LINGI 1113: Systèmes informatiques 2

Mission 5: ordonnancement

ÉCOLE
POLYTECHNIQUE
DE LOUVAIN

UCL

Ordonnancement

Ordonnancement des uniprocesseurs

- Long terme: démarrer
- Moyen terme: swapper
- Court terme: allouer processeur

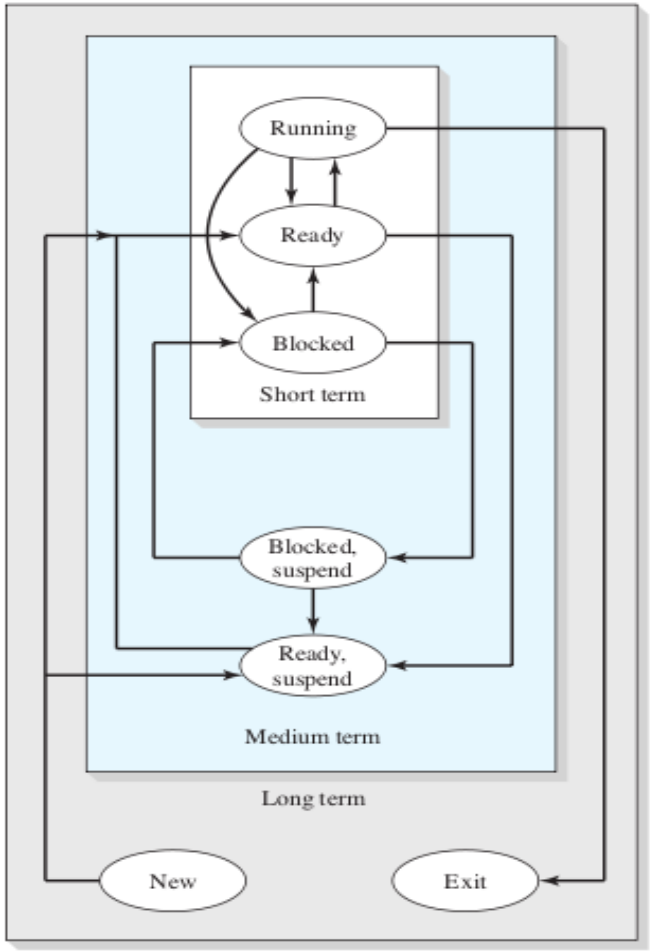


Figure 9.2 Levels of Scheduling

Ordonnancement

Données intervenant dans le choix d'un processus/thread

- Durée totale d'exécution (« turnaround time »)
- Temps de réponse (procs interactifs)
- Deadlines (procs. Temps réel)
- Prédictabilité
- Productivité (« troughput ») (batch)
- Taux d'utilisation du processeur (batch)
- Équité entre les utilisateurs (« fairness »)
- Discernement, privilégier l'important
- Equilibrage de l'usage de ressources (« balancing »)

Ordonnancement

Politiques d'ordonnancement

Table 9.3 Characteristics of Various Scheduling Policies

	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s − e]	$\max\left(\frac{w+s}{s}\right)$	(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

First -come-first-served,Shortest process next;Shortest remaining time;

Highest response ratio next; feedback (lower prio of preempted processes

Ordon

Politiques d'ordonnancement: un exemple

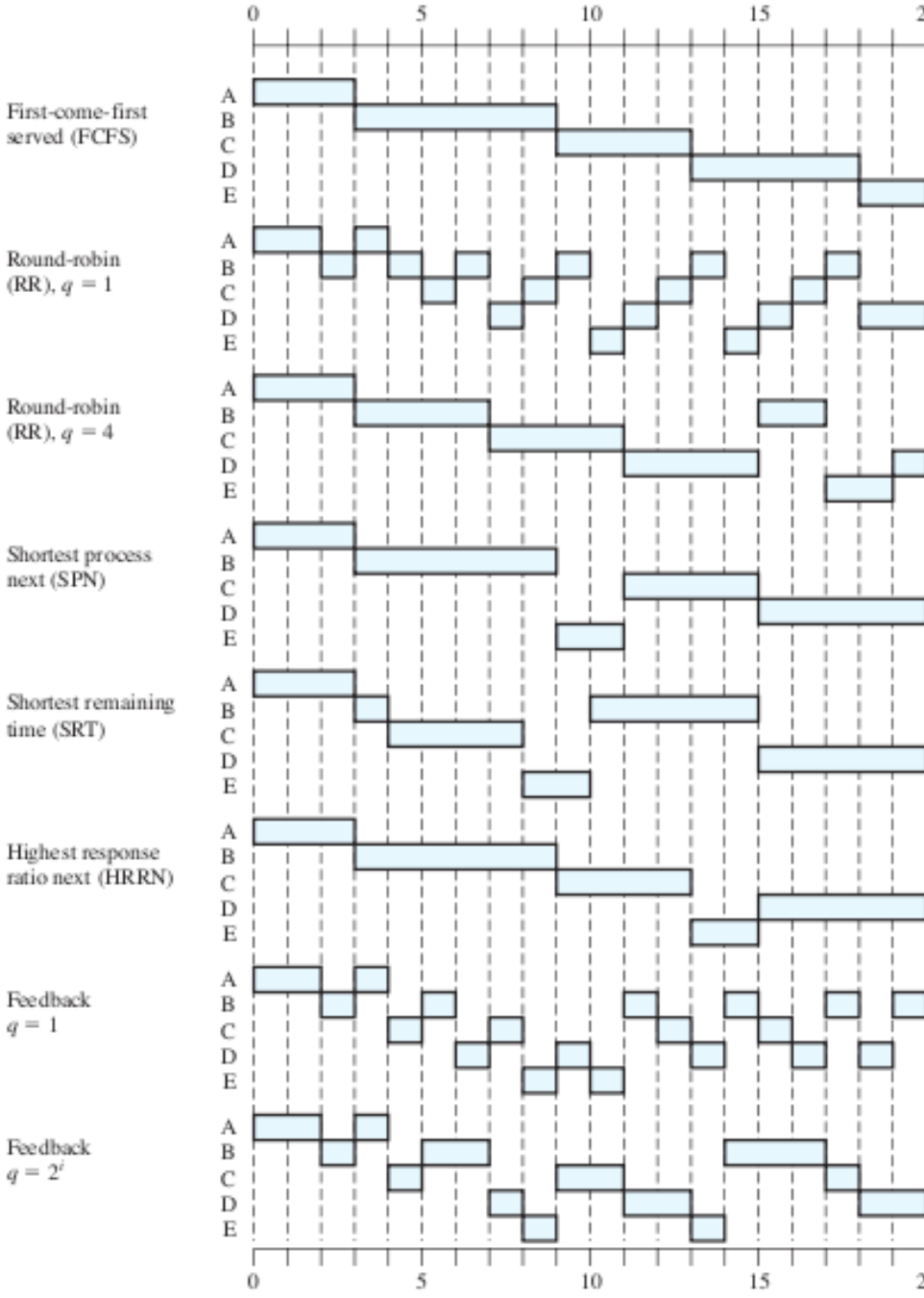


Figure 9.5 A Comparison of Scheduling Policies

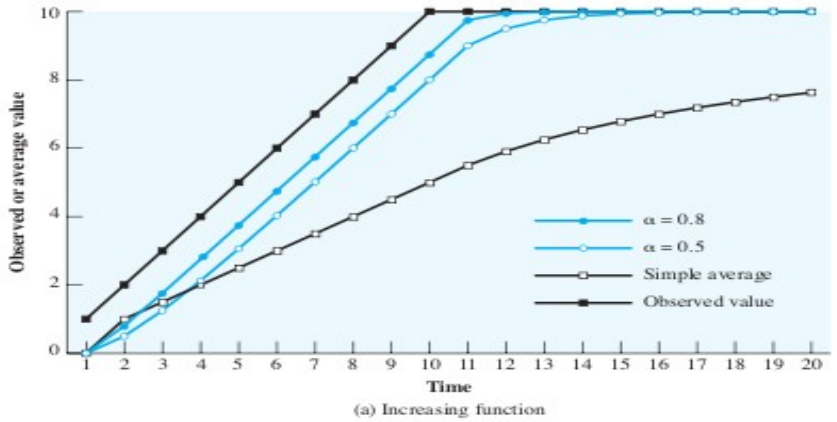
Ordonnancement

Politiques d'ordonnancement: favoriser processus courts:

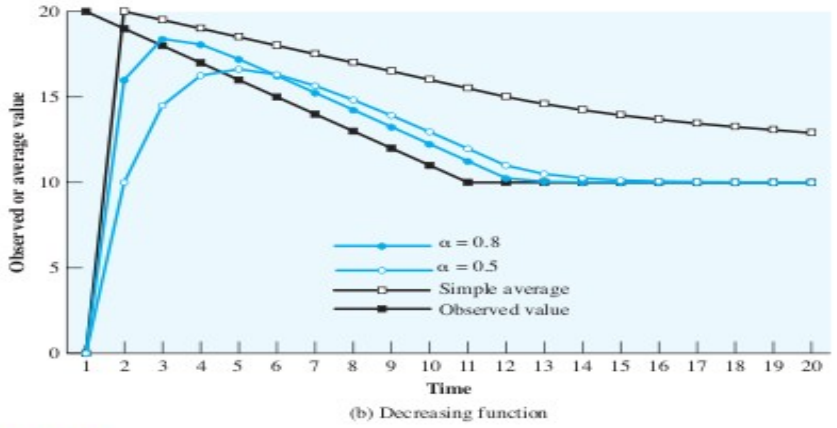
'Shortest process next' : effet du lissage exponentiel

$S_{n+1} = \alpha T_n + (1 - \alpha) S_n$

Soit on connait tous les T_n d'avance, soit feedback: on mesure le dernier



(a) Increasing function



(b) Decreasing function

Figure 9.9 Use of Exponential Averaging

Ordonnancement

Politiques d'ordonnancement: favoriser processus courts:

Priorité aux traitements courts (i/o plutôt que calcul)

Exemple:

- 2 classes (courts et longs)
- Autant de courts que de longs
- Longueur= 1 ou 5

tous les procs:

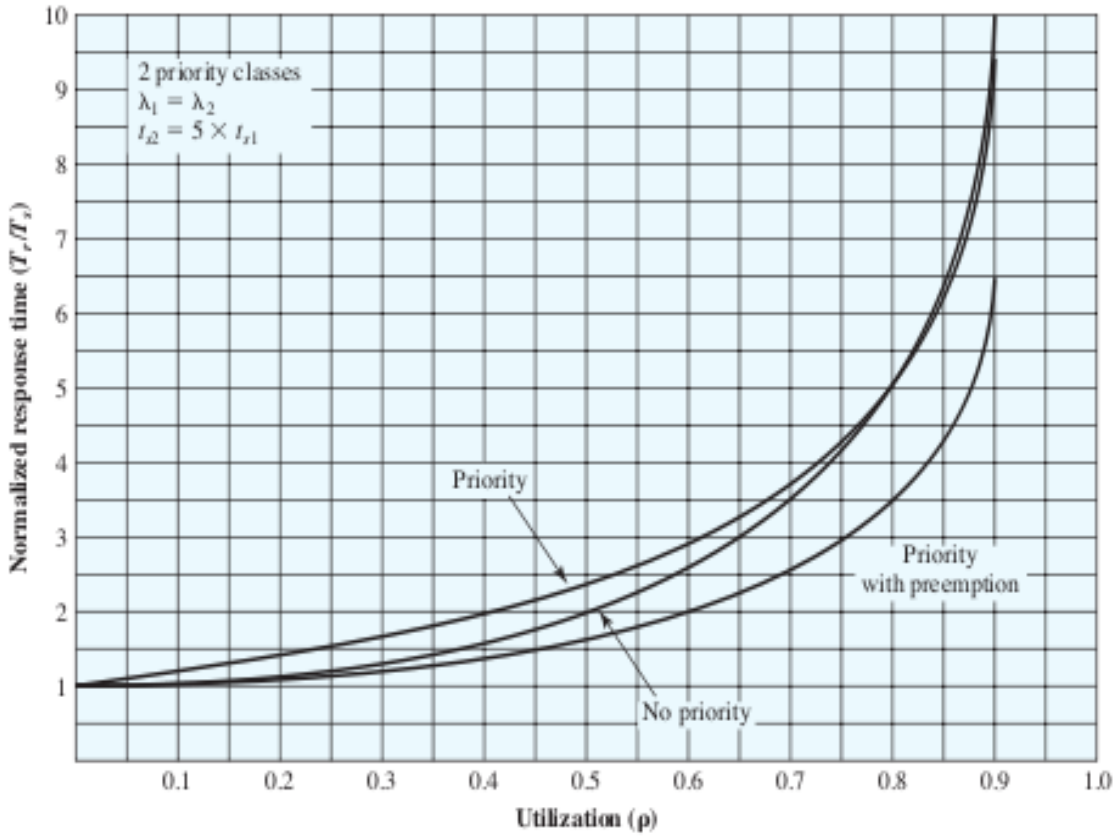


Figure 9.11 Overall Normalized Response Time

Ordonnancement

Politiques d'ordonnancement: favoriser processus courts:

Priorité aux traitements courts

Exemple:

- 2 classes (courts et longs)
- Autant de courts que de longs
- Longueur= 1 ou 5

rien que les courts:

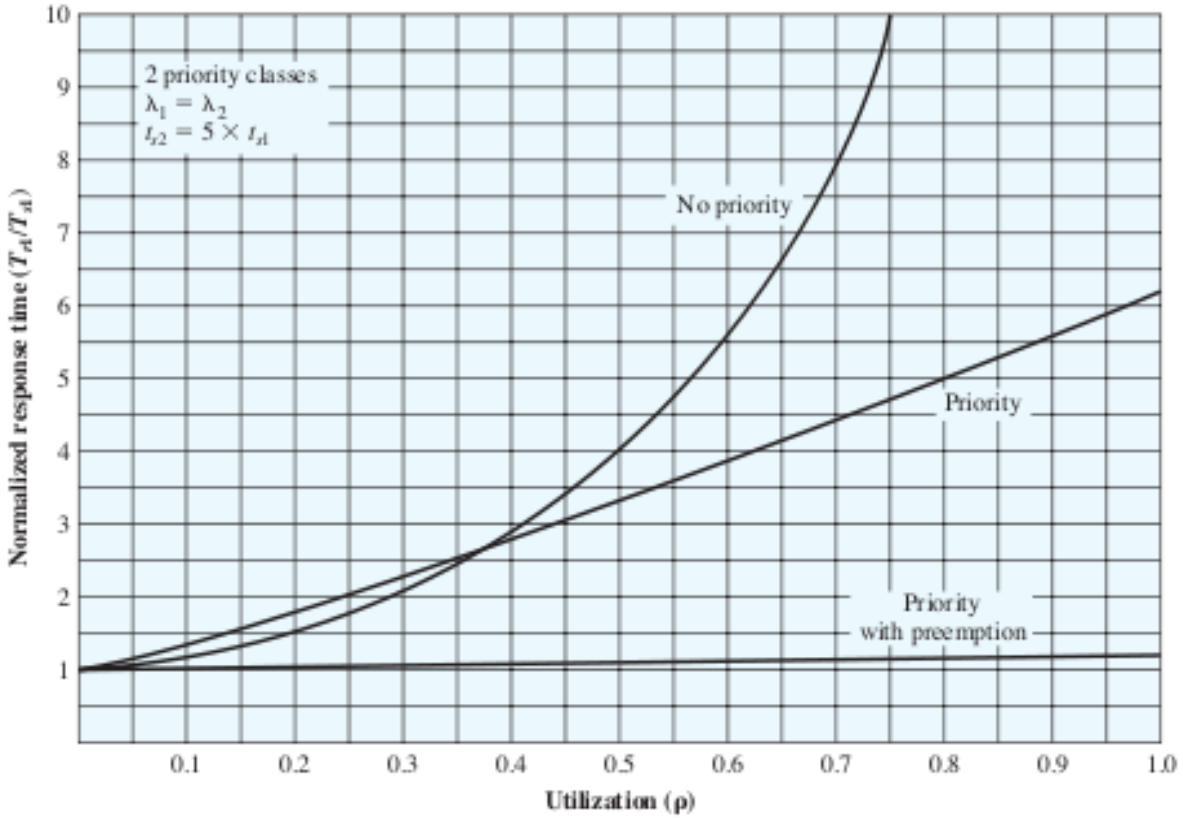


Figure 9.12 Normalized Response Time for Shorter Processes

Ordonnancement

Politiques d'ordonnancement: favoriser processus courts:

Priorité aux traitements courts

Exemple:

- 2 classes (courts et longs)
- Autant de courts que de longs
- Longueur= 1 ou 5

rien que les longs:

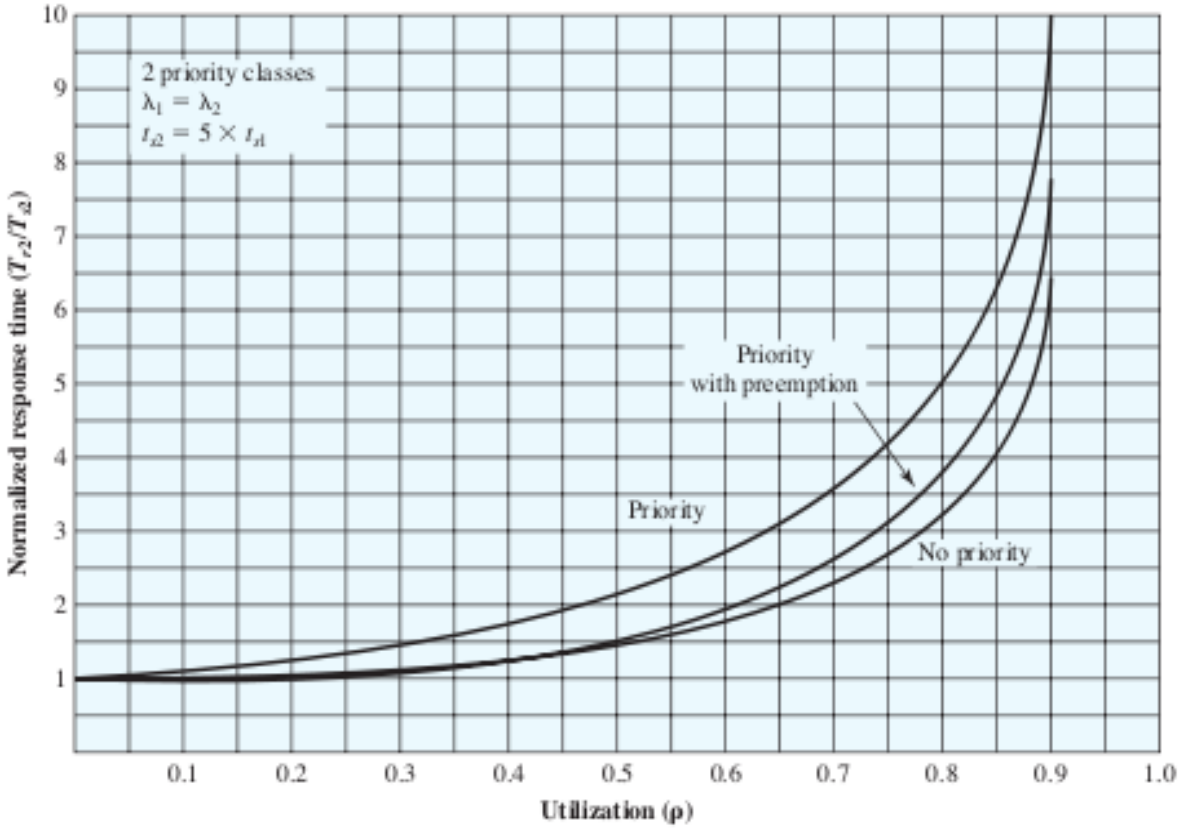


Figure 9.13 Normalized Response Time for Longer Processes

Ordonnancement

Politiques d'ordonnancement: « Fair share »

Être équitable entre utilisateurs: pénaliser processus qui utilisent le processeur (privilégie interactifs)

Idem feedback mais on tient compte des autres procs de l'utilisateur (groupe de procs)

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times W_k}$$

where

- $CPU_j(i)$ = measure of processor utilization by process j through interval i
- $GCPU_k(i)$ = measure of processor utilization of group k through interval i
- $P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities
- $Base_j$ = base priority of process j
- W_k = weighting assigned to group k, with the constraint that $0 < W_k \leq 1$ and $\sum_k W_k = 1$

Ordonnancement

Politiques d'ordonnancement: UNIX traditionnel: exemple de priorité aux traitements courts

Plus petite valeur de priorité = plus prioritaire

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

where

$CPU_j(i)$ = measure of processor utilization by process j through interval i

$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities

$Base_j$ = base priority of process j

$nice_j$ = user-controllable adjustment factor

(quelques procs systèmes comme le swapper sont plus prioritaires que les procs utilisateur qui ne sont jamais autorisés à entrer dans la gamme de priorité des procs système)

Ordonnancement

Multiprocesseurs

- Cluster (système réparti, couplage lâche)
- Multiprocesseurs à mémoire partagée:
  - Hiérarchiques (processeurs spécialisés: p. ex. carte écran)
  - Symmétriques: multi-coeurs etc: ce qui nous intéresse ici

Table 10.1 Synchronization Granularity and Processes

Grain Size	Description	Synchronization Interval (Instructions)
Fine	Parallelism inherent in a single instruction stream.	<20
Medium	Parallel processing or multitasking within a single application	20–200
Coarse	Multiprocessing of concurrent processes in a multiprogramming environment	200–2000
Very Coarse	Distributed processing across network nodes to form a single computing environment	2000–1M
Independent	Multiple unrelated processes	not applicable

superscalaires

multithread

multi processus

Ordonnancement

Multiprocesseurs symmétriques: ordonnancement

- Assignation permanente des threads aux processeurs
  - un processeur peut être inactif si tous ses threads sont bloqués
  + bonne exploitation de la cache
- Quand un processeur est libre il prend le thread le plus prioritaire, (les threads peuvent passer d'un processeur à l'autre
  + un processeur n'est jamais inactif s'il reste qq chose à faire dans le machine
  - pas terrible pour les caches (en gén chaque proc. a la sienne)

Ordonnancement

Multiprocesseurs symmétriques: ordonnancement

- **Load sharing:** Processes are not assigned to a particular processor. A global queue of ready threads is maintained, and each processor, when idle, selects a thread from the queue. The term **load sharing** is used to distinguish this strategy from load-balancing schemes in which work is allocated on a more permanent basis (e.g., see [FEIT90a]).[2]
- **Gang scheduling:** A set of related threads is scheduled to run on a set of processors at the same time, on a one-to-one basis.
- **Dedicated processor assignment:** This is the opposite of the load-sharing approach and provides implicit scheduling defined by the assignment of threads to processors. Each program is allocated a number of processors equal to the number of threads in the program, for the duration of the program execution. When the program terminates, the processors return to the general pool for possible allocation to another program.
- **Dynamic scheduling:** The number of threads in a process can be altered during the course of execution.

Ordonnancement

Multiprocesseurs symmétriques: choisir le nombre de threads: exemple: 16 processeurs, 2 programmes multithread

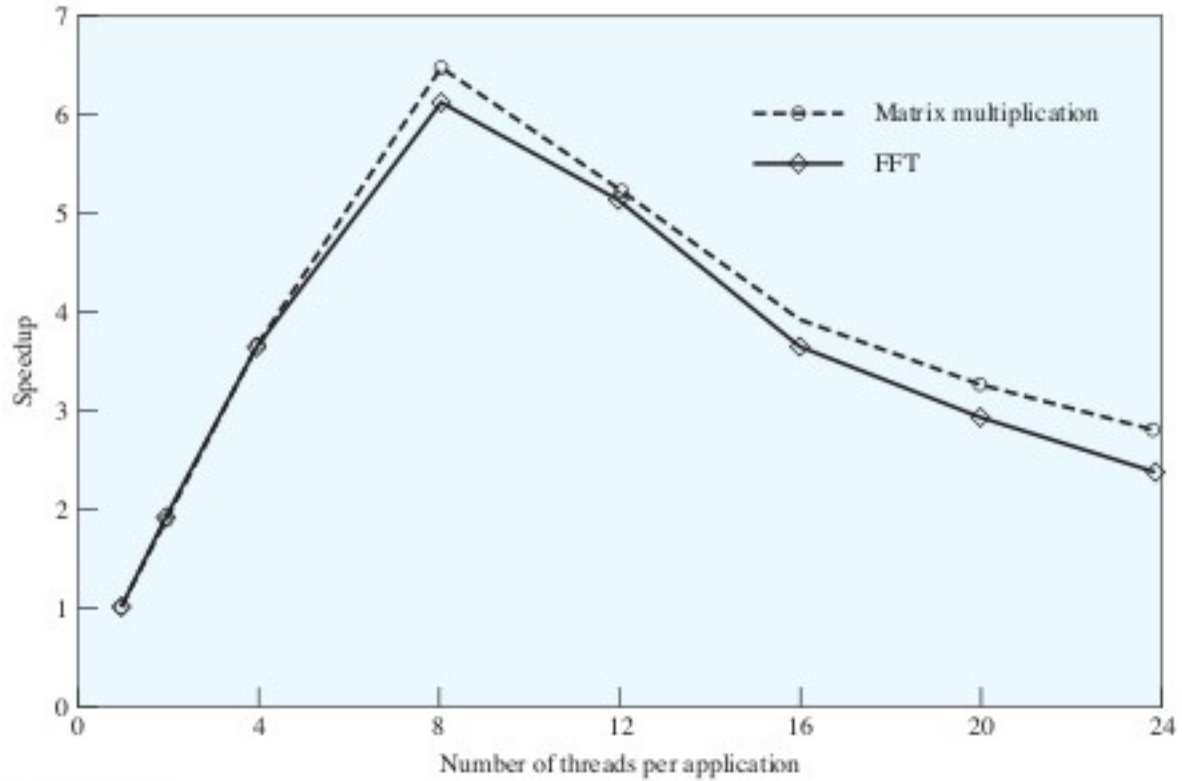


Figure 10.4 Application Speedup as a Function of Number of Threads [TUCK89]

Ordonnancement

Systèmes temps réel: qualités recherchées

- Déterminisme
- Réactivité: capacité de terminer un traitement en un temps limité après un stimulus
- Ordonnancement contrôlable par l'utilisateur
- Fiabilité
- Graceful degradation (fail soft)

Caractéristiques classiques des OS temps réels

- Commutation de processus rapide
- Petite taille
- Réaction rapide aux interruptions externes
- Multitâche avec outils de synchronisation (mutex, sémaphores
- Ordonnancement préemptif avec priorités contrôlables, etc.

Ordonnancement

Ordonnancement temps réel:

- Ordonnancement statique : conçu avant de lancer le système
- Ordonnancement à priorités statiques
- Ordonnancement dynamique à réservation
- Ordonnancement dynamique sans réservation (best effort)

Ordonnancement statiques: surtout pour tâches périodiques

Priorité statiques: p. ex. rate « monotonic scheduling »

Ordonnanceur à réservation: on décide dynamiquement mais dès que possible si on accepte la tâche et quand l'ordonnancer

Best effort: comme dans un système classique, on décide quelle tâche lancer lorsque le processeur devient libre: pas de garantie de respecter les échéances, mais souvent utilisé

Ordonnancement

Ordonnancement temps réel: données utilisées pour décider de l'ordonnancement

- Ready time: stimulus qui active la tâche
- Completion deadline (earliest deadline first)
- Processing time
- Starting deadline (completion - processing time): si on démarre plus tard, on n'aura pas fini à temps (least laxity first)
- Ressources nécessaires
- Importance relative de la tâche (concept différent de l'urgence)
- Présence d'éléments optionels dans le traitement

Ces informations ne sont pas facile à connaître !

Des algorithmes de choix dynamiques compliqués coûtent du temps!

- Un choix facile: priorités fixes mais risque de « starvation »

Ordonnancement

Ordonnancement temps réel: l'inversion de priorité

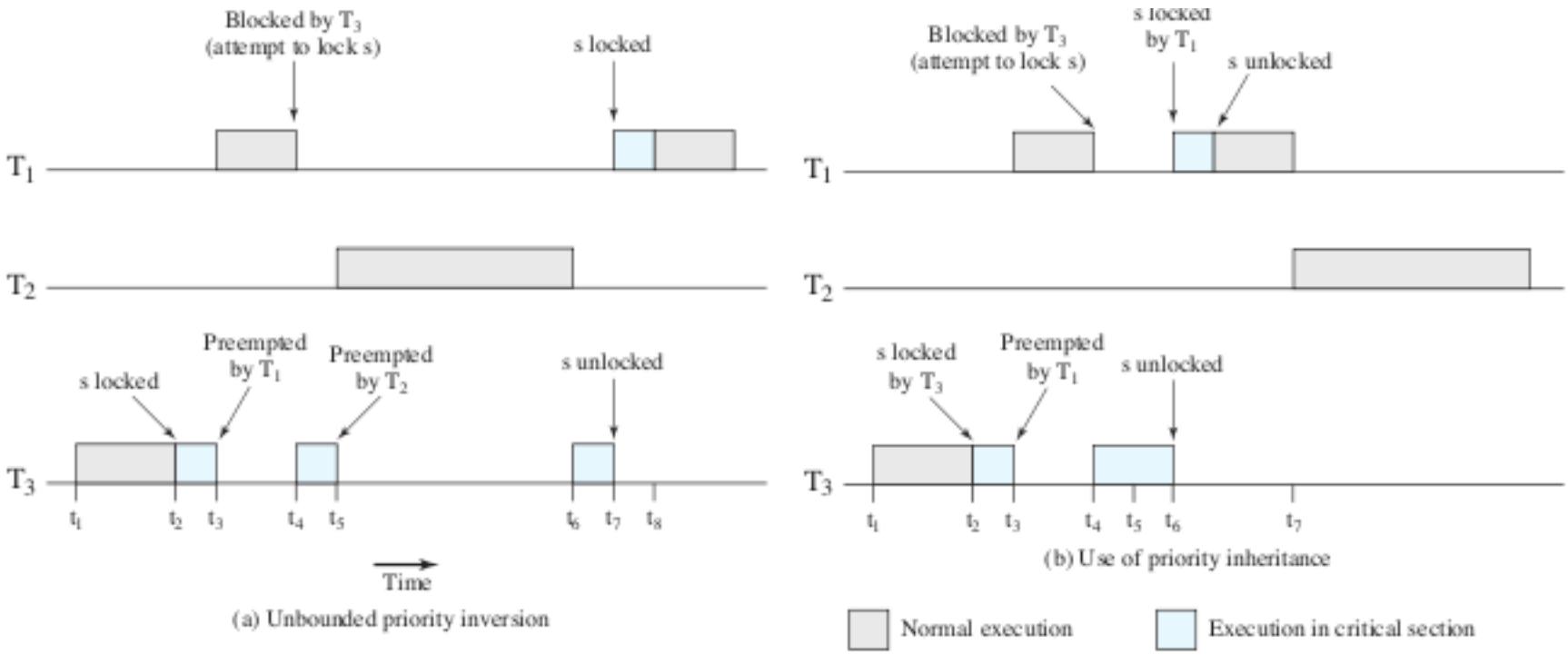


Figure 10.10 **Priority Inversion**