Els problemes d'accés simultani a recursos són de resolució intuitiva i simple.

• La resposta a esdeveniments sempre es produeix amb un cert retard ja que són reconeguts per enquesta.

• Com en tot sistema cooperatiu, la dinàmica d'execució de les tasques depèn en gran mesura de quan les pròpies tasques cedeixen la CPU.

4.3.2. Tasques en Ada. Exemple de multitasca apropiativa

• Ada permet organitzar l'aplicació en tasques concurrents. El gestor està implementat de forma transparent a l'usuari.

• Permet escollir una gestió basada en prioritats. Sinó s'especifica, la gestió per defecte és indeterminada. Normalment dependrà del SO.

```
task Tasca_A; --declaració

task body Tasca_A is
        <declaracions de variables
        i funcions pròpies>
begin
        <accions seqüencials>
end Tasca_A;
```

```
procedure Exemple is
        task Tasca_A;
        task Tasca_B;

        task body Tasca_A is <...>
        task body Tasca_B is <...>
begin
        null;
end Exemple;
```

Per tasques independents, els únics *punts* possibles *d'invocació* del gestor són la finalització d'una tasca i les instruccions de retard:

• L'aplicació finalitza quan totes les tasques finalitzen. Una tasca pot finalitzar en qualsevol moment amb l'instrucció *abort*. Una tasca abandona un bucle amb l'instrucció *exit*.

• Les instruccions *delay* i *delay until* suspenen la tasca en execució a l'espera que l'instant de temps especificat sigui superat.

• Un cop en suspensió el gestor invoca una altra tasca.

• Un cop assolit l'esdeveniment temporal la tasca passa a estar preparada per executar-se quan el gestor ho consideri oportú.

4.4. Protecció de recursos en sistemes apropiatius

A continuació proposarem certs mecanismes d'exclusió mútua basats en l'ús de variables indicador i altres, considerant les següents suposicions:

- Les tasques són gestionades per round robin time-slice
- La integritat en l'assignació de variables simples està garantida:

Tasca A	Tasca B
$10 \rightarrow x$	$15 \rightarrow x$

Després d'executar-se concurrentment ambdues tasques, **x** val o bé 10 o bé 15

Inhabilitació d'interrupcions

La primera solució consisteix en inhabilitar interrupcions mentre s'accedeix a la regió crítica

- És senzill d'implementar
- És efectiu
- No és òptim:impedeix l'activació d'altres tasques, encara que no hagin d'accedir al recurs

4.4.1. Variables indicador
Mecanisme I

Condicions inicials
indicador = 0

Tasca A
mentre indicador = 1 esperar
1 → indicador
Secció Crítica A
0 → indicador

Tasca B
mentre indicador = 1 esperar
1 → indicador
Secció Crítica B
0 → indicador

- Tasca A: mentre …
- Gestor: A → B
- Tasca B: mentre …, 1 → indicador, SC B
- Gestor: B → A
- Tasca A: 1 → indicador, SC A

SC A i SC B !!

Mecanisme II

Condicions inicials
indicador = 0

Tasca A
mentre indicador = 1 esperar
Secció Crítica A
1 → indicador

Tasca B
mentre indicador = 0 esperar
Secció Crítica B
0 → indicador

Alternància estricte

Mecanisme III

Condicions inicials
indicador1 = 0
indicador2 = 0

Tasca A
mentre indicador2 = 1 esperar
1 → indicador1
Secció Crítica A
0 → indicador1

Tasca B
mentre indicador1 = 1 esperar
1 → indicador2
Secció Crítica B
0 → indicador2

- Tasca A: mentre …
- Gestor: A → B
- Tasca B: mentre …, 1 → indicador2, SC B
- Gestor: B → A
- Tasca A: 1 → indicador1, SC A

SC A i SC B !!

Mecanisme IV

Condicions inicials
indicador1 = 0
indicador2 = 0

Tasca A
1 → indicador1
mentre indicador2 = 1 esperar
Secció Crítica A
0 → indicador1

Tasca B
1 → indicador2
mentre indicador1 = 1 esperar
Secció Crítica B
0 → indicador2

- Tasca A: 1 → indicador1
- Gestor: A → B
- Tasca B: 1 → indicador2

Ambdues tasques en espera indefinida → livelock !!

Solució de Peterson

Condicions inicials
indicador1 = 0
indicador2 = 0

Tasca A
1 → indicador1
2 → torn
mentre indicador2 = 1 i torn = 2 esperar
Secció Crítica A
0 → indicador1

Tasca B
1 → indicador2
1 → torn
mentre indicador1 = 1 i torn = 1 esperar
Secció Crítica B
0 → indicador2

- solució efectiva
- esperes indesitjables
- complexe de dissenyar, testejar i compendre

4.4.2. Test and Set

• La problemàtica esdevé del fet de no poder comprovar el valor de l'indicador i canviar-lo de forma atòmica.
• Alguns SO o llenguatges implementen test & set (TAS)
• TAS(*indicador*) assigna 1 a *indicador* i retorna el valor anterior d'*indicador* de forma totalment atòmica. Si retorna 0 el recurs és disponible, sinó cal esperar
• Comporta esperes indesitjables, però és senzill
• Les esperes comporten blocatges en cas de tasques periòdiques prioritzades

Tasca A
    mentre TAS(indicador) = 1 esperar
    Secció Crítica A
    0 → indicador

Tasca B
    mentre TAS(indicador) = 1 esperar
    Secció Crítica B
    0 → indicador

4.4.3. Semàfors

• La millora consisteix en suspendre la tasca que està a l'espera d'un recurs. El gestor no la torna a tenir en compte fins que el recurs és alliberat. Llavors, si té la major prioritat de totes les tasques preparades, serà reactivada. En cas contrari permaneixerà com a preparada

• El protocol de protecció del recurs consisteix amb l'ús de dues instruccions, una per accedir al recurs i l'altre per alliberar-lo.

•Exemple d'ús (inicialment *semàfor* val 1):

```
Tasca A
    Accés(semàfor)
    Secció Crítica A
    Alliberament(semàfor)
```

```
Tasca B
    Accés(semàfor)
    Secció Crítica B
    Alliberament(semàfor)
```

• *semàfor* s'ha d'inicialitzar al número de recursos disponibles

Accés(semàfor)
- si *semàfor* val 0, la tasca es suspèn
- si *semàfor* és positiu, es decrementa i la tasca accedeix a la secció crítica

Alliberament(semàfor)
- si hi ha tasques a l'espera, la més prioritària és activada
- si no hi ha tasques a l'espera, *semàfor* s'incrementa

Un cas particular és quan hi ha un sol recurs. Llavors es pot utilitzar el *semàfor binari*.

Deadlock

Tasca A
- Accés(semàfor1)
- *Secció A1*
- Accés(semàfor2)
- *Secció A2*
- Alliberament(semàfor2)
- Alliberament(semàfor1)

Tasca B
- Accés(semàfor2)
- *Secció B1*
- Accés(semàfor1)
- *Secció B2*
- Alliberament(semàfor1)
- Alliberament(semàfor2)

- Tasca A: Accés(semàfor1), *Secció A1*
- Gestor: A → B
- Tasca B: Accés(semàfor2), *Secció B1*

Ambdues tasques en suspensió indefinida → deadlock !!

Altres mecanismes d'interacció més estructurats mitjançant memòria compartida

• Els semàfors són senzills i eficients, però presenten certs inconvenients degut al seu poc nivell d'abstracció:

- L'omissió d'un sol *Alliberament* segurament provocarà un *deadlock*
- L'omissió d'un sol *Accés* segurament provocarà una falta de exclusió mútua
- Incloure *Accés* o *Alliberament* en llocs incorrectes provocarà certament un comportament erroni
- Degut a la carència d'estructuració, pot resultar complex de depurar

4.4.4. Regions crítiques

- No s´han de confondre amb *seccions crítiques*

- Certs llenguatges o SO proporcionen aquest mecanisme estructurat de protecció:
  - Les variables que s'han d'accedir en exclusió mútua s'han de declarar d'una manera especial
  - Tot codi que faci ús d'aquesta variable ha d'estar inclòs en una estructura especial

- D'aquesta manera les inconsistències poden detectar-se en temps de compilació

```
Tasca A
    V: compartida
    IniciRegió V
        Secció crítica (Ex: V + 1 → V)
    FiRegió V
```

4.4.5. Monitors

Certs llenguatges o SO proporcionen aquest mecanisme estructurat de protecció:

- Les variables que s'han d'accedir en exclusió mútua s'han de declarar d'una manera especial.

- Aquestes variables només es poden manipular mitjançant un conjunt de funcions especialment declarades.

- Es garanteix l'exclusivitat en la crida a aquestes funcions. Si el recurs està ocupat, tota tasca que vulgui accedir simultàniament serà suspesa.

• La declaració de les variables i de les funcions associades es fa de forma agrupada en un bloc estructurat, en contraposició a les regions crítiques, en que estava dispers

```
Monitor M
    Variables
        V: compartida
    Funcions
        Funció1
            Codi (Ex: V + 1 → V)
        Funció2
            Codi (Ex: V - 1 → V)
FiMonitor
```

```
Tasca A
    M.Funció1
```

```
Tasca B
    M.Funció1
    M.Funció2
```

```
Tasca C
    M.Funció2
```

4.4.6. Protecció de recursos en Ada

• Ada permet la protecció de recursos amb un mecanisme similar als monitors: **objectes protegits**.

• Un objecte protegit agrupa un conjunt de variables privades i de subprogrames que les manipulen:

- Les funcions permeten accés de lectura simultani.
- Els procediments permeten accés de lectura i escriptura exclusiu.
- Les entrades (*entry*) permeten accés condicional de lectura i escritura exclusiu.

• Quan una tasca realitza una crida a un subprograma d'un recurs en ús es suspèn a l'espera del seu alliberament. També es produeix la suspensió quan una tasca realitza una crida a una entrada amb incompliment de la condició. Es pot limitar el bloqueig causat per l'entrada mitjançant un *timeout* (*select-or-delay*) o fins i tot ignorar i continuar (*select-else*).

• Serveixen per protegir memòria com altres recursos compartits.

Declaració

```
protected Recurs is
        procedure Accio1(...);
        function Accio2(...) return ...;
        entry Accio3(...);
private
        <declaració de variables privades>
end Recurs;

protected body Recurs is
        procedure Accio1(...) is
                <...>
        function Accio2(...) return ... is
                <...>
        entry Accio3(...) when <condició> is
                <...>
end Recurs;
```

Crides

```
Recurs.Accio1(...);

a := Recurs.Accio2(...);

select
        Recurs.Accio3(...);
or
        delay 10.0;
end select;

select
        Recurs.Accio3(...);
else
        <...>
end select;
```

4.5. Comunicació mitjançant missatges

• La interacció es fa mitjançant el lliurament i la recepció d'informació com si de missatges es tractés

• No s'utilitza cap tipus de variable compartida, evitant els problemes vistos de compartiment de memòria

• En sistemes amb memòria compartida és un mecanisme alternatiu d'interacció

• Per sistemes distribuïts l'ús de memòria compartida deixa de tenir sentit. Tanmateix la inhabilitació d'interrupcions no és possible. Llavors l'ús de missatges és l'única alternativa possible

• Normalment aquest tipus d'interacció involucra una forta component de sincronisme

4.5.1. Mecanismes i modalitats de transacció

Des del punt de vista del *receptor*, aquest ha d'esperar a que sigui prèviament lliurat per l'emissor. En el cas que no n'hi hagi cap existeixen dos plantejaments:

- Es queda suspès a l'espera d'un nou missatge.
- Continua la seva execució sense cap missatge.

Des del punt de vista de l'*emissor* existeixen diverses modalitats de transacció:

- *lliurament asíncron*: un cop lliurat el missatge, l'emissor continua. No hi ha confirmació de recepció, ni el receptor sap l'estat de l'emissor. Ex: lliurament d'una carta
- *lliurament síncron* (*rendezvous*): l'emissor es suspèn a l'espera que el receptor rebi el missatge. Ex: fax
- *invocació remota* (*rendezvous estès*): l'emissor espera més enllà de la recepció, suspenent-se a l'espera que el receptor elabori una missatge de rèplica. Ex: telèfon

Existeixen diverses possibilitats segons la manera en que s'anomena el destinatari o el remitent d'un missatge:

- Transacció *directa*
  - el missatge s'envia a una tasca en concret
  - el missatge s'espera arribi d'una tasca en concret
  - un missatge s'espera procedent d'una tasca sense identificar

  Per exemple: **envia** <*missatge*> **a** <*tasca*>

- Transacció *indirecta*
  - la transacció es fa per mitjà d'un canal o bústia

  Per exemple: **reb** <*missatge*> **de** <*bustia*>

Existeixen diverses possibilitats segons si s'anomena el destinatari i/o el remitent d'un missatge:

- Transacció *simètrica*
  Emissor i receptor són explícitament especificats entre ells en cada transacció.
  Per exemple: **reb** <*missatge*> **de** <*tasca*>

- Transacció *assimètrica*
  La recepció o el lliurament es fan de forma anònima.
  Per exemple: **reb** <*missatge*>

4.5.2. Invocació remota en Ada

Ada proveeix d'un potent mecanisme de comunicació basat en missatges: *invocació remota directa assimètrica*.

Dues activitats coincideixen en el temps, es passen informació i continuen.

La seva declaració i definició es fa de forma similar a la d'un subprograma mitjançant una entrada (*entry*).

El receptor insereix el codi de l'entrada allí on està preparat per acceptar comunicacions.

L'emissor realitza una crida a l'entrada allí on està preparat per iniciar comunicació.

La comunicació no s'inicia fins que emissor i receptor no assoleixen els punts de comunicació respectius. En qualsevol altre cas el que arriba primer és suspès a l'espera que l'altre arribi.

Un cop iniciada la comunicació, el receptor executa un codi de resposta i continua. L'emissor continua suspès fins que el receptor no haig executat el codi de resposta.

- El receptor pot especificar:
  - el lloc en que efectuar la resposta dins el seu codi (*accept*).
  - una espera selectiva entre vàries entrades (*select-accept-or*).
  - una espera condicional (*select-when-accept*).
  - una recepció sense suspensió (*select-accept-else*).
  - una espera amb *timeout* (*select-accept-or-delay*).
  - una espera amb finalització automàtica (*select-or-terminate*).

- L'emissor pot especificar:
  - el lloc en que efectuar la crida.
  - una recepció sense suspensió (*select-else*).
  - una espera amb timeout (*select-or-delay*).

```
task Tasca_Receptora is
        entry Accio1(...);
        entry Accio2(...);
end Tasca_Receptora;

task body Tasca_Receptora is
        <...>
begin
        <...>
        select
                when <condició> =>
                        accept Accio1(...) do
                                <...>
                        end Accio1;
        or
                accept Accio2(...) do
                        <...>
                end Accio2;
        or
                delay 10.0;
        end select;
        <...>
end Tasca_Receptora;
```

Dins la tasca emissora

```
<...>
Tasca_Receptora.Accio1(...);
<...>
select
        Tasca_Receptora.Accio2(...);
or
        delay 10.0;
end select;
<...>
select
        Tasca_Receptora.Accio1(...);
else
        <...>
end select;
<...>
```

Sincronisme de tasques per mitjà de la invocació remota

La invocació remota permet implementar un mecanisme de sincronisme pur. Per exemple es desitja garantir que la *Secció A1* i la *Secció B1* s'executaran sempre ambdues abans que la *Secció A2* o la *Secció B2*.

```
task Tasca_A is
        entry Espera;
end Tasca_A;

task body Tasca_A is
        <...>
begin
        <Secció A1>
        accept Espera;
        <Secció A2>
end Tasca_A;
```

```
task Tasca_B;

task body Tasca_B is
        <...>
begin
        <Secció B1>
        Tasca_A.Espera;
        <Secció B2>
end Tasca_B;
```

4.6. Gestió apropiativa basada en prioritats
4.6.1. Mecanisme de planificació

La gestió basada en prioritats és la més adecuada per un sistema en temps real ja que permet realitzar un estudi de planificabilitat que garanteixi els terminis desitjats.

- Totes les cues d'espera (preparades i en suspensió per cada esdeveniment) s'ordenen per prioritats.

- Les tasques de major prioritat són servides abans pel gestor.

- Tasques d'igual prioritat són tractades segons el criteri FIFO.

4.6.2. Inversió de prioritats

Problemes amb tasques que interaccionen amb blocatge

• Quan una tasca vol accedir a un recurs que alguna altra està ja utilitzant, aquesta es queda blocada a l'espera que s'alliberi el recurs, siguin quines siguin les seves prioritats respectives

• Aquest fet pot comportar problemes de planificabilitat (i.e., determinades tasques no podran finalitzar dins el seu termini establert)

• Dos problemes concrets apareixen: inversió de prioritats i *deadlocks*

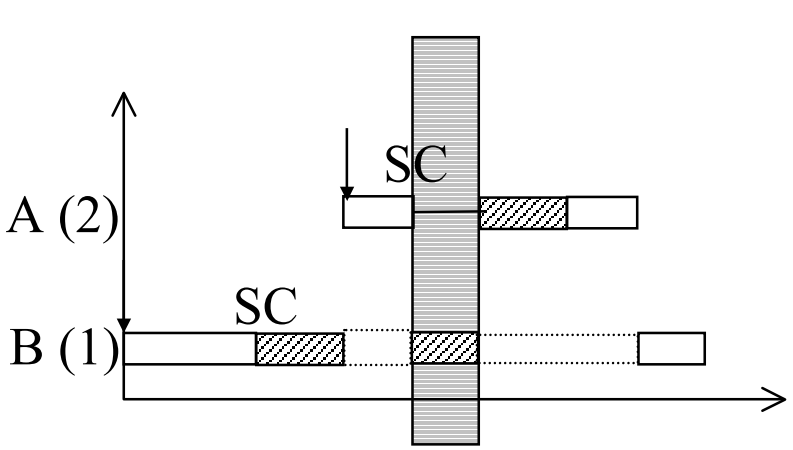
• Existeixen mecanismes per garantir la planificabilitat. Tots ells es basen en modificar dinàmicament la prioritat de les tasques mentre accedeixen a un recurs compartit

• Quan varies tasques interaccionen es poden produir dos tipus de bloqueigs: directes i indirectes.

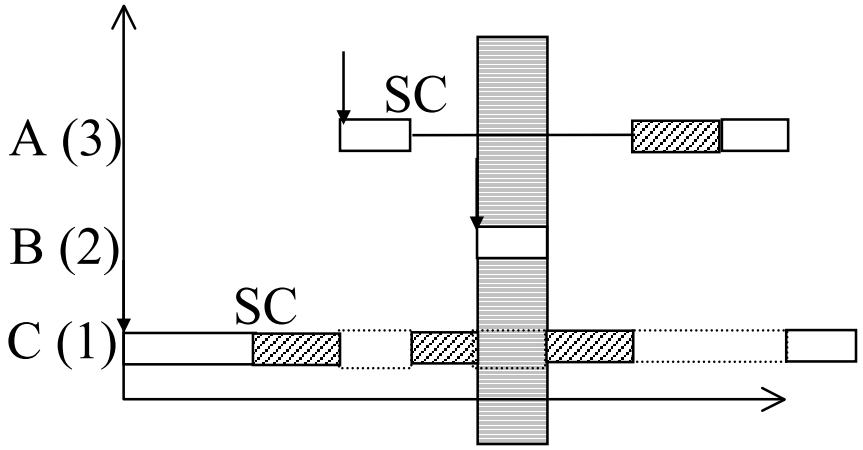
• Ambdós bloqueigs poden comportar que tasques de menys prioritat gaudeixin de CPU mentre altres tasques de més prioritat estiguin blocades. Es diu que s'està produint una inversió de prioritats.

• El bloqueig directe és necessari per garantir l'integritat del recursos compartit.

• El bloqueig indirecte pot esdevenir no acotat i és desitjable eliminar-lo.



bloqueig directe



bloqueig indirecte

4.6.3. Herència de prioritats

• Permet acotar la inversió de prioritats i així planificar millor.

• Elimina el bloqueig indirecte i pot arribar a evitar la possibilitat de *deadlock*.

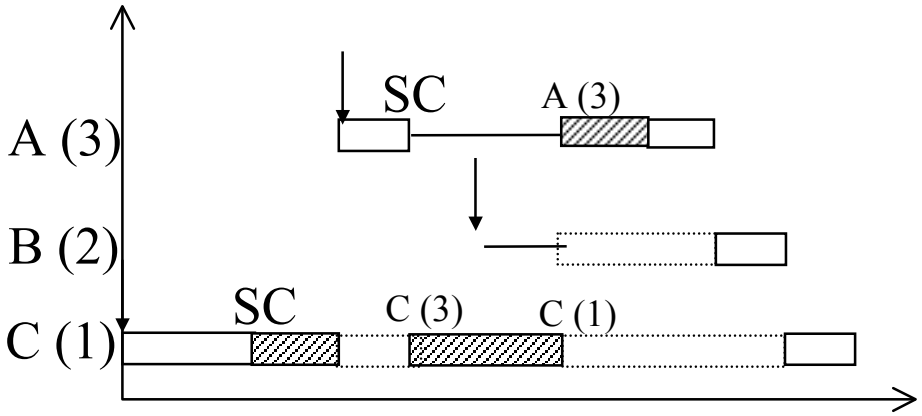
• Es basa en augmentar dinàmicament la prioritat de la tasca de menor prioritat bàsica durant una part o tot de l'accés al recurs compartit.

• Veurem dos dels mecanismes existents: l'herència de prioritat simple i el protocol de sostre de prioritat immediat.

Herència de prioritats simple

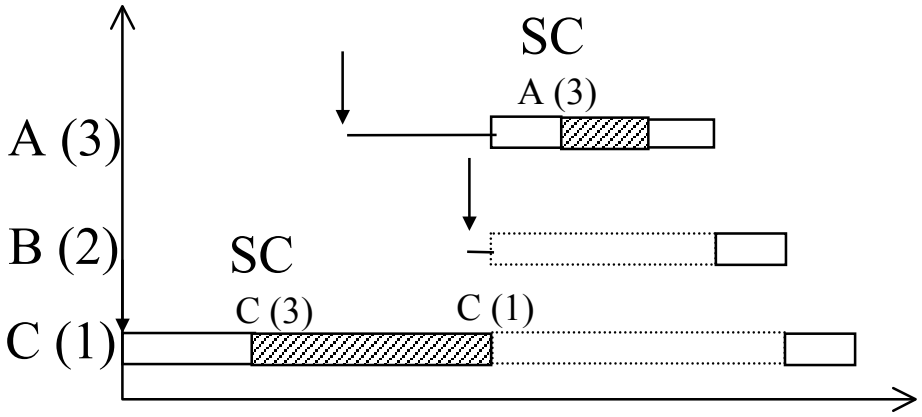
• En el moment del bloqueig, la tasca menys prioritària hereta la prioritat de la tasca que es bloqueja fins que abandoni la secció crítica, moment en que finalitza el bloqueig i recupera la seva prioritat bàsica original.

• Elimina el bloqueig indirecte però no evita la possibilitat de *deadlock*.



Protocol de sostre de prioritat immediat

• Cada recurs té un sostre de prioritat definit per la màxima prioritat de les tasques que l'utilitzen.
• En el moment d'accedir a un recurs, la tasca hereta el sostre de prioritat del recurs.
• En el moment d'alliberar el recurs recupera la seva prioritat bàsica.
• Elimina el bloqueig indirecte i evita a la vegada la possibilitat de *deadlock*.



3.6.4. Gestió basada en prioritats en Ada

El paquet *Real_Time* permet parametritzar la gestió multitasca d'Ada per treballar amb prioritats.

La prioritat bàsica d'una tasca s'assigna incloent *pragma Priority(prioritat);* en la seva especificació. A major valor de *prioritat*, major prioritat.

L'ordre d'encuament de tasques en les entrades (*entry*) de tasques i objectes protegits és FIFO per defecte. Si es vol ordenament per prioritats cal incloure *pragma Queuing_Policy(Priority_Queuing);*.

L'ordre d'encuament de tasques preparades i a l'espera d'esdeveniments temporals és indeterminat en cas d'igualtat de prioritats. Es pot escollir l'encuament segons el criteri FIFO en aquests casos amb l'instrucció

pragma Task_Dispaching_Policy(FIFO_Within_Priority);

Ada permet gestionar l'accés als recursos compartits segons el protocol de sostre de prioritat immediat. Per activar-ho cal:

- Incloure *pragma Locking_Policy(Ceiling_Locking);*
- Incloure *pragma Priority(prioritat);* en l'especificació d'un objecte protegit per assignar el sostre de prioritat al recurs.

Ada permet treballar amb prioritats dinàmiques amb les instruccions *Set_Priority* i *Get_Priority* definides en el paquet *Dynamic_Priorities*.

Detecció de terminis no assolits

Ada permet detectar violacions de terminis utilitzant la construcció *select-delay until-then abort*.

```
select
        delay until Instat;
        <Accions per timeout>
then abort
        <Accions normals>
end select;
```

```
task Tasca_Periodica;
task body Tasca_Periodica is
   Periode : Time_Span := Milliseconds(1000);
   Temps_Seguent : Time := Clock + Periode;
   Termini : Time_Span := Milliseconds(900);
   Termini_Seguent : Time := Temps_Seguent + Termini;
begin
   loop
      delay until Temps_Seguent;
      select
         delay until Termini_Seguent;
      then abort
         <Codi periòdic>
      end select;
      Temps_Seguent := Temps_Seguent + Periode;
      Termini_Seguent = Termini_Seguent + Termini;
   end loop;
end Tasca_Periodica;
```

5. Planificació d'un SITR
5.1. Conceptes i definicions

La gestió basada en prioritats permet realitzar un estudi de planificabilitat *a priori* analític.

L'anàlisi que es descriurà es fonamenta en vàries **suposicions**, encara que n'existeixen de més exhaustives que relaxen algunes d'elles.

Les especificacions temporals es concreten per cada tasca mitjançant un **termini** D. L'anàlisi de planificabilitat ens ha de dir si el conjunt de tasques gestionades per prioritats acompleixen els seus terminis, en el cas més **desfavorable**.

Per realitzar l'anàlisi és necessari conèixer de cada tasca el seu temps de **còmput** màxim, C_M i la seva **prioritat**, P. En un principi suposarem que la prioritat és coneguda. Més endavant veurem **criteris** d'assignació de prioritats.

Suposarem que treballarem només amb tasques **periòdiques** de les que es coneix el **període**, T.

Si treballem amb tasques que comparteixen **recursos** haurem de conèixer els màxims temps d'**accés** de cada tasca per cada recurs, C_X i el mecanisme d'herència de prioritats.

Suposarem finalment que el temps de CPU utilitzat pel **gestor** és despreciable.

Factor d'utilització de la CPU

Una condició necessària però no suficient és que el factor d'utilització de la CPU sigui inferiror a la unitat en cas d'un sistema monoprocessador.

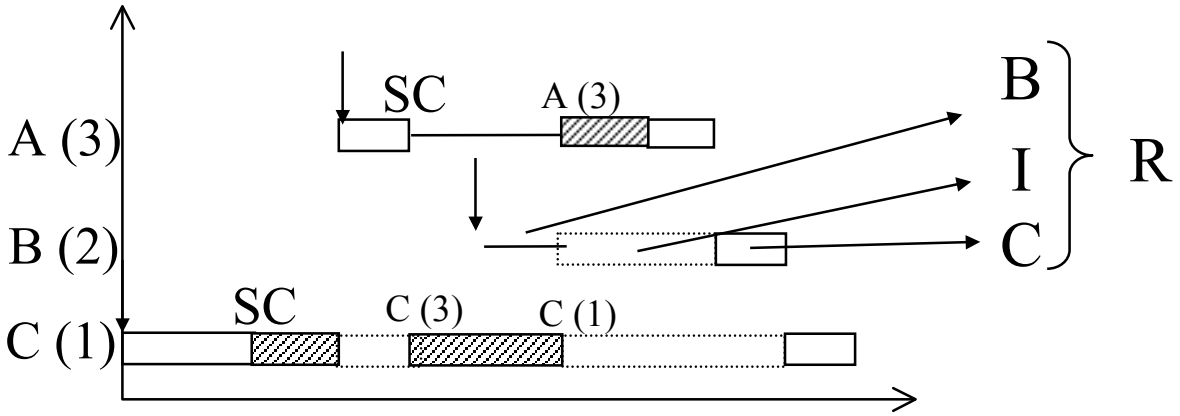
$$U = \sum_i \frac{C_i}{T_i} \leq 1$$

5.2. Càlcul del temps de resposta a esdeveniments

Avaluació analítica del temps de resposta (I)

Una condició suficient i necessària es pot obtenir determinant analíticament el temps de resposta a un esdevenient, R. Llavors es tracta de comprovar si per cada tasca $R_i \leq D_i$.

El temps de resposta R es pot descomposar en tres termes R = C + I + B. **I** és l'**interferència** deguda a tasques de prioritat superior i **B** és el **bloqueig** degut a que tasques de prioritat inferiror estan accedint a recursos compartits.



Avaluació analítica del temps de resposta (II)

Si partim de l'instant crític, el temps de resposta d'una tasca es veurà allargat per l'apropiació de tasques més prioritàries (R > C). Per cada apropiació d'una tasca més prioritària el temps de resposta s'augmentarà amb el temps de còmput d'aquella tasca. Així es pot obtenir la següent expressió per I on **j** són les tasques més prioritàries que **i**.

$$I_i = \sum_j \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Avaluació analítica del temps de resposta (III)

Per calcular el valor de B, cal seguir la següent metodologia:

- Enumerar els recursos a que accedeixen tasques de prioritat inferior a la **i**.
- Enumerar d'entre ells els que hi accedeixen també tasques de prioritat igual o superior a la **i**.
- Pels recursos que compleixen ambdues condicions escollir el màxim temps d'accés.

El temps de resposta s'avalua aplicant de forma recursiva l'equació següent, partint d'un valor inicial R_i^0 fins que el valor de R s'estabilitzi.

$$R_i^{k+1} = C_i + \sum_j \left\lceil \frac{R_i^k}{T_j} \right\rceil C_j + B_i \qquad R_i^0 = C_i + \sum_j C_j + B_i$$

Avaluació analítica del temps de resposta (IV)

• El temps de còmput màxim es pot determinar o bé per mesura o bé per anàlisi

• La dificultat pot estar en determinar realment el valor màxim. Per facilitar aquest fet cal programar evitant per exemple bucles no acotats

• Un tècnica d'anàlisi per determinar aquest temps consisteix en
  • descomposar el codi en blocs bàsics
  • determinar el temps associat a cada bloc a partir del codi màquina corresponent
  • combinar adequadament tots els temps per determinar-ne el total definitiu

5.3. Criteris d'assignació de prioritats

• L'objectiu és assignar prioritats a les tasques de forma que s'acompleixin els requeriments temporals

• Cada tasca té assignada una prioritat que indica la seva importància relativa respecte a les demés tasques que constitueixen l'aplicació

• En tot moment, la tasca amb més prioritat és la que està activa

• L'assignació de prioritats pot ser **estàtica** (i.e., constant o fix) o **dinàmica** (i.e., variable en funció de l'estat del sistema)

• Exemples de criteris d'assignació de prioritats:
  • *estàtiques*: prioritat al més freqüent, prioritat al més urgent
  • *dinàmiques*: primer el més urgent, primer el menys folgat

Prioritat al més freqüent

En anglès és *rate monotonic scheduling*, RMS

- És un mecanisme d'assignació de prioritats estàtiques
- Cal que $C_M \leq D = T$ per cada tasca
- S'assignen les prioritats en ordre invers al dels seus períodes
- És una assignació òptima en el sentit que si d'aquesta manera no es poden assolir els requeriments temporals, no es podrà aconseguir amb cap altre mètode basat en prioritats estàtiques
- Els terminis estan garantits si $U = \sum_{i=1}^{n} \frac{C_M^i}{T_i} \leq n\left(2^{1/n} - 1\right)$

Prioritat al més urgent

En anglès és *deadline monotonic scheduling*, DMS

- És una generalització del RMS
- També és un mecanisme d'assignació de prioritats estàtiques
- S'assignen les prioritats en ordre invers al dels seus terminis
- És una assignació també òptima
- Cal que $C_M \leq D \leq T$ per cada tasca

Primer el més urgent

En anglès és *earliest deadline first*, EDF

- És un mecanisme dinàmic de prioritització
- Es prioritza en tot moment la tasca amb el termini més proper
- És una assignació òptima. Només cal que $U \leq 1$ per acomplir els terminis
- Cal que $C_M \leq D \leq T$ per cada tasca

Primer el menys folgat

En anglès és *least slack first*, LSF

- És un mecanisme dinàmic de prioritització
- La *folgura* és la diferència entre el temps que falta per acabar el termini i el temps de còmput que falta per executar
- Es prioritza en tot moment la tasca amb menys folgura
- És una assignació òptima. Només cal que U≤1 per acomplir els terminis
- Cal que $C_M \leq D \leq T$ per cada tasca

6. Sistemes distribuïts

6.1. Aspectes dels sistemes distribuïts en temps real: Conceptes i definicions

Sistema distribuït - un sistema constituït per múltiples unitats processadores autònomes inerconnectades que cooperen per aconseguir un objectiu comú.

- Sistemes fortament acoblats: quan comparteixen memòria
- Sistemes dèbilment acoblats: quan no la comparteixen

Dues característiques interessants:

- La referència de temps no pot diferir d'un node a un altre. En l'accés a recursos compartits pot comportar problemes
- La xarxa de comunicacions passa a ser un recurs compartit més. No pot gestionar-se com la CPU ja que les apropiacions portarien a retransmetre el missatge

Algorisme d'ordenació d'esdeveniments

Problema:

- Cada node treballa amb un rellotge diferent
- La xarxa pot comportar diferents retards en la recepció de missatges
- La sincronització necessària per accedir a recursos compartits es fa mitjançant missatges
- L'ordenació d'esdeveniments pot ser que no sigui la mateixa per tots els nodes

Solució de Lamport:

- Cada màquina associa una marca temporal lògica a cada esdeveniment que genera
- Només cal que les marques entre nodes siguin coherents a partir de que interaccionin entre sí. Amb cada lliurament caldrà enviar la marca temporal
- Si la recepció és síncrona la marca del receptor s'igualarà a la marca del missatge

Exclusió mútua distribuïda

Cal un algorisme distribuït per accedir seccions crítiques

Solució de Ricart i Agrawala:

- Quan un node vol accedir a una secció crítica cal que rebi el permís de tots els nodes als quals ha de garantir exclusió mútua
- Cada missatge va acompanyat de la marca temporal
- Quan un node rep una petició d'accés a una secció crítica es pot donar una de les següents situacions:
  - Si el node receptor és a la seva secció crítica posposa la rèplica
  - Si el node receptor no vol accedir a la seva secció crítica dóna permís
  - Si el node receptor està a l'espera d'accedir a la seva secció crítica i la marca temporal del seu missatge lliurat és menor que la del missatge rebut posposa la rèplica. Donarà permís si es produeix a l'inrevés
- Quan un node allibera una secció crítica dóna permís a tots els nodes als que havia posposat la rèplica

6.2. Xarxes de comunicació en temps real: CAN

Característiques de les xarxes en temps real:

- Missatges de poc tamany (10:20 bytes)
- Transaccions a elevada velocitat (~ Mbits)
- Protocol d'accés al medi no destructiu ni purament aleatòria (CSMA/CD vs. pas de testimoni)
- Mecanismes de detecció d'errors robusts: transmissió fiable

Controller Area Network, CAN (I)

- Utilitzada a la indústria del automòbil
- Trama de 8 bytes de dades amb identificador de 11 bits
- Velocitats fins a 1 Mbit
- Accés al medi CSMA/CD amb resolució de col·lisió:
  - L'identificador indica prioritat del missatge (0 prioritat màxima)
  - Durant el lliurament simultani de missatges, la xarxa actua com una AND
    - Si un node envia un 0 passa al següent bit
    - Si un node envia un 1 i llegeix un 1 passa al següent bit
    - Si un node envia un 1 però llegeix un 0 abandona fins a finalitzar

Controller Area Network, CAN (II)

- Sincronisme: transmissió asíncrona
  - Sincronisme a l'inici de trama
  - Resincronització bit a bit

- Detecció d'errors a múltiples nivells (probabilitat d'error no detectat 10^{-11})

  A nivell de missatge
  - Cyclic Redundancy Check
  - Bits d'estat fix en la trama

  A nivell de bit
  - Monitorització dels bits emesos
  - Bit stuffing