UNIVERSITÉ BORDEAUX 1 Sciences Technologies

Année 2008-2009
Étape : Master Sciences Technologies (semestre 1)
UE : INF463 (Systèmes d'Exploitation)
Modalités : Épreuve sans document

Date : 19 décembre 2008
Heure : 14h00
Durée : 1h30

Lisez l'intégralité du sujet *avant de commencer à répondre calmement aux questions.*

1 Gestion Mémoire et pagination

On considère un système de pagination à trois niveaux dans lequel les adresses (virtuelles et physiques) sont codées sur 32 bits :

- les 8 premiers bits d'une adresse virtuelle forment un index dans la table (*racine*) de premier niveau ;
- les 4 bits suivants servent à indexer les tables de second niveau ;
- les 8 bits suivants servent à indexer les tables de troisième niveau ;
- enfin, les 12 bits restants forment le déplacement.

On suppose que chacune des entrées de ces tables occupe 32 bits.

Question 1 Faites un dessin d'une telle table des pages hiérarchique en illustrant comment les différents champs d'une adresse sont utilisés par la MMU pour convertir les adresses virtuelles en adresses physiques.

Question 2 Détaillez le contenu typique (i.e. les champs de la structure C correspondante) des entrées des tables des pages de troisième niveau. Précisez notamment le rôle des différents bits de contrôle. Pour chacun d'entre eux, expliquez quelle entité les positionne (système ? matériel ? les deux ?), quelle entité les consulte et à quel moment.

Question 3 Quelle est la taille de la table des pages de premier niveau ? De quelle taille sont celles de second et de troisième niveau ? Déduisez-en la taille totale occupée par la table hiérarchique (en supposant que toutes les tables de niveau 2 et 3 sont allouées).

Question 4 À partir de quelle taille un "trou" (i.e. suite de pages virtuelles non-allouées) permet-il d'éviter l'allocation d'une table de niveau 3 ?

Même question pour une table de niveau 2.

Question 5 À partir de quelle taille un "trou" (en supposant que le trou débute par chance à une adresse multiple de 2^{24} octets) permet-il à cette table hiérarchique de consommer moins de mémoire qu'une solution s'appuyant sur une table à un seul niveau ?

2 Synchronisation

On se place dans le cadre du simulateur Nachos. On s'intéresse à l'objet `frameProvider` (instance de la classe `FrameProvider`) qui gère l'allocation des pages physiques dans le noyau. En voici une version opérationnelle dans le cas simple où seul un processus réclame/restitue des pages à la fois :

```
class FrameProvider
{
  public:

    int GetEmptyFrame()
    {
      int frame = bitmap->Find();

      if (frame != -1)
        bzero(mainMemory + ... ); // clear page

      return frame;
    }

    void ReleaseFrame(int frame)
    {
      bitmap->Clear(frame);
    }

    FrameProvider () // Initialization
    {
      bitmap = new BitMap(NumPhysPages);
    }

  private:
    BitMap *bitmap;
};
```

Question 1 En sachant que la classe `BitMap` ne comporte aucune synchronisation, expliquez précisément ce qu'il risque de se produire si deux processus exécutent `frameProvider->GetEmptyFrame()` simultanément.

Question 2 En ajoutant un ou plusieurs sémaphores dans `FrameProvider`, donnez une version synchronisée des méthodes `GetEmptyFrame` et `ReleaseFrame`. Il n'est pas demandé de donner la nouvelle version du constructeur mais juste d'indiquer à quelle valeur vous initialisez le(s) sémaphore(s).

Question 3 Lors de la création d'un processus, ses pages sont typiquement allouées de la manière suivante (lors de l'initialisation de sa table des pages) :

```
for (i = 0; i < numPages; i++) {
  pageTable[i].physicalPage = frameProvider->GetEmptyFrame();
  if(pageTable[i].physicalPage == -1) {
    ... // Oops!
  }
  ...
```

Lorsque `GetEmptyFrame` renvoie -1, il n'y a plus aucune page physique disponible. Que faut-il faire avant de retourner un échec pour la création du processus ? Donnez le corps du bloc conditionnel.

Question 4 Plutôt que de renvoyer -1, la fonction `GetEmptyFrame` pourrait bloquer le processus appelant jusqu'à ce qu'une page se libère de nouveau dans le système. Ajoutez la synchronisation nécessaire dans la classe `FrameProvider` pour implémenter ce comportement. N'oubliez pas de préciser la valeur initiale des différents sémaphores !

Question 5 Expliquez pourquoi ce fonctionnement est en réalité une très mauvaise idée qui pourrait conduire à une situation d'interblocage. Donnez un exemple précis.

Question 6 On décide d'adopter un mode de fonctionnement différent, dans lequel les demandes de pages sont toujours précédées d'une "réservation" préalable (au moyen d'une fonction `reserveFrames` de la classe `FrameProvider`). On suppose que tous les processus respectent ce protocole :

```
if (frameProvider->reserveFrames(numPages) == -1)
  // Oops!
for (i = 0; i < numPages; i++) {
  pageTable[i].physicalPage = frameProvider->GetEmptyFrame();
  ...
```

La réservation peut évidemment échouer (i.e. la fonction retourne -1 s'il ne reste pas assez de pages) mais, si elle réussit (code de retour 0), alors `FrameProvider` doit garantir que les "`numPages`" appels ultérieurs à `GetEmptyFrame` réussiront.

Donnez la nouvelle version de la classe `FrameProvider`.