laSalle ENG

Universitat Ramon Llull

Sistemes Operatius

Capítol 3: Planificació

Tema 3. Planificació

1. Introducció

La planificació de la CPU és la base dels SO amb multiprogramació. Commutant la CPU entre els processos en execució podem fer més productiu l'ordinador. De fet, l'objectiu de la multiprogramació és tenir un procés en execució en tot moment per aprofitar al màxim la utilització de la CPU.

Tot procés sol actuar de la següent manera: primer té una ràfega o sèrie (burst) de la CPU i després té una ràfega d'E/S. Aquests dos estats es van alternant durant tota l'execució, fins que hi ha una darrera ràfega de CPU per finalitzar l'execució i retornar el control al SO. Seria bo saber quant duren aquestes ràfegues per poder planificar bé els processos. Hi ha molts estudis i dependrà molt de la tipologia dels processos però en general podem concloure que hi ha un gran nombre ràfegues de CPU de curta durada i molt poques ràfegues de CPU de llarga durada.

Qui o quins mòduls de la CPU intervenen en aquest procés i quina funcionalitat tenen? Si som estrictes cal diferenciar:

- Scheduler o planificador: quan la CPU es queda inactiva és l'scheduler qui ha de seleccionar quin procés tindrà la CPU a partir d'aquell moment.
- Dispatcher: és el mòdul que s'encarrega de realitzar la commutació de processos però no la selecció.

La planificació es pot realitzar de dues maneres diferents:

a) Non Preemptive o no apropiativa o cooperativa: quan la planificació només es pot realitzar quan un procés passa de l'estat de execució a bloquejat (per exemple per culpa d'una sol·licitud d'E/S) o quan un procés acaba. És a dir, l'SO no pot intervenir per treure la CPU a un procés.
b) Preemptive o apropiativa: quan el SO pot forçar un canvi en la planificació (per exemple amb una interrupció de temporitzador o una d'E/S).

2. Criteris de planificació

Per poder avaluar els diversos algorismes de planificació que passarem a veure, definirem un conjunt de criteris a tenir en compte:

a) Utilització de la CPU: màxima ocupació de la CPU (segons estudis hauria d'estar entre un 40% i un 90%).
b) Rendiment o throughput: nombre de treballs acabats per unitat de temps. Evidentment dependrà del tipus de tasques a realitzar però és una bona mesura per fer comparatives.
c) Temps de lliurament: temps que va des de que es demana un procés fins que aquest acaba.
d) Temps d'espera: suma de tots els temps que el procés ha estat esperant sense tenir la CPU.
e) Temps de resposta: temps que va des de la presentació de la sol·licitud fins que dóna una primera resposta.
f) Equitativitat.

Cal dir també que habitualment per avaluar criteris usem comparatives amb les mitjanes aritmètiques dels temps dels processos i intentem optimitzar aquestes; però alguns cops potser és interessant optimitzar els valors màxims o mínims enfront de les mitjanes, per garantir un bon servei a tots.

3. Algorismes de planificació bàsics

A) FCFS (First Come, First Served)

Quan la CPU està disponible aquesta s'atorga respectant l'ordre de petició.

És no apropiatiu.

Implementació: amb una cua FIFO.

La mitjana del temps d'espera depèn de l'ordre de petició d'execució.

Exemple: P1(24), P2(3), P3(3). Si peticions arriben P1, P2 i P3 temps d'espera mitjà serà t=(0+24+27)/3=17 ms. Si fos P2, P3 i P1 llavors t=(0+3+6)/3=3 ms.

A més, no és gens eficient quan es barregen processos de CPU amb processos d'E/S (efecte convoy).

B) SJF (Shortest Job First)

Quan la CPU està disponible, el planificador dona la CPU al procés que té la següent ràfega de CPU més petita. En cas d'igualtat s'aplica FCFS.

Exemple: P1(6),P2(8),P3(7),P4(3). SJF=P4,P1,P3,P2 i t(SJF)=(0+3+9+16)/4 = 7 ms però si uséssim t(FCFS)= (0+6+14+21)/4 = 10.25 ms.

L'algorisme SJF minimitza el temps d'espera mitjà.

Problema: com estimem temps de la propera ràfega? Impossible. Què fer? Doncs si no podem conèixer el temps per la següent ràfega podem predir-lo o estimar-lo.

Mètode de predicció: mitjana exponencial de les longituds mesurades de les ràfegues prèvies.

$$T_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot T_n \quad \alpha \in [0,1]$$

Exemple: agafar α=1/2 i T_0=10 per 6,4,6,4,13,13,13,... Dona 8,6,6,5,9,11,12

Pot ser apropiatiu o no apropiatiu. Quan estem executant un procés i un altre fa la seva petició és llavors quan hem de decidir si el SO pot treure-li o no la CPU si troba que la propera ràfega del procés nou és menor que la del procés propietari de la CPU. Exemple:

Procés	Moment d'arribada	Temps de ràfega
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Si fem apropiatiu: P1-P2-P4-P1-P3 i t(SJF)=6.5 ms
Si fem no apropiatiu: t(SJF)=7.75 ms

També cal definir què vol dir temps més curt: el temps original del procés o el temps que li resta pendent en aquell moment? Existeixen les dues versions de l'algorisme.

Habitualment hi ha dues variants més modernes d'aquest algorisme:

- SPN (Shortest Process Next): és el SJF no apropiatiu tenint en compte tot el temps total del procés.
- SRT (Shortest Remaining Time): és el SJF apropiatiu tenint en compte el temps que li resta al procés en cada instant.

C) Planificació amb prioritat

De fet, l'algorisme SJF és un cas particular d'un algorisme de planificació amb prioritat. Cada procés té assignada una prioritat i la CPU s'assigna al procés amb prioritat més alta. En cas d'igualtat s'usa FCFS.

Exemple:

Procés	Prioritat	Temps de ràfega
P1	3	10
P2	1	1
P3	3	2
P4	4	1
P5	2	5

P2-P5-P1-P3-p4 i t_espera=8.2 ms

Les prioritats poden ser assignades internament o externament. Si es realitza internament es tenen en compte certs paràmetres que caracteritzen els processos, com mida, nombre de fitxers oberts, etc. Si es fa externament al SO es tenen en compte criteris més polítics o organitzatius que d'eficiència.

La planificació per prioritats por ser apropiativa o no. Un algorisme de prioritats apropiatiu expropiarà la CPU al procés que la tingui si arriba un procés amb més prioritat que el que es troba en execució. En canvi, si fos no apropiatiu es limitaria a posar-lo a la cua de prioritats al lloc que li pertoqui.

Un problema important d'aquest tipus d'algorismes és el bloqueig indefinit o inanició. Pot passar que en un sistema d'alta càrrega de treball, els processos amb menys prioritats estiguin

esperant eternament perquè els de més prioritats sempre se'ls colen. Explicar llegenda urbana del MIT (apagada del IBM 7094 al 1973 amb un procés pendent d'execució des del 1967). Una solució per aquest problema és l'envelliment (*aging*): augmentar la prioritat dels processos que porten molta estona a la cua.

D) Planificació RR (Round-Robin)

Va ser dissenyat especialment per sistemes de temps compartit. És similar a la FCFS però afegint apropiació per commutació de processos.

El temps d'espera mitjà és alt. Exemple comparatiu:

P1(24), P2(3), P3(3). Si peticions arriben P1, P2 i P3 en el instant 0 i tenim un quantum de 4 ms llavors

P1(4) P2(3) P3(3) P1(4) P1(4) P1(4) P1(4) P1(4) amb t_espera = 5.66 ms

El rendiment de l'algorisme RR depèn força del valor del quantum. Si el quantum és molt gran convertim l'algorisme en un FCFS. Si el quantum és massa petit llavors perdrem molt de temps ja que els canvis de context tenen una despesa de temps a considerar.

El temps de lliurament també depèn de la mida del quantum. Però no és cert que en augmentar el quantum disminueixi necessàriament el temps de lliurament. En general si que es redueix si els processos acaben les seves ràfegues de CPU amb un sol quantum. Però no oblidem que un quantum massa gran ens passa a FCFS. Una regla empírica és que el 80% de les ràfegues de CPU d'un procés han de ser més curtes que el quantum.

E) Planificació HRRN (Highest Response Ratio Next)

És un algorisme no apropiatiu per definició. Defineix una tasa de resposta que correspon a la següent fórmula:

$R = (t_{espera}+t_{CPU})/ t_{CPU}$

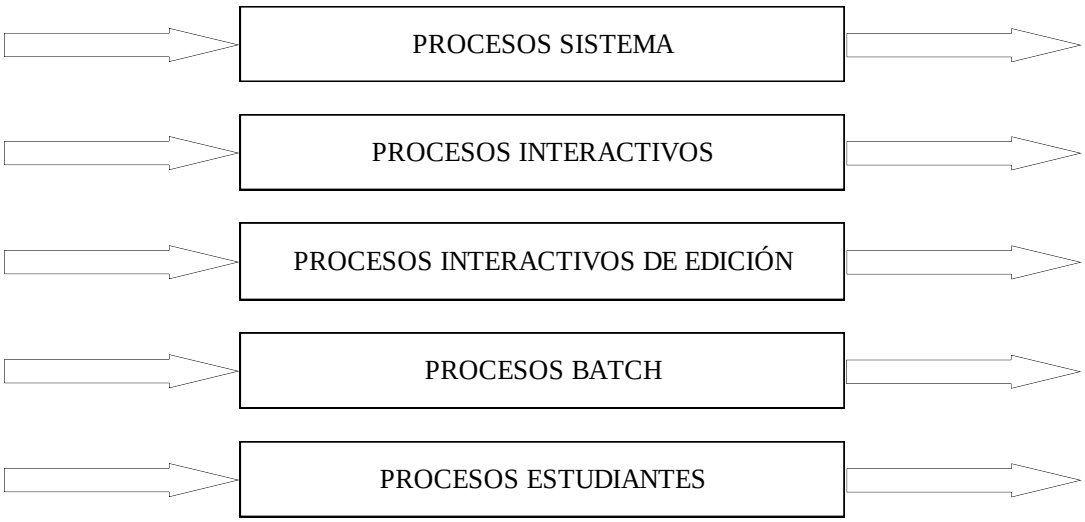
Aquest algorisme pretén prioritzar, igual que els SJF, els processos curts, però aconsegueix un bon equilibri de processos i evita la inanició possible de processos ja que si aquests porten molta estona esperant avançaran a processos que siguin més curts que ell degut a que augmentarà el temps d'espera i en conseqüència la ratio de resposta.

4. Algorismes de planificació evolucionats

A) Planificació amb cues multinivell

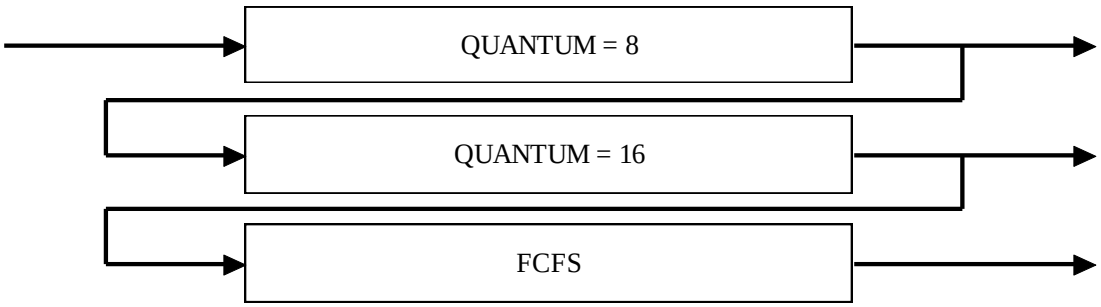
Molts cops hi ha processos que tenen necessitats molt diferents. Per exemple els processos interactius i els processos batch.

Un algorisme de planificació amb cues multinivell divideix la cua de processos en execució en diverses cues separades. Els processos s'assignen a alguna d'aquestes cues en base a alguna propietat dels mateixos (mida del procés, tipus de procés, etc.). Cada cua te el seu propi algorisme de planificació. Addicionalment, hi ha una planificació entre les cues, que habitualment és una planificació per prioritat fixa apropiativa. També hi ha la possibilitat d'assignar porcions de temps de CPU a cada cua.



B) Planificació amb cues multinivell amb retroalimentació

Assignar un procés a una cua permanentment és pot flexible. Per això s'incorpora una mobilitat entre cues en l'algorisme anteriorment presentat.



Així, caldrà definir els següents paràmetres en una planificació amb cues multinivell i retroalimentació:

- Número de cues.
- Algorisme de planificació per a cada cua.
- Criteri per determinar quan un procés passa a una cua de prioritat més gran.
- Criteri per determinar quan un procés passa a una cua amb prioritat més petita.
- Criteri per decidir quina cua se li assigna a un procés nou.

5. Planificació amb processadors múltiples

Primer de tot, cal dir que parlar de processadors múltiples pot portar a error ja que existeixen diverses arquitectures possibles i cadascuna té uns requeriments de planificació diferent. Concretament tenim:

- Multiprocessador centralitzat (fortament acoblat): és una màquina amb 2 o més CPUs però amb una única memòria i E/S. És a dir, la memòria i perifèrics són compartits per a tots els processadors i es troben sota el control d'un únic SO.
- Multiprocessador distribuït (dèbilment acoblat o cluster): és un conjunt de màquines interconnectades, on cadascuna d'elles té la seva pròpia memòria i dispositius d'E/S.

La primera estratègia va ser tenir una cua de processos a executar per a cadascun dels processadors. Però aquesta solució no permet balanceig de càrrega i potser alguna CPU estaria ociosa mentre que altres cues estarien sobrecarregades de treball.

Per evitar aquest problema es va optar per una cua única de processos, que es planifiquen cap a qualsevol CPU disponible. Aquí, ens trobem amb dues alternatives:

a) SMP o Symmetric MultiProcessing o multiprocessament simètric: cada processador examina la llista comú de processos a executar i en selecciona un per a la seva execució. Cal recordar que com que la llista és un recurs compartit caldrà posar els mecanismes per assegurar que un mateix procés no sigui escollit per dos CPUs alhora.
b) Multiprocessament asimètric: per evitar-nos problemes de compartició, la segona alternativa fa que totes les decisions de planificació, de processament de l'E/S i altres activitats del sistema les realitzi un únic processador (el processador mestre) mentre que la resta reben ordres d'aquest (processadors esclaus).

6. Planificació en temps real

Els sistemes en temps real (sobretot els anomenats hard real time o sistemes de temps real estricte) es caracteritzen perquè els seus processos i/o threads (fils) han d'haver-se acabat abans d'un cert temps límit (deadline). La mesura crítica és doncs si el sistema serà capaç d'assolir la planificació dels temps límits. La resta de mesures queden en un segon pla i acostumen a ser irrellevants.

Les tasques que s'executen en els sistemes de temps real es poden classificar en crítiques (hard real time) y no crítiques. En sistemes crítics es poden usar 2 algorismes de planificació: planificació cíclica y planificació monòtona en freqüència. En sistemes no crítics tenim el EDF (Earliest Deadline First).

A) Planificació cíclica

La planificació cíclica exigeix un determinat model de tasques amb les següents restriccions: totes les tasques s'executen cíclicament i les tasques no interaccionen entre sí. A més a més, el conjunt de tasques ha de ser conegut a priori.

Procés	Temps execució (C)	Termini de Resposta (D)	Període d'Activació (T)
P1	1	5	5
P2	2	10	10
P3	2	20	20

L'execució consisteix en la repetició d'un conjunt de tasques de manera indefinida (Cicle Principal o CP). Cal definir el Factor d'Utilització que serveix per a comprovar que el processador pot executar els treballs:

$$U=\sum(C_i/T_i)<1 \quad i=1..n$$

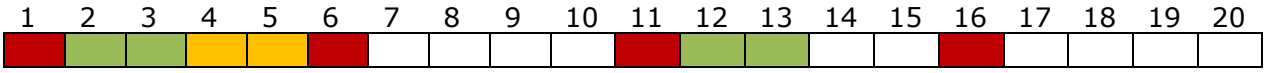
La planificació consisteix en ajustar totes les tasques dins del cicle principal (CP). Per aconseguir això és calcula un divisor del cicle principal anomenat cicle secundari (CS) i llavors es tracta d'encaixar les execucions en el cicle secundari. El problema té una complexitat exponencial així que no és possible solucionar-lo amb mètodes computacionals. Es fa servir alguna heurística per a simplificar el problema.

La solució més senzilla s'obté quan els períodes dels processos són armònics. Llavors es pot demostrar que:

$$CP=\max(T_i) \quad i=1..n$$
$$CS=\min(T_i) \quad i=1..n$$

Si apliquem el mètode a l'exemple:

U= 2/20+2/10+1/5 = 0.1+0.2+0.2 = 0.5 < 1
CP=max(5,10,20)=20
CS=min(5,10,20)=5



P1: granate P2: verd P3:groc

B)Planificació monòtona en freqüència

El 1973, Liu i Layland van crear un algorisme anomenat RMS (Rate-Monotonic Scheduling) que funciona correctament si tenim N tasques cícliques, amb prioritats estàtiques i on la periodicitat coincideix amb el deadline. Demostren també que hi haurà una planificació factible sempre que:

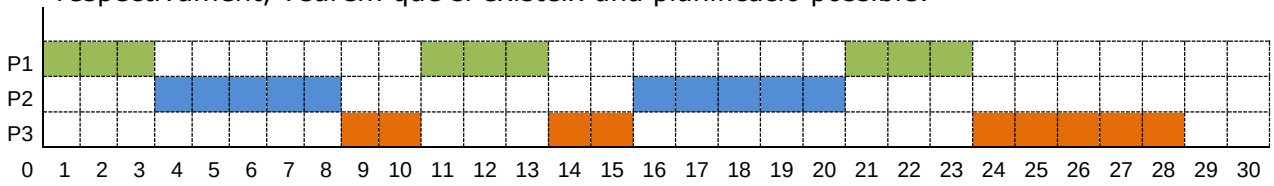
$$U=\Sigma(C_i/T_i)\leq N(2^{1/N}-1)\ i=1..N$$

Exemple:

Procés	P1	P2	P3
Periodicitat (ms)	10	15	30
T_{CPU}	3	5	9

Si fem el sumatori veiem que dona 0.93 (menor que 1), cosa que ens diu que si el temps de despesa de CPU és petit pot haver-hi una planificació però no la garanteix, ja que no és menor de $3(2^{1/3}-1)=0.77$.

Si assignem de prioritats: P1=1/0.010, P2=1/0.015 i P3=1/0.030, és a dir 100, 67 i 33 respectivament, veurem que si existeix una planificació possible:

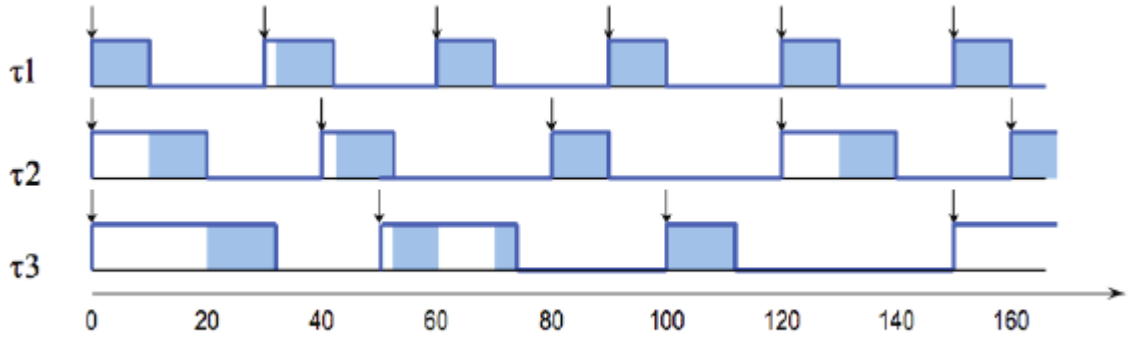


C)Planificació per temps límit (EDF – Earliest Deadline First)

Aquest algorisme exigeix conèixer a priori també el deadline de cada procés. És un algorisme que funciona amb tasques periòdiques que poden perdre la CPU (preemptive) si el deadline d'un altra tasca programada és més propera.

Exemple:

Procés	Temps execució (C)	Termini de Resposta (D)
P1	10	30
P2	10	40
P3	12	50



7. Exemples d'implementació

LINUX

La **versió 1.0.9.** de Linux hi havia una funció del nucli anomenada schedule(). Aquesta funció es cridava des d'altres funcions del SO i després de cada crida al sistema i de cada interrupció. Cada cop que aquesta era cridada, realitzava un conjunt de tasques periòdiques (com processar els senyals pendents), inspeccionava els processos de la llista de preparats i n'escollia un per ser executat segons la política de planificació i després passava al dispatcher per fer el canvi de context. Així fins a una nova interrupció. La política de planificació és una variant de la RR. Fixa un quantum com a límit de temps de CPU a usar si altres processos esperen per aquest recurs. A més, es calculava una prioritat dinàmica basada en el temps que el procés havia estat esperant la CPU.

La **versió 2.6** de LINUX (24 de maig de 2004 apareix la primera versió de 2.6) inclou la capacitat de planificació en temps real i un planificador substancialment canviat respecte les anteriors per processos no de temps real.

Existeixen 3 classes de planificació:

- SCHED_FIFO: threads de temps real amb FIFO
- SCHED_RR: threads de temps real amb Round Robin
- SCHED_OTHER: Altres threads no de temps real.

Dins de cada classe poden utilitzar-se prioritats múltiples, però les prioritats de temps real (0..99) sempre són més grans que les de la classe SCHED_OTHER (100..139).

Per als threads FIFO se segueixen les següents regles:

1. El SO no interromp un thread FIFO en execució exceptuant:

a) Un thread de més prioritat passa a estat de Ready
b) Un thread en execució es bloqueja per esperar algun event.
c) Un thread en execució cedeix voluntàriament la CPU usant la primitiva sched_yield.

2. Quan un thread és interromput se'l situa a una cua associada a la seva prioritat.
3. Quan un thread passa a Ready i és de més prioritat que el thread actual Running, aquest és expulsat i s'escull el thread de prioritat més gran. Si n'hi ha més d'un llavors s'escull el més antic.

Per als threads RR l'estratègia és la mateixa però afegint un quantum de temps (round robin).

Els threads de la classe SCHED_OTHER només s'executen si no hi ha cap thread de les altres dues pendents. Els planificadors Linux de la 2.4 no funcionaven bé per un nombre creixent de processadors i processos. El nou planificador s'anomena O(1). Aquest planificador ha estat dissenyat per aconseguir que el temps de selecció del procés adequat sigui constant, independentment del nombre de processadors i de la càrrega del sistema.

A cada tasca no de temps real se li assigna una prioritat inicial entre 100 i 139 (per defecte 120). A mesura que s'executa la tasca es recalcula la prioritat en funció de la prioritat inicial i el seu comportament d'execució. Linux intenta afavorir els processos limitats per E/S que els limitats per CPU. Això ho aconsegueix guardant, per a cada procés, el temps que ha estat adormit (esperant un event) i els temps que ha estat en execució. Els quantums que assigna LINUX van entre els 10 ms i els 200 ms.

UNIX BSD

Aquest SO usa una aproximació de cua de nivells múltiples retroalimentada, amb 32 cues d'execució. Es processos del sistema usen les cues des de la 0 a la 7 i els processos d'usuari usen les cues des de la 8 a la 31. L'scheduler selecciona un procés de la cua de més prioritat quan ha d'assignar un procés a la CPU. Dins d'una cua usa RR i només els processos de la cua no buida de més prioritat poden executar-se. Els quantums són tots menors de 100 microsegons.

Cada procés té una prioritat nice() externa que s'usa per influir sobre quina cua ha d'executar-se el procés quan ho sol·licita, però no la determina únicament. La prioritat té inicialment un valor de 0 però pot ser canviada amb una crida al sistema. El valor de prioritat pot variar entre -20 i 20, on -20 és la prioritat més alta. A cada quantum el SO recalcula les prioritats en funció de la seva prioritat i de la sol·licitud recent de la CPU per part del procés (a més demanda recent menys prioritat). L'scheduler és cridat en els mateixos casos que en Linux.

Windows NT/2000/XP

El planificador de threads de Windows NT/2000/XP és un planificador de cues múltiples retroalimentat que pretén proporcionar alts nivells de servei a threads o processos que necessiten una resposta ràpida. El quantum va entre 20 i 200 ms. En servidors es configura el quantum amb un valor 6 vegades superior.

El scheduler suporta 32 nivells de planificació diferents. Les 16 primeres cues s'anomenen *cues de nivell de temps real*, les 15 següents s'anomena *cues de nivell variable* i la darrera és la *cua del sistema*.

El planificador intenta limitar l'entrada a les cues de temps real, així augmenta la probabilitat que hi ha menys competència i les tasques puguin enllestir-se ràpidament. Però, en no ser un sistema de temps real, Windows XP no pot garantir cap deadline pels seus processos o threads.

La cua de sistema conté un únic procés (el system idle process) que és el que s'executa quan no hi ha ningú al sistema amb peticions de ràfegues de CPU. Això segueix així fins que un altre procés sol·licita la CPU per entrada al sistema o desbloqueig del mateix.

El planificador és totalment apropiatiu. Això pot provocar en sistemes monoprocessador que un procés es faci fora a si mateix si llença a un altre procés o thread de prioritat més gran. Aquesta funcionalitat es més útil i subtil en sistemes amb més d'una CPU.

Bibliografia

- Silberschatz, A. Galvin, P. Gagne, G. *Sistemas Operativos*, Ed. Limusa, 2002 ISBN 968-18-6168-X, pàgines 135-171
- Peterson, J.L. Silberschatz, A. *Sistemas Operativos*, Ed. Reverté 1989 ISBN 84-291-2693-7, pàgines 111-151
- Nutt, G. *Sistemas Operativos*, Ed. Pearson Addison Wesley, Pearson Educación, Madrid, 2004, ISBN 84-7829-067-2, pàgines 232-261
- Stallings, W. Sistemas Operativos, 5ª Edición, Ed. Pearson Prentice Hall, 2005, ISBN 84-205-4462-0. Pàgines 402-487.
- Carretero, J. García, F. De Miguel, P. Pérez, F. sistemas Operativos, Ed.Mc Graw-Hill, 2ª Edición, 2007, ISBN 978-84-481-5643-5, pàgines 153-216