UGA - IUT2 - Département informatique
BUT 2 - R3.05 Programmation système - 2022/2023

R3.05 - Contrôle sur table - Durée : 2h

Note : Cette version est légèrement différente de celle effectivement soumise aux étudiants de l'année 2022/2023. Des imprécisions ont été corrigées.

Aucun document ni calculatrice autorisé
Les exercices sont indépendants - Le barème est estimatif.

1. Questions de cours (3,5 pts)

1.1. Questions à choix multiples

Pour chacune des question suivantes, **indiquer les propositions exactes**. Sélectionner une mauvaise réponse sera pénalisé, sans pour autant que la note associée à cette question devienne négative.

1. L'appel système **`execvp()`** :
    a. duplique l'espace mémoire du processus appelant.
    b. lance systématiquement l'exécution d'un shell.
    c. permet l'exécution d'une commande existante dans l'arborescence des répertoires.
    d. remplace l'espace mémoire du processus appelant.
2. Un journal :
    a. permet de réaliser à nouveau des actions sur le Système de Fichiers si elles ont été interrompues par un arrêt intempestif de la machine.
    b. est installé principalement en mémoire principale (RAM).
    c. est installé principalement en mémoire secondaire (HDD ou SSD).
    d. est un autre nom pour le SWAP.
    e. a toujours été présent dans tous les systèmes de fichiers de type ext (Linux), dès ext2fs.
3. Sélectionner parmi les éléments suivants, le ou les périphériques de type caractères :
    a. CPU.
    b. souris.
    c. clavier.
    d. stockage SSD.
    e. disque dur HDD.

1.2. Questions ouvertes

Proposer des réponses brèves aux questions suivantes.

1. Dans quel cas un processus passe-t-il dans l'état *bloqué*?
2. Comment supprimer un processus *zombie* ?
3. A quoi correspondent les 3 descripteurs de fichiers définis systématiquement pour tout processus?
4. Pour chacun des contextes suivants, donner un exemple d'interruption vu en cours. Bien préciser pour chacune de vos réponses, à quel contexte a ou b elle se réfère.
    a. exécution d'un processus;
    b. gestion de la mémoire principale;

2. Processus (2,5 pts)

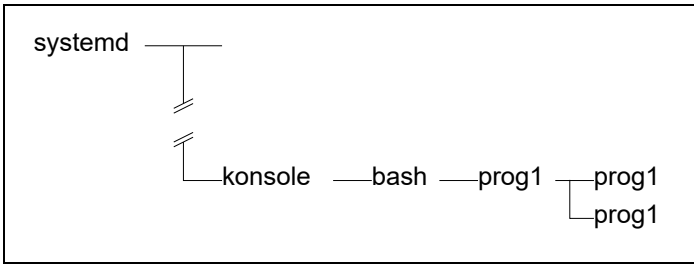
Questions :

1. Dessiner l'arbre-film des processus correspondant à l'exécution du programme `programme-1.c` (en supposant que tous les **`fork()`** s'exécutent avec succès). A l'extrémité de chaque feuille de cet arbre, écrire ce qui est affiché par la fonction **`printf()`**. Les `pids` des processus seront contenus dans [100;200]. L'ordre entre les `pids` des différents processus est sans importance.
2. Compléter cet arbre-film en dessinant une ligne horizontale correspondant à un instant compatible avec l'arbre-photo 1 donné ci-dessous (où l'on voit que l'exécutable produit par compilation de `programme-1.c` s'appelle `prog1`).

programme-1.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  pid_t a = 0;
  pid_t b = 0;
  pid_t c = 0;

  a = fork();
  if ( a != 0 ) {
    b = fork();
    if ( b == 0 ) {
      c = fork();
    }
  }

  printf("pid = %d, a = %d, b = %d, c = %d\n",
         getpid(), (int)a, (int)b, (int)c);

  return EXIT_SUCCESS;
}
```

Arbre-photo 1



3. Ordonnancement (4 pts)

Nous considérons 3 processus avec les caractéristiques suivantes :

PID	durée d'activité demandée	courtoisie NI
101	4	-10
231	3	-10
390	1	+5

La priorité d'un processus ordinaire sous Linux est définie à partir de trois valeurs :

- une valeur par défaut égale à 20;
- une courtoisie sur laquelle l'utilisateur peut agir;
- un bonus dynamique défini par le système en fonction de l'interactivité du processus.

En cours nous avons ignoré le bonus dynamique, et défini la priorité PR d'un processus à partir des deux premières valeurs seulement. Nous en faisons de même dans cet exercice.

Par ailleurs, nous considérons un ordonnancement mêlant RR (tourniquet) et priorités selon le principe suivant :

- les quanta distribués entre les processus sont infiniment petits;
- les processus de priorité PR strictement inférieure à 20 reçoivent des **quanta deux fois plus longs** que les processus de priorité PR supérieure ou égale à 20.

Questions :

1. Quelle est la priorité PR des trois processus considérés? Justifier par une équation.
2. Dessiner le chronogramme de l'exécution *simultanée* de ces trois processus si le processeur dispose d'un seul cœur. Vous prendrez soin de noter les temps de terminaison de chacun des trois processus.
3. Même question si le processeur dispose de quatre cœurs.
4. Nous considérons un utilisateur qui n'a pas les droits *root* et nous supposons que la gestion des priorités par les utilisateurs est identique à celle vue pour Linux. Cet utilisateur peut-il définir n'importe quelle nouvelle valeur de courtoisie pour chacun de ces trois processus avec la commande **`renice`**? Justifier.

4. Mémoire Principale (7 pts)

4.1. Allocations

Nous considérons une mémoire principale de 32 Kio découpée en cases de 4 Kio et gérée avec pagination avec un TLB contenant 2 entrées et des tables de pages contenant 10 entrées. Les adresses seront exprimées en hexadécimal.

Le TLB et la table des pages d'un processus nommé P1 sont dans les états suivants :

TLB

num. de page	num. de case
1	5
7	6

Table des pages de P1

num. de page	num. de case	*present*	*dirty*
0	1	p	-
1	2	-	d
2			
3			
4			
5			
6	5	-	-
7	4	p	d
8			
9			

1. De combien de cases est constituée la mémoire principale? Justifier.
2. On peut dire que le processus P1 n'est pas le processus actif. Pourquoi?
3. Le processus actif est nommé P2. Quelle est la taille (en Kio) de l'espace mémoire logique de P2?
4. P1 devient actif et demande l'accès à son adresse logique `0x7a3c`. Quel est l'adresse physique correspondante?
5. Le processus P1 demande cet accès une deuxième fois, en restant l'unique processus actif. A chaque fois la traduction est nécessaire. Mais combien d'accès à la RAM a nécessité chacune de ces deux traductions? Justifier.
6. Le processus P1 effectue ensuite des accès aux pages 0, 1, 6 et 7. Indiquer pour chaque page s'il y a **nécessairement** défaut de page **majeur**. Justifier.

4.2. Défauts de page

Nous supposons travailler sur un système où un entier de type **`int`** tient sur 4 octets et où une page tient sur 4Kio. Nous considérons les programmes `programme-2.c` et `programme-3.c` suivants :

programme-2.c

```
int main(void)
{
  int A[2048];
  int B[2048];
  register int i;

  for (i=0; i<2048; i++)
    B[i]=i;

  for (i=0; i<2048; i++)
    A[i]=B[i];

  return 0;
}
```

programme-3.c

```
int main(void)
{
  int A[2048];
  int B[2048];
  register int i;

  for (i=0; i<1024; i++)
    B[i]=i;

  for (i=0; i<1024; i++)
    A[i]=B[i];

  for (i=1024; i<2048; i++)
    B[i]=i;

  for (i=1024; i<2048; i++)
    A[i]=B[i];

  return 0;
}
```

Nous ferons l'hypothèse que la traduction en assembleur de l'opération **`z=a+b+c+...`** accède aux différents opérandes dans l'ordre suivant: **`a`**, **`b`**, **`c`**... puis finalement **`z`**. *Attention!* si un opérande est une valeur ou une variable stockée en registre, son accès n'entraîne aucun accès à une page de la mémoire!

Cet exercice illustre en particulier la différence possible entre les nombres de défauts de pages réalisés par deux programmes réalisant une tâche équivalente.

1. Donner pour chacun des deux programmes la chaîne de référence ω (on ignorera la page contenant le code). Justifier en particulier le nombre de pages nécessaires pour stocker chaque tableau.
2. Si seulement deux cases de 4Kio sont disponibles, combien de défauts de pages cause chacun des deux programmes, en supposant que la stratégie de remplacement de page FIFO est appliquée? Justifier par le dessin des pages présentes en mémoire.
   Rappel : si la chaîne de référence est ω=A B C, alors le dessin des pages présentes dans une mémoire de deux cases est :

case a	A		C
case b		B	

3. Même question si seulement trois cases de 4Kio sont disponibles.
4. Combien de cases libres faut-il pour que les deux programmes minimisent le nombre de défauts de page? Justifier.

5. Système de Gestion de Fichiers et E/S (3 pts)

5.1. Lectures d'un fichier

```
#!/bin/bash

/usr/bin/time -f "%U user - %S sys - %e real " cat /tmp/mon-fichier.txt > /dev/null
/usr/bin/time -f "%U user - %S sys - %e real " cat /tmp/mon-fichier.txt > /dev/null
/usr/bin/time -f "%U user - %S sys - %e real " cat /tmp/mon-fichier.txt > /dev/null
```

Nous considérons le script ci-dessus, dont l'exécution renvoie le résultat suivant.

```
0.01 user - 0.56 sys - 2.57 real
0.00 user - 0.14 sys - 0.14 real
0.00 user - 0.10 sys - 0.10 real
```

1. En général, dans quel cas la somme des temps `user + sys` de l'exécution d'un programme est-elle sensiblement inférieure du temps `real` mesurés par la commande `time`?
2. Le résultat de ce script indique que sur les trois exécutions, seule la première a une somme des temps `user + sys` très inférieure au temps `real`. Expliquer cela en termes de SGF.

5.2. I-nœuds

Nous travaillons avec un disque SSD sur lequel a été installé un SGF ext4 avec des blocs de données de 4Kio. La commande `ls -ls --block-size=4K /home/pjharvey/albums/LetEnglandShake/` renvoie

```
3276801 -rw------- 1 peel bbc    3276800 jan  5 08:50 On_Battleship_Hill.txt
      1 -rw------- 1 peel bbc          1 jan  5 08:01 The_Glorious_Land.txt
```

Rappels: un extent contient au plus 32 768 blocs de données de 4 Kio, et un bloc de données de 4 Kio peut contenir jusqu'à 340 descripteurs d'extents.

1. Combien de **i-nœuds** sont utilisés par le fichier de 12.5 Gio `/home/pjharvey/albums/LetEnglandShake/On_Battleship_Hill.txt`? Justifier.
2. Quels sont les nombres minimaux et maximaux d'**extents** utilisés par ce fichier pour contenir ses données (`On_Battleship_Hill.txt`)? Justifier.
3. Combien de **i-nœuds** sont utilisés par le fichier de 4 Kio `/home/pjharvey/albums/LetEnglandShake/The_Glorious_Land.txt`? Justifier.
4. Quels sont les nombres minimaux et maximaux d'**extents** utilisés par ce fichier (`The_Glorious_Land.txt`)? Justifier.