Programmation Système

Cours 4

Concurrence

Brice Goglin

Copyright

Plan

- Principes
- Interblocage et famine
- Support matériel
- Sémaphore et moniteurs
- Autres stratégies
- Détails de Linux

Principes de la concurrence

Principe de l'exclusion mutuelle

- Partage de données entre file d'exécution
- Condition de course (*Race Condition*)
  - Résultat dépend de l'ordre
- Utilisation de sections critiques protégées
  - Prise d'un verrou
- Problème dépendant de la coopération des acteurs

Coopération

- Coopération par partage
  - Pas de connaissance explicite des autres
  - Modifications doivent conserver l'intégrité
    - sections critiques
  - Accès en lecture non problématique
- Coopération par communication
  - Connaissance explicite des autres
  - Synchronisation via les communications

Compétition

- Accès concurrents à une même ressource
- Aucune connaissance des autres
- Support nécessaire dans le système d'exploitation
  - Attribution de ressources à un acteur unique
  - Les autres acteurs attendent qu'il la relâche
- Nécessité de rendre la ressource dans un état intègre

Interblocage (*Deadlock*)

- Attente circulaire
  - Exclusion mutuelle
  - Un acteur possède une ressource et attend une autre
  - Pas de préemption
- Ordonner les types de ressources pour éviter les cycles
- Prévention par connaissance du futur des acteurs
- Algorithmes de détection

Famine (*Starvation*)

- Exclusion mutuelle
  - Acteurs en attente d'une ressource
- Privation d'un acteur au profit des autres
  - Respecter l'ordre d'arrivée des acteurs en attente

Support matériel

- Désactivation des interruptions
- Instructions spéciales pour sérialiser les accès mémoire
- Instructions atomiques
  - `test_and_set`
  - `exchange`
- Attente active
- Pas de protections contre interblocage et famine

Sémaphores

- Compteur initialisé >= 0
- `SemWait` décrémente le compteur et bloque s'il deviendrait négatif
- `SemSignal` incrémente le compteur et réveille les `SemWait` bloqués
- Exclusion mutuelle si initialisé à 1
- Attente d'événement si initialisé à 0

Moniteurs

- Objet dont des données locales sont accessibles uniquement par ses méthodes
- Entrées dans le moniteur par une méthode
- Un seul acteur à la fois
- `CondWait` pour suspendre l'acteur dans le moniteur en le relâchant, jusqu'à une condition
- `CondSignal` pour reprendre l'exécution d'un acteur dans le moniteur

Problème des lecteurs et écrivains

- Plusieurs lecteurs simultanément
- Un seul écrivain
- Eviter les famines pour les écrivains
- Eviter les ping-pong entre caches de différents processeurs

Passage de messages

- Synchronisation naturelle car réception après envoi
- Adressage du destinataire et/ou de l'émetteur
  - `Broadcast`, `Reduce`
- Ordonnancement FIFO ou avec priorités
- Exclusion mutuelle par l'envoi d'un message à l'unique acteur autorisé à entrer en section critique

Mécanismes de concurrence dans Unix

- Communication entre processus
  - Tubes
  - Messages
  - Mémoire partagée
- Déclenchement selon actions des autres processus
  - Sémaphores
  - Signaux

Détails de Linux

Concurrence et synchronisation dans le noyau Linux

- Communication et synchronisation entre processus utilisateur
  - IPC pour les communications
  - Mutex et sémaphores pour la synchronisation des pthreads
- Mémoire explicitement partagée dans le noyau
- Beaucoup de stratégies pour la synchronisation dans le noyau

Synchronisation dans le noyau Linux

- Performance (notamment si contention)
- Support SMP (avec passage à l'échelle)
- Accès plutôt en lecture ou plutôt en écriture
- Minimisation des contraintes sur les différentes parties du système
  - ne pas trop désactiver les interruptions
  - ne pas trop faire attendre les tâches déferrées

Opérations atomiques

- `atomic_t`
  - `atomic_add`, `atomic_inc_and_test`
- `cmpxchg`
- Implémentation en assembleur
  - Utilisation directe des instructions atomiques quand elles sont disponibles
  - Instructions CISC suffisent à peu près
  - Emulation via plusieurs instructions sur les RISC

Spin Locks

- Attente active (*spin*)
- `spinlock_t`
  - `spin_lock_init`, `spin_lock`, `spin_unlock`
- `rw_lock_t` pour autoriser plusieurs lecteurs
- *Big Reader Lock* (`br_lock_t`) pour éviter pingpong entre les caches
  - partage des parties nécessaires à l'accès en lecture

Spin Locks (2/2)

- Ne pas dormir en tenant en lock
  - Désactivation de la préemption
  - Pas de fonction bloquante

Sémaphores

- Sommeil jusqu'à disponibilité d'une ressource
- `struct semaphore`
  - `sema_init`, `up`, `down`
- `struct rw_semaphore` pour autoriser plusieurs lecteurs

Conditions

- Sommeil dans une queue par `wait_event`
  - Variantes interruptibles par des signaux, avec un timeout, ...
  - `sleep_on` risqué car ne vérifie pas la condition avant de dormir
- Réveil d'une queue par `wake_up`
  - Variantes pour réveiller plusieurs tâches

Big Kernel Lock

- Verrou global (spinlock)
- De moins en moins utilisé
  - restreint fortement la concurrence
- `lock_kernel`, `unlock_kernel`

Réactivité et préemption

- Ne pas bloquer les autres en gardant un verrou
  - Ne pas perdre la main en tenant un verrou
- Les spinlocks désactivent la préemption
- Désactivation des interruptions
  - local_irq_save/restore
  - spin_lock_irqsave
  - disable_irq