Ordonnancement des processus

Aloÿs Dufour

ATER, LIPN équipe LoCal
Université Paris-Nord XIII

3 février 2026

Ordonnancement des processus

Dans les systèmes multi-tâches, deux processus ne peuvent pas s'exécuter en même temps sur un même cœur $\Rightarrow$ politique de gestion d'attribution des ressources aux processus qui en font la demande :
ordonnancement, partie du noyau qui s'occupe de cela : ordonnanceur (scheduler).

Ordonnancement des processus

Dans les systèmes multi-tâches, deux processus ne peuvent pas s'exécuter en même temps sur un même cœur $\Rightarrow$ politique de gestion d'attribution des ressources aux processus qui en font la demande :
ordonnancement, partie du noyau qui s'occupe de cela : ordonnanceur (scheduler).

Travail de l'ordnnanceur :

- Choisir à quel processus il doit donner la main
- et combien de temps il lui donne, pour le ordonnanceur préemptifs...

Ordonnancement des processus

Dans les systèmes multi-tâches, deux processus ne peuvent pas s'exécuter en même temps sur un même cœur $\Rightarrow$ politique de gestion d'attribution des ressources aux processus qui en font la demande :
ordonnancement, partie du noyau qui s'occupe de cela : ordonnanceur (scheduler).

Travail de l'ordnnanceur :

- Choisir à quel processus il doit donner la main
- et combien de temps il lui donne, pour le ordonnanceur préemptifs...

préemption : capacité à interrompre une tâche en cours en faveur d'une autre.

Rappel sur les changements d'état

État des processus (ou threads) considérés :

- exécution : sur un des cœurs
- prêt : le processus attend que l'ordonnanceur lui donne la main
- en attente : le processus ne peut/doit pas être exécuté pour le moment (en attente d'une I/O, d'un mutex, syscall, ...)
- terminé.

Rappel sur les changements d'état

État des processus (ou threads) considérés :

- exécution : sur un des cœurs
- prêt : le processus attend que l'ordonnanceur lui donne la main
- en attente : le processus ne peut/doit pas être exécuté pour le moment (en attente d'une I/O, d'un mutex, syscall, ...)
- terminé.

Changements :

- exécution → attente : appel système sur une ressource bloquante
- attente → prêt : la ressource se libère
- prêt → exécution : l'ordonnanceur donne la main au processus (dispatch)
- exécution → prêt :
  - le processus décide lui-même de rendre la main (`shed_yield()`)
  - le temps accordé au processus est dépassé (interruption).

Types d'Ordonnancement

À long terme : Sélection des programmes à admettre dans l'OS.

À moyen terme : Sélection des programmes à débarquer/embarquer en mémoire.

À court terme : Gestion des processus prêts.

Objectifs d'un ordonnanceur — Critères complémentaire/contradictoire

- Maximisation de l'utilisation CPU
- Réactivité : minimiser le temps de réponse
- Minimiser le temps total d'une tâche courte
- Minimires le temps total pour un long travail
- Respecter les priorités ("niceness")
- Assurer l'équité entre les processus prêts
- Prédictibilité
- Éviter de passer trop de temps à ordonnancer et faire des context-switch

Impossible de tout satisfaire

Les différents temps

- Temps total/“réel” : durée entre l’instant de création et l’instant de terminaison
- Temps user : temps passé (en exécution) en mode utilisateur
- Temps système : temps passé en mode système (syscalls)
- Temps d’attente : temps passé dans l’état “prêt”

Avec `time cmd` dans le shell.

Algorithmique de l'ordonnancement

Problème difficile, beaucoup de stratégies, beaucoup d'objectifs, les algorithmes simples ont souvent des problèmes.

Algorithmique de l'ordonnancement

Problème difficile, beaucoup de stratégies, beaucoup d'objectifs, les algorithmes simples ont souvent des problèmes.

Stratégies naïves typiques non préemptives :

- First-Come, First-Served (FCFS, FIFO) : coopératif, algo minimal (file d'attente), peu de contex-switch, temps de réponse potentiellement infini, pas de gestion de priorité.
- Shortest-Job First (SJF) : résout le problème de la réactivité. Problème : connaître (ou pire, prédire) le temps d'exécution d'un processus.

Ordonnancement préemptif

- À l'aide de l'horloge interne, on peut interrompre périodiquement.
- Périodiquement, le système prend la main et décide s'il interrompt le processus en cours ou non.
- Commutation de contexte (context-switch) : Sauvegarde le contexte du processus interrompu et charge le contexte du processus choisi.
- Stratégies d'ordonnancement préemptif :
  - Version préemptive de SJF : Shortest Remaining Time First (SRTF)
  - Round-Robin (RR) avec ou sans priorité

Ordonnancement préemptif : Round-Robin

Supposons n'avoir qu'un seul CPU, avec un seul cœur.

- La liste des processus prêts est gérée en FIFO
- le processeur est alloué au processus en tête de file pendant un quantum de temps
- si le processus se bloque (I/O) ou se termine avant la fin du quantum, le processeur est immédiatement alloué à un autre processus (tête de file)
- Si le processus courant ne se termine pas au bout de son quantum, il est suspendue (interruption), et il passe en queue de file.

Ordonnancement préemptif : Round-Robin

Supposons n'avoir qu'un seul CPU, avec un seul cœur.

- La liste des processus prêts est gérée en FIFO
- le processeur est alloué au processus en tête de file pendant un quantum de temps
- si le processus se bloque (I/O) ou se termine avant la fin du quantum, le processeur est immédiatement alloué à un autre processus (tête de file)
- Si le processus courant ne se termine pas au bout de son quantum, il est suspendue (interruption), et il passe en queue de file.

Propriétés :

- Avantages : pas de famine, bon temps de réponse moyen
- Choix : du quantum de temps (entre 1 et 100 ms en général) :
  - trop petit : beaucoup de commutations de contexte, et devient équivalent à SRTF/SJF,
  - trop élevé : temps de réponse élevé, et devient équivalent à FCFS/FIFO.
- Pas de gestion de priorité.
- Si un processus fait souvent appel à des I/O bloquantes, il est laissé.

Ordonnancement préemptif avec priorité

Pour les priorités :

- Une FIFO par priorité (si beaucoup de priorité, peut être lourd)
- Ou alors utilisation de files de priorités (plutôt que FIFO/LIFO) :
  - ABR,
  - arbres rouge-noir,
  - tas binomiaux,
  - tas de Fibonacci.

Ordonnancement préemptif multiprocesseur (SMP)

Avec plusieurs processeurs/cœurs : on associe 1 processus/thread par cœur pour éviter les histoires de cache et de mémoire.
Si un cœur est {sur,sous}-utilisé, l'ordonnanceur peut décider de déplacer la charge de travail (load-balancing).

Ordonnancement préemptif multiprocesseur (SMP)

Avec plusieurs processeurs/cœurs : on associe 1 processus/thread par cœur pour éviter les histoires de cache et de mémoire.
Si un cœur est {sur,sous}-utilisé, l'ordonnanceur peut décider de déplacer la charge de travail (load-balancing).

Architecture UMA (uniform memory access) : mémoire centrale partagée par plusieurs processeurs/cœurs. Si trop de cœurs, goulot d'étranglement pour l'accès mémoire

Architecture NUMA (non-uniform ") : chaque processeur (ou groupe de cœur) dispose de sa mémoire (et forme un nœud).
Chaque nœud peut accéder à la mémoire de toute la machine, si c'est la mémoire d'un autre nœud, passage par un bus (rapide, certes, mais lent à l'échelle du cœur).

Exemple d'ordonnanceurs réels : Linux 2.4 (2001-2011)

$O(n)$ scheduler :

- le temps est divisé en "epoch", à chacun, chaque thread a droit à un certain nombre de temps CPU basé sur sa priorité,
- si un thread n'a pas épuisé tout son temps CPU au cours d'une epoch, il rajoute la moitié du temps restant à son temps à l'epoch suivante.
- Problème : temps linéaire en le nombre de processus entre chaque epoch.

Exemple d'ordonnanceurs réels : Linux 2.6.0 à 2.6.22 (2003-2007)

$O(1)$ scheduler :

- 140 niveaux de priorité (0-99 pour le système et temps réel, 100-139 pour les utilisateurs),
- 2 queues par niveau de priorité : actif/inactif,
- pénalité si temps écoulé, récompense si I/O bloquante,
- chaque processus a un quantum de temps fixé, après lequel il est préempté et passé dans le tableau d'inactifs, les tableaux étant accessibles uniquement par pointeurs, le changement se fait aussi vite que l'on peut échanger deux pointeurs.

Exemple d'ordonnanceurs réels : Linux 2.6.0 à 2.6.22 (2003-2007)

$O(1)$ scheduler :

- 140 niveaux de priorité (0-99 pour le système et temps réel, 100-139 pour les utilisateurs),
- 2 queues par niveau de priorité : actif/inactif,
- pénalité si temps écoulé, récompense si I/O bloquante,
- chaque processus a un quantum de temps fixé, après lequel il est préempté et passé dans le tableau d'inactifs, les tableaux étant accessibles uniquement par pointeurs, le changement se fait aussi vite que l'on peut échanger deux pointeurs.

Anecdote : la complexité algorithmique compte, en passant du tri à bulle au tri fusion, une version micro-VM de FreeBSD a gagné un facteur 100 de temps de boot (maintenant à 25 ms !).

Exemple d'ordonnanceurs réels : Linux 2.6.23 à 6.5 (2007-2023)

Completely Fair Scheduler (CFS) :

- une file de priorité : arbre rouge-noir, indexé par le temps utilisé par le processus,
- temps accordé : le temps que le processus a attendu $\times$ le ratio du temps qu'il aurait utilisé sur un processeur "idéal",
- l'ordonnanceur donne la main au processus qui a le moins utilisé son temps (plus petit dans la liste),
- quand le processus rend la main, l'ordonnanceur le réinsère dans l'arbre rouge-noir, avec son nouveau temps,
- les processus avec beaucoup de temps d'attente sont naturellement récompensés.

"Fair-scheduling" (équité) : temps d'utilisation CPU équitablement réparti entre le système, les utilisateurs, les groupes (généralisation de RR à chaque niveau d'abstraction).

Exemple d'ordonnanceurs réels : depuis Linux 6.6 (2023-)

Earliest Eligible Virtual Deadline First (EEVDF) :

- publié en 1995 (Stoica, Abdel-Wahab), introduit pour éliminé les patchs concernant la latence nice (priorités).
- essaie d'être équitable, sinon calcul la différence entre la solution équitable et ce qui s'est passé (le "lag"),
- un processus est éligible si son lag ≥ 0, les processus avec une valeur négative de lag sont inéligible, mais à un moment dans le futur, la différence sera rattrapée (par le temps qui passe).
- La date limite virtuelle ("virtual deadline") est le temps le plus proche à partir duquel le processus avoir reçu le temps CPU qui lui est dû.

POSIX pour les priorités

`man 7 sched`
`nice cmd` : lancer un programme avec une priorité (niceness) alternative, a un pendant syscall dans la `libc`.
`getpriority(), setpriority()`

POSIX pour les priorités

`man 7 sched`
`nice cmd` : lancer un programme avec une priorité (niceness) alternative, a un pendant syscall dans la `libc`.
`getpriority(), setpriority()`
Possibilité de changer de politique d'ordonnancement :

- `SCHED_OTHER` : politique standard,
- `SCHED_FIFO`
- `SCHED_RR`

POSIX pour les priorités

`man 7 sched`
`nice cmd` : lancer un programme avec une priorité (niceness) alternative, a un pendant syscall dans la `libc`.
`getpriority(), setpriority()`
Possibilité de changer de politique d'ordonnancement :

- `SCHED_OTHER` : politique standard,
- `SCHED_FIFO`
- `SCHED_RR`

Pour les grosses machines, on peut vouloir contrôler/répartir sur quel nœud les processus/threads tournent : `sched_setaffinity` (spécifique à Linux).

Inversion de priorité

- A de forte priorité
- B de priorité moyenne
- C de faible priorité

Inversion de priorité

- A de forte priorité
- B de priorité moyenne
- C de faible priorité

Scénario :

- C bloque une ressource R
- A demande (et bloque) sur la ressource R
- B arrive
- B prend 100% CPU, car il a la priorité sur C
- C ne relâche pas R
- A ne peut pas être exécuté
- B bloque A par l'intermédiaire de C.

Problème réel : Mars Pathfinder

Inversion de priorité

- A de forte priorité
- B de priorité moyenne
- C de faible priorité

Scénario :

- C bloque une ressource R
- A demande (et bloque) sur la ressource R
- B arrive
- B prend 100% CPU, car il a la priorité sur C
- C ne relâche pas R
- A ne peut pas être exécuté
- B bloque A par l'intermédiaire de C.

Problème réel : Mars Pathfinder

Solutions :

- ne pas trop prioriser les threads de haute priorité,
- donner de temps CPU à des tâches de basse priorité de temps à autres,
- priorité plafond : priorité sur la gestion de ressources (mutex)
- héritage de priorité.

Exo classique

On considère des tâches A, B, C, D et E, dont on suppose connue la date de création et la durée d'exécution.

tâche	date de création	durée d'exécution
A	0	8
B	2^-	2
C	5	6
D	6^-	3
E	7	1

L'exposant $-$ dans la date de création signifie que la tâche est créée juste avant la date indiquée, pour éviter les ambiguïtés. On suppose qu'il n'y a qu'un processeur et que le temps de commutation de contexte est négligeable par rapport au reste.

Pour chaque politique d'ordonnancement ci-dessous, dessiner le diagramme de Gantt en faisant apparaître la file d'attente ou de priorité correspondante. Calculer les dates de fin, temps de traitement, d'attente et de réponse pour chaque tâche puis leurs moyennes. Enfin, dire si la politique d'ordonnancement présente un risque de famine et si oui, donner un exemple de déroulement pour le prouver.

- FCFS (FIFO),
- SJF,
- SRTF,
- Round Robin avec un quantum de temps égal à 2 unités de temps.

Correction FCFS

FIFO, non préemptif, aucun risque de famine (sous l'hypothèse que chaque tâche a une durée finie) car le temps d'attente d'une tâche est égal à la somme des durées de toutes les tâches la précédent dans la file.

tâche	date de création	durée d'exécution	date de fin	tps tr.	tps att.	tps rep.
A	0	8	8	8	0	0
B	2^-	2	10	8	6	6
C	5	6	16	11	5	5
D	6^-	3	19	13	10	10
E	7	1	20	13	12	12
moy.		4		10,6	6,6	6,6

Correction SJF

Non préemptif. Risque de famine. Par exemple si une tâche A est créée au temps 1 et dure 2 mais que pour tout $i \in \mathbb{N}$, une tâche B_i est créée au temps i^- et dure 1, alors A ne sera jamais élue.

tâche	date de création	durée d'exécution	date de fin	tps tr.	tps att.	tps rep.
A	0	8	8	8	0	0
B	2^-	2	11	9	7	7
C	5	6	20	15	9	9
D	6^-	3	14	8	5	5
E	7	1	8	1	0	0
moy.		4		$8,2$	$4,2$	$4,2$

Correction SRTF

Préemptif, même risque de famine que SJF.

tâche	date de création	durée d'exécution	date de fin	tps tr.	tps att.	tps rep.
A	0	8	14	14	6	0
B	2^-	2	4	2	0	0
C	5	6	20	15	9	9
D	6^-	3	10	4	1	0
E	7	1	8	1	0	0
moy.		4		$7,2$	$3,2$	$1,8$

Correction RR

Préemptif, pas de risque de famine. Une tâche en attente dans la file va attendre au plus q fois le nombre de tâche la précédent avant d'être de nouveau élue, où q est le quantum associé au tourniquet.

tâche	date de création	durée d'exécution	date de fin	tps tr.	tps att.	tps rep.
A	0	8	18	18	10	0
B	2^-	2	4	2	0	0
C	5	6	20	15	9	1
D	6^-	3	16	10	7	2
E	7	1	13	6	5	5
moy.		4		$10,2$	$6,2$	$1,6$