

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

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.

**barème : +0.5pt par bonne réponse, -0.5pt par mauvaise, min=0**

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.

**[ 1 pt ] c et d**

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.

**[ 1 pt ] a et c**

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 pt ] b et c**

#### 1.2. Questions ouvertes

Proposer des réponses brèves aux questions suivantes.

1. Dans quel cas un processus passe-t-il dans l'état *bloqué*?

**[ 1 pt ] attente d'une E/S**

2. Comment supprimer un processus *zombie* ?

**[ 1 pt ] En tuant son père : le processus zombie est alors adopté par systemd qui supprime son descripteur de processus.**

3. A quoi correspondent les 3 descripteurs de fichiers définis systématiquement pour tout processus?

**[ 1 pt ] entrée standard, sortie standard, erreur standard.**

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;

**[ 1 pt ]**

- a. exécution d'un processus : passage en mode noyau (ex: lors d'un appel système);
- b. gestion de la mémoire principale : défaut de page;

## 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.

[ 4 pts : voir barème ci-dessous ]

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).

[ 1 pt : arbre et ligne compatibles avec pstree ]

programme-1.c

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

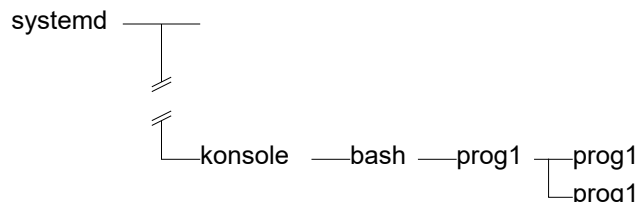
int main(void)
{
    pid_t a = 0;
    pid_t b = 0;
    pid_t c = 0;

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

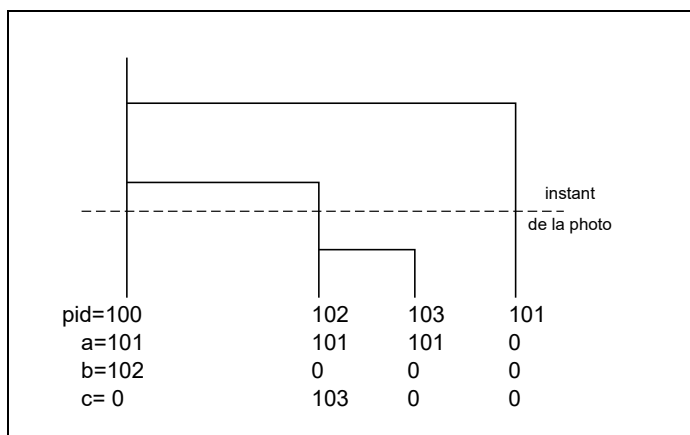
    printf("pid = %d, a = %d, b = %d, c = %d\n",
           getpid(), (int)a, (int)b, (int)c);

    return EXIT_SUCCESS;
}
```

Arbre-photo 1



Arbre-film



- si pas un arbre : 0
- si arbre, mais de mauvaise forme et pids pas cohérents : 1 pt
- si arbre, mais de mauvaise forme mais pids cohérents : 2 pts
- si arbre, de bonne forme mais mauvais pids : 2 pts
- si arbre, de bonne forme et bons pids : 4 pts

## 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.

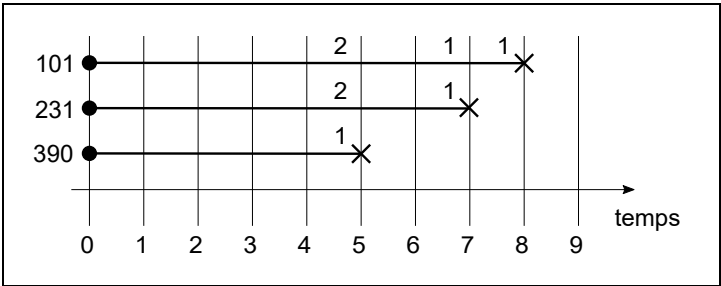
[ 1 pt ]  $PR = 20 + NI$  donc

- 101:  $PR=10$
- 231:  $PR=10$
- 390:  $PR=25$

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 pts = 1pt pour la forme (simultanéité des exécutions) + 1pt pour la compréhension de la répartition + 1pt pour l'exactitude des calculs]

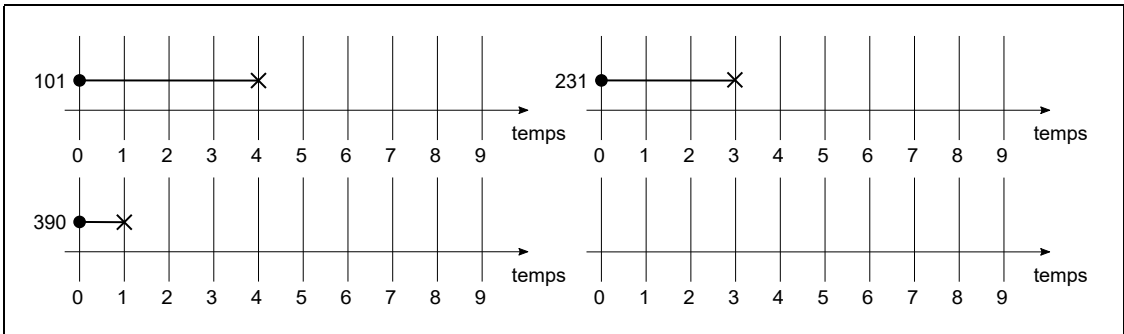
Un seul cœur



3. Même question si le processeur dispose de quatre cœurs.

[ 2 pts = 1pt pour la forme (répartition des exécutions entre les cœurs) + 1pt pour l'exactitude des tracés et des durées]

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.

[ 2 pts = 1pt pour la réponse + 1 pour la justification] Un utilisateur non root ne peut modifier la courtoisie que des processus dont il est propriétaire et pour l'augmenter seulement. De plus il ne peut pas lancer une exécution avec une courtoisie négative. Un utilisateur non root ne peut pas être propriétaire de 101 et 231 et peut augmenter la courtoisie de 390 s'il en est le propriétaire.

## 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.

[1pt] 32 Kio / 4Kio = 8 cases.

2. On peut dire que le processus P1 n'est pas le processus actif. Pourquoi?

[1pt] Sa table des pages n'est pas cohérente avec le TLB.

3. Le processus actif est nommé P2. Quelle est la taille (en Kio) de l'espace mémoire logique de P2?

[1pt] La même que P1 comme tout processus dans ce système: 10 entrées, donc 10x4Kio = 40 Kio

4. P1 devient actif et demande l'accès à son adresse logique 0x7a3c. Quel est l'adresse physique correspondante?

[1 pt] 0x4a3c

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.

[1 pt] 1 accès pour mettre à jour le TLB pour la première traduction, puis aucun pour la deuxième traduction.

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.

[3 pts, 1 par cas avec la justification]

- 0 et 7 : non car la page est présente;
- 1 : oui, la page est absente et dirty; elle doit donc être recherchée depuis le SWAP;
- 6 : non s'il faut simplement rechercher une case vide (et donc s'il ne s'agit pas d'une page du segment de code par exemple).

## 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  $z=a+b+c+\dots$  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  $\omega$  (on ignorera la page contenant le code). Justifier en particulier le nombre de pages nécessaires pour stocker chaque tableau.

**[3 pts : 1 pour nb pages par tableau + 1 pour  $\omega$  de prog-2 + 1 pour  $\omega$  de prog-3]**

**Le tableau A tient sur  $2048 \times 4 / 4096 = 2$  pages, A1 et A2.**

**De même le tableau B tient sur 2 pages B1 et B2.**

**programme-2.c :  $\omega = B1^{*1024} B2^{*1024} (B1 A1)^{*1024} (B2 A2)^{*1024}$**

**programme-3.c :  $\omega = B1^{*1024} (B1 A1)^{*1024} B2^{*1024} (B2 A2)^{*1024}$**

**on accepte aussi**

**programme-2.c :  $\omega = B1 B2 (B1 A1)^{*1024} (B2 A2)^{*1024}$**

**programme-3.c :  $\omega = B1 (B1 A1)^{*1024} B2 (B2 A2)^{*1024}$**

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  $\omega = A B C$ , alors le dessin des pages présentes dans une mémoire de deux cases est :*

case a	A		C
case b		B	

**[2 pts : 1pt par programme avec justification] programme-2.c en fait 6 (B1 B2 - A1 B1 - B2 A2) alors que programme-3.c en fait 4 (B1 A1 - B2 A2).**

3. Même question si seulement trois cases de 4Kio sont disponibles.

**[1 pt] programme-2.c en fait 4 (B1 B2 A1 - A2) tout comme programme-3.c (B1 A1 B2 - A2).**

4. Combien de cases libres faut-il pour que les deux programmes minimisent le nombre de défauts de page? Justifier.

**[1 pt] On ne peut pas avoir moins de 4 ddp, donc 3 cases suffisent pour minimiser les ddp par les deux programmes.**

## 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?

**[1pt] Lorsque le processus fait des E/S : il passe une partie de sa vie dans l'état bloqué.**

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.

**[1pt] Chaque commande effectue la lecture du même fichier. Les data-blocks utilisés sont cherchés en mémoire secondaire à la première exécution. Lors des exécutions suivantes ils sont cherchés en mémoire principale, dans le cache disque. La durée totale d'exécution passe alors de 2.57 sec. à 0.14 sec.**

### 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.

**[1 pt pour l'ensemble des deux questions 1 et 3]**

**Le fichier comme tout fichier, utilise 1 seul i-nœud.**

2. Quels sont les nombres minimaux et maximaux d'**extents** utilisés par ce fichier pour contenir ses données (`On_Battleship_Hill.txt`)? Justifier.

**[2 pts]**

**D'une part 3276800 datablocks = 100 x 32768 = au moins 100 extents. D'autre part 3276801-3276800=1 data bloc d'indirection = au plus 340 extents.**

3. Combien de **i-nœuds** sont utilisés par le fichier de 4 Kio `/home/pjharvey/albums/LetEnglandShake/The_Glorious_Land.txt`? Justifier.

**[1 pt pour l'ensemble des deux questions 1 et 3]**

**Le fichier comme tout fichier, utilise 1 seul i-nœud.**

4. Quels sont les nombres minimaux et maximaux d'**extents** utilisés par ce fichier (`The_Glorious_Land.txt`)? Justifier.

**[1 pt]**

**1 datablock = 1 et 1 seul extent.**