université de BORDEAUX

Collège Sciences et technologies

Année	2015-2016	**Type**	Devoir Surveillé
Master	Informatique		
Code UE	J1INAW11	**Épreuve**	Systèmes d'Exploitation
Date	12/11/2015	**Documents**	Non autorisés
Début	10h15	**Durée**	1h30

1 Question de cours (échauffement)

Question 1 Dans les systèmes d'exploitations « interactifs », quelle catégorie de processus est privilégiée par l'algorithme d'ordonnancement ? Comment le système d'exploitation reconnaît-il de tels processus ?

Question 2 Rappelez précisément le rôle du circuit `MMU` dans un processeur. À quel endroit de la machine se trouvent les tables des pages des processus ? Expliquez par quelle technique les systèmes d'exploitation peuvent diminuer la taille totale occupée par ces tables (aidez-vous d'un petit schéma).

2 Gestion de processus

L'algorithme d'ordonnancement interactif implanté dans les noyaux Linux 2.4.x utilise un système de quanta de temps (i.e. crédits) que les processus sont autorisés à utiliser pendant une durée que l'on appelle une « *époque* ». Le nombre de quanta attribués à chaque processus lors du démarrage d'une nouvelle époque dépend de sa priorité et du nombre de quanta qu'il n'a pas consommés lors des époques précédentes.

Question 1 Décrivez l'algorithme principal, c'est-à-dire le code exécuté dans le noyau à chaque fois que le temporisateur (horloge) expire. Un processus peut-il encore avoir des crédits à la fin d'une époque ? Pourquoi ?

Question 2 En prenant un exemple simple constitué d'un processus P_1 ayant droit à 5 quanta de temps au début d'une époque, et de P_2 ayant droit à 2 quanta, tracer un petit chronogramme (sur la durée d'une époque) illustrant à quels moments surviendront les changements de contexte.

3 Synchronisation

On considère un système d'impression composé de plusieurs processus et d'une imprimante qui partagent une mémoire commune. La *file d'impression* a une taille maximale de `MAX_SPOOL` octets. Les processus soumettent des *tâches d'impression* de taille variable. Une tâche d'impression est décrite par la structure suivante :

```
struct job_t {
  void *data;      /* adresse des données à imprimer */
  size_t size;     /* taille des données */
  unsigned status; /* vaut INIT ou COMPLETED */
};
```

Pour soumettre un travail d'impression, un processus doit initialiser une variable de type `struct job_t` et appeller une fonction `submit` décrite ci-après. Cette fonction bloque le processus s'il ne reste pas assez de place dans la file, puis recopie les données de la tâche d'impression dans la file (au moyen d'une primitive `spool_insert` que vous n'avez pas à écrire).

```
// Variables globales (partagées entre tous les processus) indiquant
// respectiveent la place restante dans la file et le nombre de travaux soumis
unsigned available = MAX_SPOOL;
unsigned nb_jobs = 0;

void submit(struct job_t *job)
{
  while (available < job->size) ; /* rien */

  available -= job->size;
  nb_jobs++;
  job->status = INIT;
  spool_insert(job);

  while (job->status != COMPLETED) ; /* rien */
}
```

De son côté, un processus en tâche de fond (gérant l'imprimante) exécute la fonction `printer_daemon` suivante. La primitive `spool_remove` (que vous n'avez pas à écrire) récupére une tâche de la file d'impression.

```
void printer_daemon()
{
  struct job_t *job;

  while (1) {
    while (nb_jobs == 0) ;/* rien */

    job = spool_remove(); /* récupère une tâche et place les données
                             dans la mémoire interne de l'imprimante */
    nb_jobs--;
    available += job->size;

    SLEEP(DUREE_IMPRESSION); /* on simule la durée d'impression */

    job->status = COMPLETED;
  }
}
```

Question 1 Montrez que le système d'impression tel que programmé ci-dessus risque d'aboutir à des incidents, c'est-à-dire à des situations où la file d'impression est corrompue.

Question 2 Corrigez donc le protocole en introduisant des moniteurs de Hoare (et peut-être d'autres variables ou champs dans la structure `job_t`) partagés et en modifiant les programmes. Profitez de l'occasion pour éviter l'utilisation des boucles d'attente active. N'oubliez pas de préciser les valeurs initiales.

Voici pour rappel les primitives que vous pouvez utiliser :

```
typedef ... mutex_t ;
typedef ... cond_t ;
void mutex_lock(mutex_t *m);
void mutex_unlock(mutex_t *m);
void cond_wait(cond_t *c, mutex_t *m);
void cond_signal(cond_t *c);
void cond_bcast(cond_t *c);
```

4 Appels système

Certains appels systèmes sont destinés à créér (ou à allouer) de nouveaux objets dans le noyau (*e.g.* sémaphores, descripteurs de fichiers) et ils retournent un "identifiant" d'objet (souvent de type opaque) qui sera manipulé par l'application. Pour accélérer les accès ultérieurs à un objet, il serait possible de retourner véritablement un pointeur sur l'objet "déguisé" en type opaque à l'application. Expliquez pourquoi cela n'est pas concevable. Quelle est la bonne manière de faire ?