université de BORDEAUX

Collège Sciences et technologies

Année	2019-2020	**Type**	Devoir Surveillé
Master	Informatique		
Code UE	4TIN705U	**Épreuve**	Systèmes d'Exploitation
Date	6/11/2019	**Documents**	Non autorisés
Début	10h30	**Durée**	1h30

La plupart des questions peuvent être traitées même si vous n'avez pas répondu aux précédentes. *À la fin du sujet, vous trouverez un memento vous rappelant la syntaxe de quelques fonctions utiles.*

1 Questions de cours (échauffement)

Question 1 À l'aide d'un joli chronogramme, décrivez la séquence d'événements qui se produisent à partir du moment où un processus effectue un appel système bloquant (e.g. `GetChar`) jusqu'au moment où l'appel système retourne en espace utilisateur…

Question 2 Qu'appelle-t-on le *noyau* dans un système d'exploitation ? À quels moments s'exécute-t-il sur le processeur ? Comment réussit-il a être incontournable pour un processus qui souhaite accéder au matériel ?

Question 3 L'algorithme d'ordonnancement implanté dans les noyaux Linux $2.4.x$ utilise un système de crédits que les processus sont autorisés à utiliser pendant une durée que l'on appelle « *époque* ». En prenant un exemple simple constitué de trois processus P_1, P_2 et P_3 dotés au départ respectivement de 1, 2 et 3 crédits, tracer un petit chronogramme (sur la durée d'une époque) illustrant à quels moments surviendront les changements de contexte.

2 Le grand bain

On souhaite simuler le fonctionnement d'une piscine municipale. Le comportement des baigneurs est modélisé au moyen de *threads* : leur nombre est aléatoire, tout comme le moment où chacun d'eux est lancé. Chaque thread exécute la fonction `thread_baigneur` (décrite ci-après) puis disparaît.

```
1  void thread_baigneur ()
2  {
3    if (payer_au_guichet () ==-1)
4      return;      // c'est complet :(
5
6    unsigned bracelet = trouver_casier ();
7
8    sleep (SE_CHANGER_PUIS_SE_BAIGNER);
9
10   liberer_casier (bracelet);
11 }
```

Lorsqu'un baigneur arrive, il passe d'abord par un guichet pour payer le droit d'entrée (ligne 4). Une fois à l'intérieur de l'enceinte, le baigneur doit d'abord trouver un casier libre pour en récupérer la clé (ligne 7), puis il peut y déposer ses affaires et se baigner (ligne 9). En fin de baignade, il se change et libère son casier (ligne 11).

2.1 Gestion des casiers

Intéressons-nous à la gestion des casiers dans un premier temps. On suppose que le paiement au guichet ne garantit pas à un baigneur de trouver immédiatement un casier libre pour se changer [1]. Le code ci-après constitue une première tentative d'implémentation de la fonction `trouver_casier`. Remarquez que si le nombre de baigneurs qui tentent d'obtenir un casier est supérieur à `MAX_CASIERS`, certains sont obligés de patienter jusqu'à ce qu'un casier se libère…

1. On peut considérer cela comme une forme de surbooking.

```
bool casiers [MAX_CASIERS] = { FAUX, ..., FAUX }; // FAUX = LIBRE

// retourne le numéro du casier obtenu (sorte de "bracelet" que le baigneur portera au poignet)
unsigned trouver_casier ()
{
  while (1)
    for (int i = 0; i < MAX_CASIERS; i++)
      if (casiers [i] == FAUX) {
        casiers [i] = VRAI;
        return i;
      }
}

void liberer_casier (int num)
{
  casiers [num] = FAUX;
}
```

Question 1 Montrez que l'implémentation de la fonction `trouver_casier` ci-dessus risque d'aboutir à des erreurs.

Question 2 Corrigez le code en introduisant des moniteurs/conditions (et sans doute d'autres variables) partagés. Profitez de l'occasion pour éviter l'utilisation des boucles d'attente active. N'oubliez pas de préciser les valeurs initiales. *Conseil :* Une variable comptant le nombre de casiers libres vous facilitera les choses.

2.2 Paiement au guichet

Voici une implémentation préliminaire de la fonction `payer_au_guichet` :

```
int reste = CAPACITE_MAX; // nombre de places disponibles à la vente

int payer_au_guichet ()
{
  if (reste == 0)
    return -1; // c'est complet

  reste--;

  sleep (DELAI_PAIEMENT);
  return 0; // tout va bien
}
```

Question 3 Donnez une version de `payer_au_guichet` synchronisée avec des moniteurs/conditions, de manière à ce que les baigneurs obtiennent leur ticket tour à tour. Notez qu'une file d'attente implicite va se créer, en raison du temps de paiement de chaque baigneur. Néanmoins, dès que la capacité maximale de la piscine est atteinte, on souhaite que la fonction retourne *immédiatement* -1.

Question 4 On souhaite que si le nombre de personne dans la file (i.e. bloqués à l'intérieur de la fonction) atteint une constante `MAX_FILE`, cela devienne dissuasif pour les personnes qui arrivent après : elles font demi-tour immédiatement (i.e. la fonction `payer_au_guichet` doit renvoyer -1 *sans délai*). Donnez le nouveau code de la fonction.

Question 5 De manière plus réaliste, il y a maintenant plusieurs guichets auprès desquels les clients achètent leur ticket, mais toujours une seule file d'attente. La constante `NB_GUICHETS` indique le nombre de guichets. Donnez une nouvelle version de la fonction `payer_au_guichet` dans laquelle il peut y avoir jusqu'à `NB_GUICHETS` personnes qui achètent simultanément leur place (i.e. ces personnes exécutent `sleep(DELAI_PAIEMENT)` en parallèle).

Memento

```
typedef ... mutex_t;
void mutex_lock(mutex_t *m);
void mutex_unlock(mutex_t *m);
```

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