Systèmes d'exploitation centralisés

1re année Informatique et Réseaux

17 juin 2019

Documents autorisés. Les exercices sont indépendants. Une réponse non justifiée est sans intérêt.

I Coroutines (4 points)

On souhaite réaliser un distributeur (dispatcher) d'événement, tel que, selon l'événement, le contrôle soit transféré à un contexte (une coroutine) spécifique. La coroutine spécifique est nommé le traitant de l'événement. Le gestionnaire est déclenché magiquement (en fait, c'est une interruption matérielle qui le déclenche) et son algorithme est :

```
/* les événements sont identifiés de 0 à NBEVENT - 1 */
#define NBEVENT 20
/* Le tableau des coroutines à invoquer pour un événement donné. */
/* Ce tableau a été initialisé au démarrage avec des coroutines utilisateurs. */
coroutine_t event[NBEVENT];

/* le code du dispatcher */
void dispatcher()
   idev = obtenir_le_numéro_d_événement_à_distribuer(); // magique
   dest = event[idev];  // la coroutine correspondant à cet événement
   transférer le contrôle à la coroutine dest
   nettoyer et réarmer l'interruption matérielle // magique
```

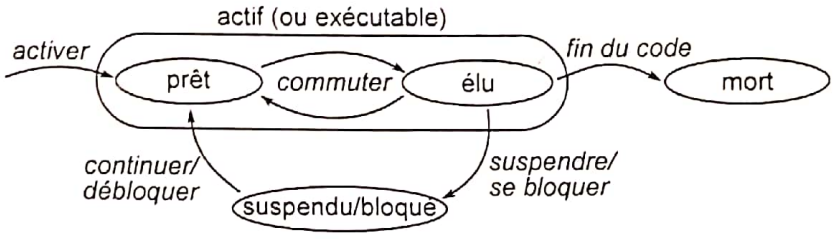
Pour fonctionner correctement, la coroutine qui traite un événement doit finalement retourner le contrôle au dispatcher. Pour cela une fonction `return_to_dispatcher(/* sans paramètre */)` doit être définie et un traitant d'événement s'engage à l'appeler.

Questions

1. Donner le code précis du dispatcher (en ignorant les parties magiques)
2. Donner le code de `return_to_dispatcher`
3. Comment pourrait-on détecter qu'un traitant a oublié d'invoquer `return_to_dispatcher` ? Expliquer alors la mise en œuvre pour corriger automatiquement un tel oubli.

II Ordonnancement de processus (7 pt)

On considère le problème de l'ordonnancement des processus dans un noyau de système d'exploitation, c'est-à-dire les stratégies d'allocation du temps processeur aux processus prêts à s'exécuter. Il peut exister plusieurs processeurs réels (architecture multi-processeurs). On étudie des solutions basées sur l'affectation d'une *priorité* aux processus, et sur le partage du temps processeur par *quantum*. Un processus libère le processeur s'il se bloque, ou s'il est préempté et ce uniquement à la fin de son quantum : la création d'un nouveau processus de priorité supérieure à celle des processus élus n'interrompt par l'un d'eux pendant son quantum.



On suppose que de nouveaux processus prêts sont sporadiquement créés et leur profil d'exécution évolue en passant par des phases interactives (entrées-sorties bloquantes fréquentes) et des phases calculatoires.

II.1 Stratégie à quantum de durée unique

Dans cette stratégie, la durée d'un quantum est unique et fixe, quelle que soit la priorité du processus. Chaque processus est créé avec une priorité prise dans $[1..PRIOMIN]$, 1 étant le *maximum* et $PRIOMIN$ la priorité minimale. Une file d'attente des processus prêts est associée à chaque priorité.

Priorité fixe Quand un processeur se libère, un processus prêt de priorité pr reçoit un quantum de temps si et seulement si aucun processus de priorité supérieure n'existe et s'il est en tête de la file d'attente pr (= il est en tête de la file la plus prioritaire non vide).

En fin du quantum d'un processus élu, l'ordonnanceur met ce processus en fin de la file d'attente correspondant à sa priorité (stratégie du tourniquet) et il choisit alors le processus prêt le plus prioritaire comme indiqué précédemment. Si un processus actif se bloque (= se suspend) avant la fin de son quantum, il sera réinséré en fin de la file d'attente correspondant à sa priorité quand il redeviendra prêt (= qu'il sera continué).

1. Expliquer pourquoi cette stratégie présente un risque de famine en donnant un scénario précis conduisant à une situation où certains processus prêts ne deviennent jamais élus.

Priorité variable, première variante Sur fin de quantum, un processus élu de priorité pr est inséré en fin de la file de priorité supérieure $pr - 1$ si $pr > 1$. On augmente donc si possible sa priorité au fur et à mesure de son exécution. Comme précédemment, un processus qui se bloque sera réinséré à sa priorité initiale quand il redeviendra prêt.

2. Quel profil de processus est favorisé par cette stratégie et pourquoi ?
3. Cette stratégie comporte-t-elle encore un risque de famine ? Si oui, pour quel(s) profil(s) de processus ?

Priorité variable, seconde variante Sur fin de quantum, un processus élu conserve la même priorité. Par contre un processus élu de priorité pr qui se bloque sera inséré, quand il redeviendra prêt, dans la file de priorité supérieure $pr - 1$ (à condition que $pr > 1$).

4. Quel profil de processus est favorisé par cette stratégie et pourquoi ?
5. Cette stratégie comporte-t-elle encore un risque de famine ?

II.2 Stratégie à quantum de durées distinctes

On conserve la gestion des priorités mais, à chacune des priorités, est associé un quantum de durée différente : un processus de priorité pr reçoit un quantum $pr \times d$, où d est le quantum minimum. Comme précédemment, l'ordonnanceur choisit le processus en tête de la file la plus prioritaire non vide. Sa stratégie consiste à modifier la priorité courante d'un processus ainsi :

- réinsertion d'un processus élu de priorité pr en fin de la file de priorité inférieure $pr + 1$ si et seulement si il consomme son quantum et n'est pas encore de priorité minimale $PRIOMIN$;
- réinsertion, lorsqu'il redeviendra prêt, d'un processus de priorité pr en fin de la file de priorité supérieure $pr - 1$ si et seulement si il se bloque avant la fin de son quantum et n'est pas déjà de priorité maximale 1.

6. Expliquer comment cette stratégie essaie d'adapter de manière optimale la durée du quantum selon l'évolution du profil d'exécution du processus.
7. Cette stratégie comporte-t-elle un risque de famine ?

III Mémoire virtuelle (4 points)

1. Quel est le rôle de l'unité de gestion mémoire (la MMU) dans un système de mémoire virtuelle ? En particulier, préciser sous quelles conditions l'unité de gestion mémoire engendre une interruption « défaut de page ».
2. Indiquer si les énoncés suivants sont vrais ou faux, en justifiant la réponse :
   - Un programme peut s'exécuter même s'il est de taille plus grande que la mémoire physique ;
   - Une page virtuelle peut être représentée à la fois par une page en mémoire physique et par une page sur disque (zone de swap).
3. Le noyau étudié en TP implante une mémoire virtuelle paginée, avec va-et-vient (swapping) mais sans translation d'adresse (dans notre noyau, l'adresse virtuelle est égale à l'adresse en mémoire centrale). En quoi la translation d'adresse serait un complément intéressant au va-et-vient pour réduire le nombre de pages écrites ?
4. Dans le noyau étudié en TP, quel serait l'intérêt de partager une même page de swap par plusieurs processus ?

IV Systèmes de fichiers (5 points)

Un fichier épars (*sparse file*) est un fichier pour lequel seuls les blocs contenant effectivement des données sont alloués : les blocs vides (ne contenant que des 0) ne sont pas alloués. Pour autant, l'utilisateur peut parfaitement lire l'intégralité du fichier, le système traduisant l'absence de bloc en un bloc vide.

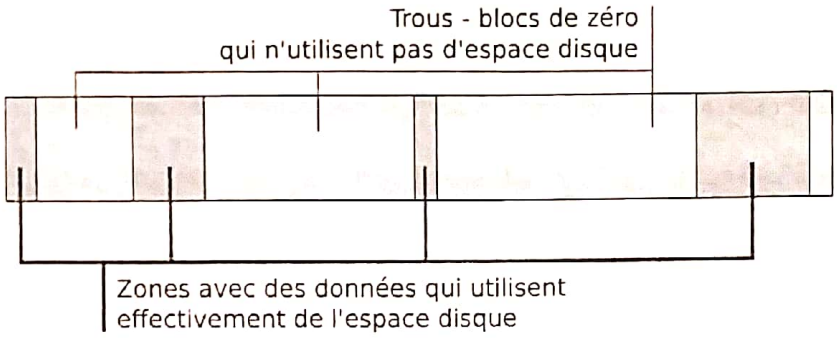


FIGURE 1 – Fichier épars [1]

1. Quel est l'intérêt de cette technique ?
2. La taille (longueur) d'un fichier et son occupation sur disque ne sont plus nécessairement identiques. Quel est l'inconvénient ?
3. Soit le programme suivant :

```
#include <unistd.h>
#include <fcntl.h>
int main()
{
    int fd = open("toto", O_WRONLY|O_TRUNC|O_CREAT, 0644);
    write(fd, "coucou", 6);
    lseek(fd, 1024000, SEEK_SET);
    write(fd, "toto", 4);
    close(fd);
    return 0;
}
```

   (a) Quel est la longueur (en octets) du fichier ?

   (b) En supposant des blocs de 1Ko, combien de blocs de données ce fichier utilise-t-il s'il n'y a pas de support pour les fichiers épars (ignorer les blocs d'indirection) ?

   (c) En supposant des blocs de 1Ko, combien de blocs de données ce fichier utilise-t-il s'il y a un support pour les fichiers épars (ignorer les blocs d'indirection) ?

4. Expliquer quels modifications on peut faire à la couche inode (rangement des données de type UFS) pour supporter les fichiers épars.

1 Source : wikimedia GFDL