Système d'exploitation – TD2 (en salle de TD)
Sommeil et réveil des processus

1 Définitions

Supposons que les processus sont complètement définis par leur structure `proc` qui contient les champs suivants :

```
struct proc {
  struct proc * p_next; /* maillon de liste */
  int p_status; /* statut du processus */
  ...
}
```

Le système a, à tout moment, accès à la variable globale `struct proc * current` qui pointe vers le processus actif sur le processeur.
Un processus en train de s'exécuter n'appartient pas à la `runq`. C'est l'ordonnanceur qui, lorsqu'il choisit un processus à exécuter, l'enlève de la `runq`.
Le champ `p_status` est à la valeur `STATUS_BLOCKED` ou `STATUS_READY`.
Tous les processus `STATUS_READY` sont dans une liste chaînée globale `struct proc * runq`, reliés par les maillons `p_next` des structures `proc`.
Tous les processus `STATUS_BLOCKED` sont dans une liste chaînée globale `struct proc * sleepq` (également reliés par les maillons `p_next` des structures `proc`).
On ignorera, au moins pour commencer, l'ordre des processus sur la `runq` et la `sleepq`.
Une procédure `switch()` demande à l'ordonnanceur de sauvegarder le contexte du processus et d'exécuter à la place un autre processus parmi les `STATUS_READY`. Si le processus qui a perdu la main était `STATUS_READY`, il est remis dans la `runq` par l'ordonnanceur. Lorsqu'il reprendra la main, il exécutera le code juste après l'appel à `switch()`.

2 Implémentation de base

2.a) Écrivez les fonctions `wait()` et `wakeup()` qui permettent de s'endormir et de réveiller les processus.

2.b) `wakeup()` peut intervenir dans une routine d'interruption. Donnez un exemple. Quels problèmes cela pose-t-il ? Comment les résoudre si on possède maintenant des fonctions `irq_disable()` et `irq_enable()` qui permettent de masquer et réactiver des interruptions ?

2.c) Supposons qu'un processus veuille s'endormir, et qu'avant qu'il rende la main une interruption réveille tous les processus en attente. On considère que les interruptions pendant le masquage sont stockées et délivrées lors de leur réactivation. Que va-t-il se passer ? Est-ce grave ? Quelles solutions proposez-vous ?

2.d) Si notre machine était multiprocesseur ou multicœur, que faudrait-il changer dans votre code ?

3 Réception de paquets réseau

On zoome maintenant sur le cas concret d'une réception réseau. Les processus s'endorment en attendant un paquet (par exemple dans l'appel-système `recv()`) et sont réveillés par une interruption de la carte réseau qui a reçu un paquet.

On dispose d'une variable globale `received_count` décrivant le nombre de paquets reçus par la carte réseau et pas encore consommés par un processus. Un processus appellera la fonction `consume()` pour consommer un paquet après avoir vérifié que `received_count` est strictement positif. Chaque paquet ne peut être consommé qu'une fois.

3.a) Précisez votre code de `wakeup()` et `wait()` pour ne s'endormir que si aucun paquet réseau n'est disponible, et vérifier qu'il y en a bien un de disponible au réveil.

3.b) On modifie maintenant `received_count` pour contenir le nombre d'octets reçus au lieu du nombre de paquets. La fonction `consume()` va elle aussi prendre un nombre d'octets en paramètre. Modifiez `wakeup()` pour incrémenter `received_count` d'un nouveau paramètre `length`. Puis modifiez `wait()` pour consommer autant d'octets qu'indiqués dans un nouveau paramètre `length`.

3.c) Si plusieurs processus peuvent attendre des octets en même temps, et qu'un seul peut s'exécuter à la fois, quelles sont les faiblesses de cette implémentation ? Comment y remédier ?

3.d) Dans quel cas un processus voudrait-il attendre un paquet réseau sans le consommer immédiatement ? Comment en tenir compte ?

3.e) On va maintenant distinguer les différentes connexions dans lesquelles arrivent les paquets. Modifiez les fonctions `wakeup()` et `wait()` pour tenir compte de la connexion cible qui sera donnée dans un nouvel argument `struct conn * conn`.

3.f) Comment améliorer votre code pour éviter de parcourir toute la `sleepq` si aucun processus n'attend sur la socket cible ?

4 Extensions

4.a) Formalisez la notion d'événement qu'un processus peut attendre et ce que la structure associée doit contenir.

4.b) Comment gérer le cas où un processus souhaite dormir jusqu'à ce qu'un événement se produise parmi un certain nombre ? Dans quel cas est-ce utile ? Comment allouer toutes les structures nécessaires ? Et si on veut attendre que tous les événements se produisent ?

4.c) Comment améliorer votre code pour prendre en compte un champ priorité dans la structure `proc` ? On souhaite que les processus prioritaires soient réveillés en premier et exécutés en premier, il faudra donc trier les processus dans les files `runq` et `sleepq`. Quel problème peut se présenter avec ce modèle quand il y a beaucoup de processus avec des priorités différentes et comment y remédier ?

4.d) Que doit-on faire si un signal arrive pendant qu'un processus est endormi ? Toutes les raisons d'endormissement sont-elles équivalentes ? Proposez une solution.

5 Exemples plus concrets

5.a) Comment adapter le code si on considère une lecture dans un tube au lieu d'une connexion réseau ? Et comment bloquer un processus s'il écrit trop dans le tube ?

5.b) Implémentez un sémaphore (`sem_init`, `sem_post`, `sem_wait`). On veillera à ce qu'il puisse être utilisé dans un traitant d'interruption. Donnez un exemple concret où cela pourrait arriver.

5.c) Implémentez des variables de condition (`pthread_cond_wait`, `signal` et `broadcast`). En quoi est-ce similaire à un sémaphore ?