université de BORDEAUX

Collège Sciences et technologies

Année	2024-2025	Type	Examen Session 1
Master	Informatique		
Code UE	4TIN705EX	Épreuve	Systèmes d'Exploitation
Date	13/12/2024	Documents	Non autorisés
Début	14h30	Durée	1h30

De nombreuses 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 appels système utiles.*

1 Questions de cours (échauffement)

Question 1 Les systèmes d'exploitation modernes utilisent la pagination en s'appuyant sur des tables de pages à plusieurs niveaux. Faites un joli dessin d'une table des pages à 3 niveaux, en détaillant le mécanisme de conversion d'adresses. Mentionnez le principal avantage d'un passage à une table à 4 niveaux (plutôt que 3). Y a-t-il une contrepartie ?

Question 2 Décrivez brièvement le principe de la segmentation mémoire. De quel support matériel a-t-on besoin pour l'implémenter ? Donnez quelques avantages et inconvénients de cette technique.

2 À l'aéroport

On souhaite simuler le fonctionnement d'un aéroport en modélisant le flux des passagers à leur sortie d'avion au moyen de *threads*. Leur nombre est aléatoire, tout comme le moment où chacun d'eux se décide à sortir de l'aéronef.

Chaque thread enchaine l'exécution de `montrer_passeport` puis de `recuperer_bagage` puis se termine.

Question 1 Montrez que l'implémentation suivante de `montrer_passeport` risque d'aboutir à des erreurs.

```
bool guichet_dispo[NB_GUICHETS] = { TRUE, ..., TRUE }; // guichets de contrôle des passeports

void montrer_passeport ()
{
  int i;
  while (1)
    for (i = 0; i < NB_GUICHETS; i++)
      if (guichet_dispo [i] == TRUE) {
        guichet_dispo [i] = FALSE;
        break; // on suppose que ça provoque la sortie du while(1)
      }
  sleep (DELAI_EXAMEN_PASSEPORT);
  guichets_dispo [i] = TRUE; // le contrôle est terminé
}
```

Question 2 Corrigez le code en introduisant des moniteurs/conditions (et peut-être d'autres variables) partagés. Profitez de l'occasion pour éviter l'utilisation de boucles d'attente active tout en autorisant plusieurs contrôles simultanés.

Question 3 Une fois leur passeport vérifié, les passagers peuvent enfin aller flâner devant le tapis roulant qui leur apportera leur bagage tôt ou tard. Le tapis roulant est simplement implémenté par un tableau d'entiers de taille `MAX_BAGAGES` : chaque case contient soit -1 si elle est vide, soit un numéro de bagage. En coulisse, un thread « bagagiste » s'affaire à déposer des bagages sur le tapis dès que c'est possible, tandis que les passagers scrutent le tapis avec anxiété...

```
int tapis[MAX_BAGAGES] =  { -1, ..., -1 };
int charge = 0;

void recuperer_bagage (int mon_ticket)
{
  while (1) {
    while (charge == 0) ; // attendre

    for (int p = 0; p < MAX_BAGAGES; p++)
      // On admire le tapis qui tourne
      if (tapis [p] == mon_ticket) {
        // gagné... y'a plus qu'à attendre le bus !
        tapis [p] = -1;
        return;
      }
  }
}
```

```
void bagagiste ()
{
  while (1) {
    int bagage = recuperer_depuis_chariot();
    if (bagage == -1) return; // j'ai fini ma journée

    while (charge == MAX_BAGAGES) ; // attendre

    for (int p = 0; p < MAX_BAGAGES; p++)
      if (tapis [p] == -1) {
        // on a trouvé un emplacement, alors on y
        // jette le bagage de toutes ses forces
        tapis [p] = bagage;
        break;
      }
  }
}
```

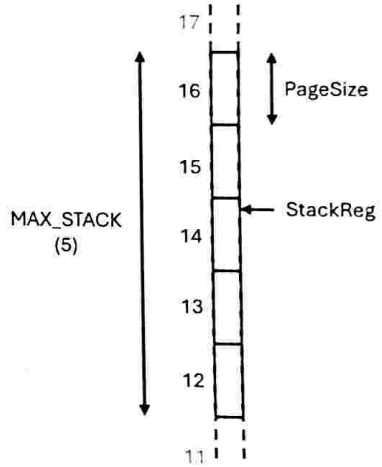
Assurez la synchronisation correcte des threads en utilisant des moniteurs/conditions.

3 Gestion mémoire

On souhaite mettre en place l'allocation automatique de pile dans Nachos, c'est-à-dire une allocation des pages à la demande dans ces zones. On suppose qu'en dehors des piles des threads, toutes les pages valides du processus sont allouées au lancement (pas d'allocation paresseuse généralisée).
La classe `Thread` possède un champ `stack_base` qui contient le numéro de la page virtuelle de la base de sa pile (e.g. 12 dans l'illustration ci-contre). Une constante `MAX_STACK` indique la taille de la pile en nombre de pages (5 dans l'exemple).
Lors de la création d'un thread, aucune page n'est allouée dans sa pile :

```
for (i = currentThread->stack_base;
       i < currentThread->stack_base + MAX_STACK;
       i++) {
  pageTable[i].physicalPage = 0;
  pageTable[i].valid = FALSE;
  pageTable[i].readOnly = FALSE;
  ...
```

L'idée est d'attendre le premier accès à une page de pile pour l'allouer « juste à temps. » Dans l'exemple ci-contre, on remarque que le thread courant a déjà accédé aux deux premières pages de sa pile puisqu'elles sont allouées.



(a) Illustration d'une pile de 5 pages réservée pour un thread. Dans cet exemple, seule les deux premières pages (grisées) sont allouées.

Question 1 Sur l'illustration précédente, on remarque que le sommet de pile (contenu dans le registre `StackReg`) pointe au sein d'une page pas encore allouée. C'est possible ça ? Expliquez.

Question 2 Écrivez une fonction `int inside_stack (int virt_address)` qui renvoie 1 si l'adresse virtuelle[1] passée en paramètre est comprise dans les limites de la pile du thread courant , 0 sinon.

Question 3 Lorsqu'un thread tente d'accèder à une page de sa pile qui n'a pas encore été allouée, l'exception `PageFaultException` est levée, ce qui provoque le basculement dans le noyau dans le traitant `ExceptionHandler`.

1. Indépendamment du fait que la page correspondante soit allouée ou pas)

```
void ExceptionHandler (ExceptionType which)
{
  if (which == PageFaultException) {
    int address = machine->ReadRegister (BadVAddrReg);
      ... // À compléter
    }
```

Complétez le code de manière à traiter correctement l'exception et allouer, si opportun, la page de pile manquante. Dans le cas contraire, `interrupt->PowerDown()` fera l'affaire.

Question 4 Le code écrit jusqu'à présent alloue les pages de pile du thread courant « à la demande », sans distinguer les bons et les mauvais accès à la pile. Par exemple, le code suivant, qui accède à la pile sous le sommet de pile, n'est normalement pas correct :

```
void f()
{
  int i; int *ptr = &i - 1024;
  *ptr = 12;
}
```

On voudrait qu'un tel code, lorsqu'il accède à une page pas encore allouée, soit sanctionné. Modifiez la fonction `inside_stack` en conséquence.

Question 5 Expliquez dans quelles circonstances ont pourrait arriver à une situation où les pages déjà allouées au sein d'une pile ne sont pas toutes contiguës (par exemple il existe une page non-allouée entre deux pages allouées).

Si on souhaite éviter ça, il faut, lorsqu'on alloue une page, garantir que toutes les pages « plus haut » sont également allouées. Modifiez `ExceptionHandler` dans ce sens.

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);
```