

La planificació dels processos

Sistemes Operatius I

Lluís Garrido – lluis.garrido@ub.edu

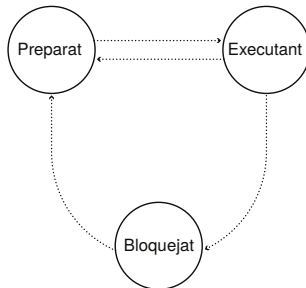
Grau d'Enginyeria Informàtica

Organització

- 1 Estats d'un procés
- 2 Planificació
- 3 Conceptes de planificació
- 4 Algorismes de planificació

Estats d'un procés

El sistema operatiu gestiona l'execució dels processos a l'ordinador. Els estats d'un procés són bàsicament tres: **preparat**, **executant** i **bloquejat**¹.



¹Fixar-se en les fletxes del graf!

Estats d'un procés

Descripció dels estats

- Un procés està **preparat** si està disponible per ser executat però no està executant realment. El sistema operatiu el té emmagatzemat en aquells moments com un dels candidats a ser executat.
- Un procés està **executant** si s'està executant a algun dels processadors disponibles.

Observar que

- Un procés pot canviar d'estat entre preparat i executant múltiples (milers!) vegades mentre executa.
- En general el sistema operatiu decideix quan un procés deixa d'executar i passa a l'estat de preparat, i a la inversa. El procés també pot, voluntàriament, deixar d'executar i passar a estat preparat (funció `sched_yield` de C).

Descripció dels estats

- Un procés està **bloquejat** si està esperant algun esdeveniment. Un procés bloquejat no és elegible per poder ser executat pel sistema operatiu fins que té lloc l'esdeveniment. Aleshores passa de l'estat bloquejat a preparat.

Un procés es pot bloquejar quan s'està executant i realitza una operació com, per exemple,

- Crida a una funció de lectura: crida a una funció per llegir dades de xarxa, de teclat o de disc. Fins que les dades no estiguin disponibles el procés està bloquejat.
- S'adorm voluntàriament amb un `sleep` (adormir-se uns segons) o `wait` (esperar que el procés fill acabi).

En passar un procés de mode executant a preparat o bloquejat es produeix un **canvi de context**

- Per deixar d'executar un procés, el sistema operatiu ha d'emmagatzemar tota la informació del procés que s'està executant al seu BCP (Bloc de Controlde Processos). Entre altres, ha d'emmagatzemar-hi tots els valors dels registres de la CPU en què s'estava executant el procés (comptador de programa, punter a la pila, etc.)
- Un cop el sistem operatiu escull el procés a executar cal restaurar els valors de registres emmagatzemats al seu BCP. Aquests valors es "copien" a la CPU i així el procés continua executant "allà on ho havia deixat".

Perquè es pugui fer un canvi de context el BCP emmagatzema, per a cada procés,

- Un identificador de procés (un sencer a UNIX).
- L'estat del procés (preparat, executant, bloquejat).
- Els registres de CPU en cas que estigui preparat o bloquejat.
- Els privilegis i/o prioritat.
- En quina part de la memòria física resideix, quin és el fitxer executable associat, etc.

Cal diferenciar entre

- El **canvi de context**: el mecanisme mitjançant el qual el sistema operatiu canvia l'execució d'un procés a un altre; concretament el mode dual de funcionament.
- El **planificador**: l'algorisme que utilitza el sistema operatiu per decidir quin és el següent procés a executar. Cada sistema operatiu pot utilitzar una tècnica diferent (i, per tant, informació diferent) per fer aquesta decisió.

El **canvi de context** es pot realitzar gràcies al mode dual de funcionament de la màquina. Un exemple:

- 1 Es produeix una interrupció de temporitzador.
- 2 El maquinari interromp l'execució del procés actual, emmagatzema els valors dels registres de la CPU a una posició de memòria coneguda pel sistema operatiu i es crida a una determinada funció del sistema operatiu.
- 3 El sistema operatiu pot decidir canviar de procés. Emmagatzema a la BCP la informació dels registres emmagatzemada memòria i la sobreescriu amb els nous valors del procés a executar.
- 4 En retornar de la interrupció el maquinari carrega a la CPU els nous valors... i voilà, ja hem fet un canvi de context!

Quan es pot produir un canvi de context? En general...

- En produir-se una **interrupció de temporitzador**: el planificador pot decidir deixar d'executar el procés actual i passar a executar un altre.
- En produir-se un **altre tipus d'interrupció**: en arribar dades per la xarxa, en haver-se llegit les dades de disc, ... s'interromp l'execució del procés. El planificador pot decidir executar, per exemple, el procés que ha demanat les dades o continuar l'execució del procés que hi havia en aquell moment.
- En **bloquejar-se un procés**: en aquest moment el planificador ha de decidir quin dels processos preparats passa a executar-se.
- Altres casos no llistats aquí.

Si hi ha múltiples processos preparats, com escollim quin és el següent procés a executar? Quina és la millor opció? El planificador determina quins processos s'han d'executar primer.

Sembla que hi ha solucions òbvies

- La solució “més justa” sembla executar els processos fent servir una cua FIFO circular. Però això a vegades pot ser la pitjor solució, sobretot si volem que l'usuari percebi visualment una bona resposta. Ara ho veurem!
- Avui en dia els ordinadors són molt ràpids i, per tant, sembla que no sigui important aquest problema. Però els servidors, per exemple, van molt sobrecarregats i cal decidir bé quin és el següent procés a executar.

Respecte els algorismes de planificació

- Algorismes de planificació que funcionen bé en un entorn de càrrega computacional alta poden funcionar molt malament en entorns de amb càrrega entrada-sortida alta, i a la inversa.
- L'objectiu aquí és parlar d'algorismes que funcionen en bé en un entorn amb una càrrega variada². Són els que s'utilitzen en ordinadors personals.

²Tant processos amb càrrega de computació alta com processos amb càrrega alta d'entrada-sortida.

La planificació no té solució única. Cada política de planificació té les seves propietats desitjables i desavantatges. Veurem coses com:

- Planificació **uniprocessador**: algorismes bàsics
- Planificació **multiprocessador**: com canvia la planificació en cas de tenir múltiples processadors?
- Planificació d'**estalvi energètic**: com afecta la planificació en l'ús d'energia?
- Planificació d'**execució en temps real**: com executar processos si hi ha terminis?
- Planificació en **sistemes sobrecarregats**: com queda afectada la planificació en sistemes que reben moltes peticions?

Un concepte abans de continuar...

Tasca o treball a realitzar

- Un procés pot executar, al llarg de la seva execució, múltiples tasques diferents, cadascuna amb característiques diferents. Per exemple, calcular la forma d'una proteïna, desar dades a disc, dibuixar el menú a pantalla en fer clic amb el ratolí, o pulsar una tecla del teclat perquè es mostri a la pantalla del processador de textos.
- Típicament un procés pot estar format per múltiples fils d'execució; cada fil pot executar diverses tasques diferents³.
- En aquest tema parlarem a partir d'ara parlarem de tasca i no de procés.

³Introduïrem els fils al tema "Introducció a la concurrència".

L'algorisme bàsic és el **round robin** (o FIFO circular)

- El planificador disposa d'una llista de tasques i les executa de forma ordenada, des de la primera fins a l'última, de manera circular.
- Cadascuna de les tasques s'executen un temps màxim limitat, anomenat **llesca de temps**. Cada cop que finalitza la llesca el planificador fa un canvi de context a una altra tasca. La tasca també es pot executar menys temps.
- En fer un canvi de context es posa la tasca actual al final de la llista de tasques i s'agafa la primera tasca de la llista de tasques preparades.

Es crític la selecció del **valor** per a la llesca de temps

- Si la llesca de temps és massa petita hi haurà una sobrecàrrega de planificació: es “perdrà” molt de temps fent canvis de context i les tasques no podran fer feina. Hi haurà molta sobrecàrrega.
- Si la llesca de temps és massa alta, les tasques hauran d’esperar molt de temps per obtenir el seu torn i poder executar. El temps de resposta serà baix.

Anem a veure alguns detalls...

Planificació uniprocessador: round robin

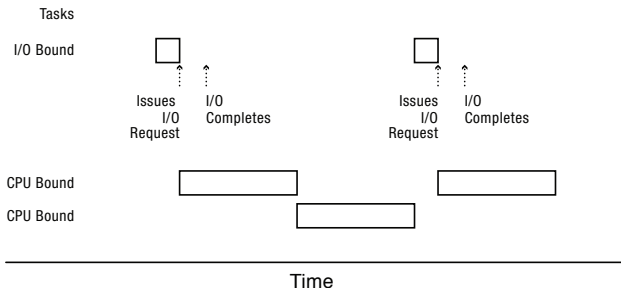
Respecte el canvi de context i la llesca de temps. Quin és el **cost** associat?

- En fer un canvi de context cal guardar els registres del procés actual que s'executa, seleccionar el nou procés i carregar a la CPU els registres del nou procés. Això és força ràpid de fer avui en dia: poques desenes de microsegons!
- Cal tenir en compte l'impacte en la memòria cau (*cache*): la tasca seleccionada per executar pot o no tenir dades a la cache de la seva llesca de temps anterior. Depèn del que han fet les altres tasques. Potser la tasca en execució haurà d'accedir a memòria RAM perquè es copiïn les dades a cache, i és molt lent!
- Hi ha doncs un compromís al valor de llesca de temps entre el temps de resposta i la sobrecàrrega. El temps de llesca s'estableix típicament entre **10 i 100 milisegons**.

Planificació uniprocessador: round robin

La planificació round robin generalment no és adequada en un entorn amb una càrrega variada. Suposem tres tasques:

- Dues amb alta càrrega computacional (*CPU bound*)
- Una tasca amb càrrega entrada-sortida (*I/O bound*) que a cada iteració només necessita la CPU per 1ms i 10ms de disc.



De la figura anterior, observar que

- Si la tasca “IO bound” s’executés sola podria tenir ben ocupat el disc i acabar la feina ràpid.
- Amb les altres dues tasques “CPU bound” i una llesca de temps de 100ms, la tasca “IO bound” s’alenteix pràcticament en un factor de 20! Cada cop que aquesta tasca té el processador només el necessita 1ms, però després s’ha d’esperar 200ms poder tornar a executar.
- Caldria “prioritzar” el procés “IO bound” perquè no s’hagi d’esperar tant...

La planificació round robin no és una bona solució hi si ha tasques “CPU bound” i “IO bound”.

- Les tasques amb càrrega alta en termes d'entrada-sortida generalment necessiten poc temps de CPU per emetre la següent operació d'entrada-sortida. Un cop emesa l'operació, es fa un canvi de context a una altra tasca.
- En un editor de text, es necessiten pocs mil·lisegons per mostrar a pantalla la tecla pulsada. Amb una llesca de temps de 100ms, l'editor pot haver d'esperar múltiples llesques de temps per executar i mostrar la tecla pulsada per pantalla: això serà molt molest per a l'usuari!

Quan va bé el round robin?

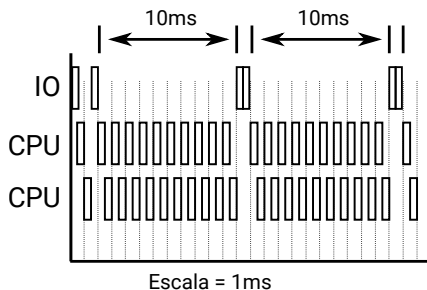
- La planificació round robin és adequada en certes circumstàncies si totes les tasques són del mateix tipus (en particular, “CPU bound”).
- Per exemple, en servidor de streaming de vídeo (o música): és molt important poder enviar tots els streams a una velocitat uniforme encara que el servidor envii les dades a una velocitat més alta de la que necessita el receptor.

Com es gestionar millor tasques “CPU bound” i “IO bound”? Amb l'algorisme **max-min fairness**!

Planificació uniprocessador: max-min fairness

L'algorisme **max-min fairness** va ser desenvolupat per a planificació de paquets de xarxa i després va ser adaptat a planificació de tasques. Per a planificació de tasques el funcionament és el següent:

- Intuïtivament, la idea és assignar en cada instant de temps la CPU a la tasca preparada que ha rebut menys temps de CPU.
- Suposem l'exemple anterior i que cada 0.5ms fem un canvi de context a la tasca que ha rebut menys temps de CPU.



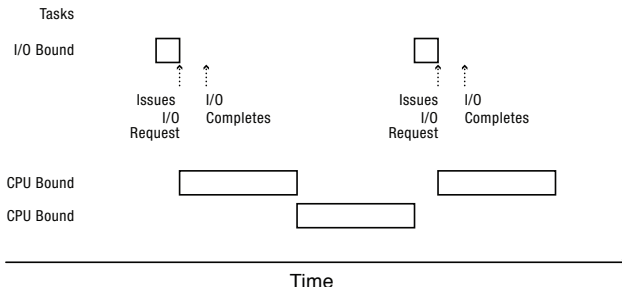
De l'exemple anterior, observar que

- El max-min fairness dóna a les tasques “petites” l'assignació que demanen i reparteix la resta entre les tasques “grans”. Els recursos són assignats en ordre creixent de demanda i cap tasca obté més recursos dels que demana.
- Si hi ha barreja de tasques (computacionals i entrada-sortida) resulta en una assignació més justa. Si només hi ha tasques computacionals és senzill: a cada tasca obté mateixa assignació de processador (és a dir, el round robin).

Planificació uniprocessador: max-min fairness

Fem alguns números amb round robin:

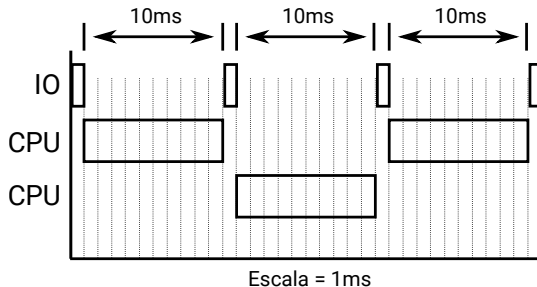
- Una llesca de 100ms, dues tasques computacionals i una d'entrada-sortida que només necessita 1ms de CPU i 10ms de disc.
- En 201ms s'executen les 3 tasques: a la tasca "IO bound" rep aproximadament un 0.5% de CPU mentre que les altres dues reben cadascuna pràcticament un 50% de la CPU.



Planificació uniprocessador: max-min fairness

Fem alguns números amb el max-min fairness

- En haver-se escrit les dades a disc es produeix una interrupció i el sistema operatiu pot decidir si fer un canvi de context.
- El max-min fairness assigna, idealment, aproximadament un 10% de CPU per a la tasca "IO bound" i 45% a les computacionals. Hi ha una assignació més justa!



Planificació uniprocessador: max-min fairness

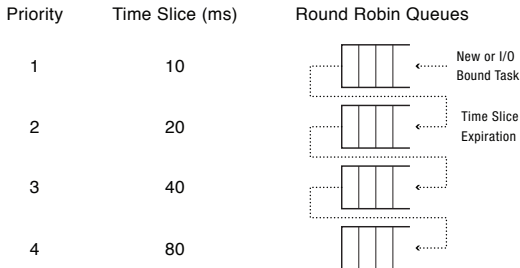
A la pràctica, com s'implementa el max-min fairness? Idealment...

- Cada cop que el planificador pot escollir una tasca a executar, escull aquella que té fins aquell moment el menor temps acumulat al processador.
- Requereix una cua de prioritats cosa i, per tant, implica un alt cost computacional: un servidor potser ha de prendre centenars o milers de decisions de planificació cada segon i no es convenient mantenir una cua de prioritats "ideal". Es presenta a continuació un algorisme simplificat...

Planificació uniprocessador: multi-level feedback queue

Sistemes operatius com Windows, Mac o Linux utilitzen l'algorisme **multi-level feedback queue** (MFQ) per planificar tasques

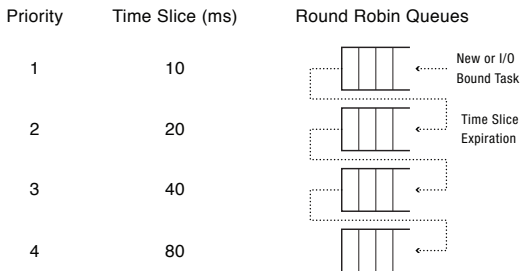
- Permet implementar una aproximació a l'algorisme d' max-min fairness fent servir una extensió del round robin.
- Al MFQ hi ha múltiples cues round robin: cada cua té una prioritat i una llesca de temps diferent de les altres cues .



Planificació uniprocessador: multi-level feedback queue

De forma senzilla, al MFQ

- Les tasques d'una cua del mateix nivell son planificades segons un round robin. Una tasca de prioritat alta s'executa abans que una de prioritat més baixa.
- Les tasques es mouen pel planificador entre les cues de prioritat per obtenir equitat. Si una tasca consumeix la seva llesca de temps (sense haver accedit a disc) es mou a una cua de prioritat més baixa. Si no ho fa, es mou amunt a la prioritat.



Polítiques de planificació

- Planificació **uniprocessador**: algorismes bàsics que hem vist
- Planificació **multiprocessador**: com canvia la planificació en cas de tenir múltiples processadors?
- Planificació d'**estalvi energètic**: com afecta la planificació en l'ús d'energia?
- Planificació d'**execució en temps real**: com executar processos si hi ha terminis?
- Planificació en **sistemes sobrecarregats**: com queda afectada la planificació en sistemes que reben moltes peticions?

Anem a estendre les idees de la planificació uniprocessador a la resta de casos.

A l'actualitat es habitual que un ordinador (o smartphone, o tablet) tingui **múltiples processadors**. Ens preguntem:

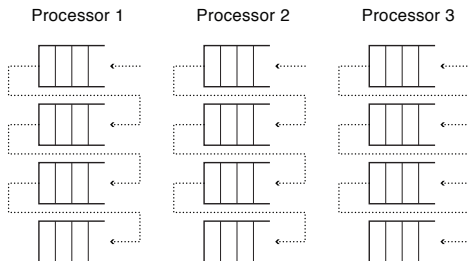
- Com podem utilitzar de forma efectiva múltiples processadors per executar tasques seqüencials?
- Com adaptem els algorismes de planificació uniprocessador a múltiples processadors?

Per fer tot això al tenir en compte que en sistemes multiprocessador cada processador té la seva pròxima memòria cau (*cache*).

Planificació multiprocessador

Els sistemes operatius com Windows, Mac o Linux cada processador té la seva pròpia estructura MFQ i, per tant, les seves pròpies tasques.

- Una tasca s'executa doncs sempre al mateix processador per aprofitar les dades emmagatzemades a la memòria cau d'aquest.
- Si un processador es queda sense res a executar, pot “robar” una tasca d'un altre processador.



Planificació d'estalvi energètic

L'impacte de la planificació en l'ús d'energia, tant en sistemes que utilitzen bateria com servidors, és important atès que gran part del seu cost hi està associat. L'estalvi energètic és possible gràcies al maquinari:

- **Disseny del processador:** un processador ràpid requereix una circuiteria complexa i gasta més energia; un processador lent és més senzill i pot reduir molt el consum energètic. Avui en dia alguns sistemes inclouen tots dos tipus de processadors deixant al sistema operatiu decidir quines tasques es planifiquen a cada processador.
- **Ús del processador:** es possible deshabilitar (apagar) un processador perquè no consumeixi energia o reduir la freqüència de rellotge del processador.
- **Dispositius d'entrada-sortida:** dispositius com la pantalla, el Wifi o la xarxa del telèfon pot consumir molta energia. Els dispositius que no s'utilitzen haurien d'apagar automàticament (i engegar quan faci falta).

En alguns sistemes el planificador ha d'assegurar que determinades tasques tenen **terminis** per executar-se. Per exemple,

- En un avió el programari associat a el control del vol ha d'executar-se abans determinats instants de tems perquè sigui efectiu.
- El programari d'un cotxe que controla el bloqueig dels frens o evita el control de tracció ha d'ocórrer a temps perquè sigui efectiu.
- En entorns menys crítics, com per exemple a la reproducció d'un vídeo a un ordinador, cada imatge ha de visualitzar-se quan toca per evitar una percepció visualment dolenta.

A l'actualitat existeixen diverses formes per intentar assegurar que es compleixen els terminis (però no ho asseguren al 100%!).

- **Sobre-aprovisionament**: les tasques de temps real han d'ocupar, globalment, només una fracció de la capacitat de processament del sistema. Les tasques de temps real seran planificades ràpidament sense haver d'esperar altres tasques.
- **Priorització per termini** (*earliest deadline first*): les tasques de temps real són prioritzades de forma que s'executen primer aquelles que tenen un termini més proper. Aquesta tècnica funciona si les tasques són computacionals (i.e. no necessiten entrada-sortida).

Què passa quan el sistema es sobrecarrega? És particularment important en sistemes web en què el nombre de connexions pot variar al llarg del dia...

- Molts sistemes operen sense un control directe de la seva càrrega i el seu rendiment pot baixar molt si hi ha sobrecàrrega.
- En una autopista no es gestiona correctament la sobresaturació de cotxes!
- En un restaurant amb molts clients (i pocs cambrers) ens haurem d'esperar molt si trigen molt a servir-nos. No hi tornarem pas al restaurant! El mateix passa amb els sistemes web. Què es pot fer?

Què passa quan el sistema es sobrecarrega? Una solució senzilla és **rebutjar peticions de connexió** que poden portar a la sobrecàrrega.

- Així poden assegurar un bon servei a les connexions existents. Aquest comportament és típic en sistemes web de streaming de vídeo.
- Al restaurant es tracta de no deixar entrar-hi per assegurar un bon servei als clients que són a dins.

Què passa quan el sistema es sobrecarrega? Una altra forma de gestionar la sobrecàrrega es **reduir el temps de resposta** per a determinades peticions! Per exemple,

- El 9/11 (atemptats a NY) la pàgina web de CNN va utilitzar una pàgina estàtica per informar i que és més ràpida de servir.
- Amazon utilitza un sistema en què es retorna una pàgina estàtica en cas la informació demanada no està disponible a temps.
- EBay actualitza menys freqüentment la llista de subhastes.
- Serveis de vídeo poden reduir la qualitat del vídeo.

En general, però, molts sistemes acostumen a fer el contrari: realitzen més feina per tasca a mesura que augmenta la càrrega!