Programmation Système

Cours 7

Ordonnancement

Brice Goglin

Copyright

Plan

Objectifs

Type d'ordonnancement

- à long terme
  - Quels processus vont être exécutés ?
- à moyen terme
  - Quels processus sont résidents en mémoire ?
- à court terme
  - Quel processus est exécuté par le processus ?
- Entrées-sorties
  - Quelles I/O de quel processus sont traitées par un périphérique disponible ?

Ordonnancement à long terme

- Degré de multiprogrammation avec des batchs
  - Maximiser l'utilisation du matériel
    - Processeurs et périphériques
  - Eviter trop de défauts de page
- Peut utiliser différents critères
  - Priorités, temps d'exécution prévu, besoins en I/O
  - Optimiser l'utilisation des différentes ressources
- Exécuté au lancement des processus
- Doit être contourné pour les processus interactifs

Ordonnancement à moyen terme

- Consiste essentiellement à évincer ou rappeler en mémoire des processus
- Egalement lié au degré du multiprogrammation
- Exécuté assez régulièrement

Ordonnancement à court terme

- Exécuté très regulièrement
- Quand un événement risque de bloquer un processus
  - I/O
- Quand il est bon de favoriser un autre processus
  - Interruption d'horloge
  - Signaux

Fonctionnenement

Critères pour l'utilisateur

- Critères relatifs à la performance
  - Rapidité d'exécution des processus
  - Temps de réponse
  - *Deadlines*
- Autres critères
  - Prédictabilité

Critères pour le système

- Critères relatifs à la performances
  - Quantité de travail effectué
  - Utilisation des processeurs
    - et des périphériques
- Autres critères
  - Equité
  - Respect des priorités
  - Gestion des ressources

Préemption

- Un processus ne rend jamais la main
  - sauf en cas d'I/O
- Le système prend la main de force
  - Léger surcoût
  - Bien meilleur temps de réponse
    - Intéractivité
  - Quand préempter ?

Algorithmes classiques

- FIFO
- *Round-robin*
  - Exécution pendant petites périodes (*Timeslice*)
- *Short Process Next* et *Short Remaining Next*
  - Demande de prévoir la durée d'exécution
- *Feedback*
  - Pénalité pour les processus trop gourmands
    - Ajuste la priorité initiale des processus

Multiprocesseurs

- *Load Sharing*
  - File générale de processus
- *Gang Scheduling*
  - Threads reliés ordonnancés simultanément sur un ensemble de processeurs
- Assignation d'un processeur dédié
  - Processeur inaccessible jusqu'à la fin d'exécution
- Ordonnancement dynamique

Multiprocesseurs (2/2)

- Concurrence réelle entre les files d'exécution sur différents processeurs
  - Pas seulement en cas de préemption
  - Synchronisation critique
- Effets de cache entre threads
- Localisation mémoire sur NUMA

Besoins du temps réel

- Déterminisme
- Réactivité
- Fiabilité
- Contrôle utilisateur

Caractéristiques des ordonnanceurs temps réel

- Changement de contexte rapide
- Taille et fonctionnalités réduites
- Réaction rapide aux interruptions
  - Minimisation des désactivations
- Ordonnancement préemptif basé sur priorités
- Primitives de retardement et pause de tâches
- Alarmes et *timeouts* dédiés

Ordonnancement des entrées-sorties

Accès disque

- Temps d'accès
  - Recherche du cylindre + rotation jusqu'au secteur
  - De l'ordre de 10 ms
- Débit
  - De l'ordre de 50 Mo/s
- Optimiser les accès pour augmenter le débit réel
  - Regrouper les accès proches

Accès disque (2/2)

- FIFO, avec priorité
- *Elevator*
  - Réduire les déplacements de la tête de lecture
- *Deadline*
  - Lectures privilégiées
- *Anticipatory*
  - Petite pause pour attendre des requêtes consécutives
- *Completely Fair Queuing*
  - Accès aux périphériques par *Timeslice*

Accès réseau

- *Quality of Service*
- *Queuing Disciplines*

Ordonnancement des processus dans Linux

Ordonnancement traditionnel dans Unix

- Priorité de base dynamique
  - *Feedback*
- Facteur ajustable par l'utilisateur (*nice*)
- Swapping très prioritaire
- Priorité croissante des applications aux périphériques réels

Le *Scheduler* de Linux

- Ordonnanceur Unix traditionnel
- Bonne intéractivité et équité
- Priorité effective ajustée selon bonus/penalités
  - Crédit d'intéractivité quand la tâche dort
- Beaucoup d'optimisations locales
  - Tâche créée sans `CLONE_VM` préempte père
    - Evite coût du *Copy-on-Write*

Les *Timeslices*

- Temps divisé en cycles d'ordonnancement
  - Chaque processus prêt recoit une *timeslice* par cycle
- *Timeslice* entre 5 et 800 ms (100 par défaut)
  - Priorité de 19 à -20 (0 par défaut)
- Peut-être consommé en plusieurs fois
  - Si préempté pendant
- Partage de la *timeslice* entre père et fils

Le *O(1)-Scheduler*

- Supporte beaucoup de processus et processeurs
  - Optimisation des structures pour performance
- *runqueues* indépendantes sur chaque processeur
- Chaque processus RUNNING placé dans une des *runqueues*
  - Tableau de priorités
    - Actif ou expiré
- Processus non prêts dans files d'attente externes

Le *O(1)-Scheduler* (2/2)

- Permutation des tableaux expiré et actif à la fin de chaque cycle
- Prochaine *timeslice* calculée à l'expiration de la *timeslice* précédente
- Affinités pour processeurs
  - Champ cpus_allowed de task_struct
    - sched_set/getaffinity()
  - Migration uniquement en cas de déséquilibre
    - *Load Balancer*

Préemption

- Si un processus prioritaire arrive
  - Bit NEED_RESCHED de la tâche courante
  - Vérifié régulièrement par le *scheduler*
- Préemption possible lors du retour en espace utilisateur
  - Fin d'appel système
  - Fin d'interruption
    - Interruption d'horloge

Préemption dans le noyau

- Préemption dans le noyau depuis 2.6
- Compteur `preempt_count` dans `task_struct`
- Augmenté pendant les opérations où il la préemption doit être désactivée
  - Tenue d'un *spin lock*
  - Possession d'un mapping atomique (`kmap_atomic`)
  - à la main `preempt_disable/enable()`
- `need_resched` testé quand compteur nul

Qui préempte qui ?

- Les interruptions sont prioritaires
  - Sauf si désactivées
  - Exécution dans un contexte spécial
    - Un contexte dédié par processeur (`irq_ctx hard_irq`)
  - Interruption peuvent être désactivées pendant traitant
- *Bottom Halves* au retour du traitant d'interruption
  - Exécution dans un contexte spécial
    - Un contexte dédié par processeur (`irq_ctx soft_irq`)
  - Peut être préempté par une interruption

Qui préempte qui ? (2/2)

- Les threads noyaux sont préemptés par
  - les interruptions puis les B*ottom Halves*
  - les tâches prioritaires à certains endroits
    - sauf si `preempt_count > 0`
- Les processus utilisateurs sont préemptés dans le noyau comme les threads noyaux
  - et avant leur retour en espace utilisateur
    - Interruptions (horloge), signaux et appels-système

Changement de contexte

- `schedule()` appelé par un processus
  - Effectue lui-même les tests du *scheduler*
- `switch_mm()` permute les espaces d'adressage
- `switch_to()` permute les contextes d'exécution

sched_yield

- Demande explicite de passage de main
  - Déplace la tâche dans le tableau expiré
  - La *timeslice* est perdue !
- Comportement différent entre 2.4 et 2.6
  - La *timeslice* n'était pas perdue en 2.4
  - Incompatibilités entre applications pour 2.4 et 2.6

Temps réel

- Politiques plus prioritaires que SCHED_OTHER
- SCHED_FIFO
  - Garde le processeur jusqu'à le rendre explicitement
- SCHED_RR
  - *Timeslice* prédéfinie
- Pas de garantie sur l'efficacité
  - Satisfaisant dans les cas simples
- Gros impact sur les autres tâches