Ordonancement avancé

Démo #7, IFT2245

Recap Ordonnancement

- Quel buts?
- Quel algos à-t’on vus?

TP2

- Tâche : Écrire un ordonnanceur
- Multi-coeur
- Minimiser temps total, temps d’attente, temps de réponse

Ordonnancement aléatoire

- Un système d’ordonnancement simple - on pige un processus au hasard, et on attribue soit un quantum, ou on exécute jusqu’à échéance
- Si avec quantum, manière la plus facile d’avoir une distribution équitable du temps d’éxecution en moyenne
- On peut pondérer certain processus plus, donc priorité
-

HRRN (Highest response ratio next)

- Cet algorithme d’ordonnancement essaye de mitiger le problème de famine avec Shortest Job First
- On choisit le processus avec la plus haute valeur de (temps d’attente du processus)÷(temps d’éxecution estimé) et on l’exécute
- Ainsi, tout processus est garanti d’éventuellement être exécuté
- Non-préemptif

CFS (Completely fair scheduler)

- Ordonnanceur de Linux depuis presque 20 ans jusqu'à récemment
- Si on a n processus, on voudrait que chaque processus utilise 1/n du temps du processeur
- Si un processus est en attente depuis t secondes, on voudra alors l'éxecuter pour t/n secondes
- On exécute la tâche qui a été exécutée le moins de temps
- On peut faire un time-warp pour certaine tâches pour implémenter la priorité
- Une nouvelle tâche prend le même temps d'exécution pour valeur initial que le plus petit -> ordonné immédiatement

EEVDF (Earliest eligible virtual deadline first)

- Nouvel ordonnanceur sur Linux (depuis 3 mois)
- Même logique que CFS pour déterminer le temps qu'un processus aurait dû avoir : t/n
- On calcule pour chaque processus le lag : différence entre le temps qu'il aurait dû avoir et temps qu'il a eu
- Lag négatif (eu plus de temps) n'est pas ordonnancé
- Deadline virtuelle : time-slice pour ce processus + lag
- On exécute le processus avec la deadline virtuelle la plus proche
- Encore une fois on peut implémenter priorité avec du time-warp

Problèmes additionnels d’ordonnancement

- Processeurs de nos jours on plusieurs coeurs, certains qui sont plus rapides que d’autre (dépendamment de leur fabrication, mais aussi de leur température). Un ordonnanceur intelligent prend cela en compte quand il décide à quel coeur allouer un processus.
- Coeurs fondamentalement inhomogènes - pas mêmes accès mémoires (NUMA), pas même caches -> temps différent pour bouger/reprendre un processus sur un coeur différent, caractéristiques de performance différentes pour certaines cache sur un type de coeur donné
- Certains programmes (UI) très sensibles à la latence - le time-slice maximum est parfois plus que le temps qu’on a pour générer et afficher une image

Problèmes, bis

- Certains algorithmes ont vraiment besoin du burst-time, mais on ne peut pas facilement savoir c'est quoi. Comment le trouver/estimer?
- Certaines applications ont des demandes de latences très serées : par ex. PID pour un robot -> mois de 1ms. Comment marier cela avec des applications normales?
- Ordonnanceurs compliqués peuvent prendre beaucoup de temps à exécuter, comment implémenter pour que ça soit rapide? Performance asymptotique?
- Comment mesurer responsivité subjective?