USTHB **Le 13/01/2015**
Faculté d'Electronique et Informatique **Année 2014/2015**
Département Informatique **Master SSI**

Examen de Systèmes d'Exploitation
Master SSI 1ère année
Durée : 1h30 – Documents non autorisés

Exercice 1 : (10 points)

Un processus lit séquentiellement un fichier de 8 Mo, à raison de 256 octets à la fois. On suppose que les blocs disque sont de 1024 octets et qu'un numéro de bloc occupe 4 octets. Par ailleurs, le temps d'accès moyen au disque est de 40 ms.

1. Donnez la structure d'une inode et d'un fichier Unix
2. Le système ne gère pas de mécanisme de buffer cache. Donnez le nombre total d'accès disque nécessaire et le temps d'attente en entrées/sorties
3. Le système gère un mécanisme de buffer cache

a. Rappelez le fonctionnement de ce mécanisme. Pourquoi la gestion de remplacement est-elle LRU plutôt que FIFO ?
b. Donnez le nombre total d'accès disque nécessaire et le temps d'attente en entrées/sorties
c. L'écriture physique des blocs modifiés ne se fait que lorsqu'un bloc du buffer cache doit être libéré. Quel avantage et inconvénient cela présente-t-il ?

Exercice 2 : (10 points)

Le jeu de la patate chaude peut se résumer ainsi :

Soit « A » un arbitre ; soient 4 joueurs « J0 », « J1 », « J2 », et « J3 » qui ne se connaissent pas au départ. À noter qu'initialement, le score de chaque joueur vaut 0 ; au début de la partie, l'arbitre donne l'identité des autres joueurs à chaque joueur ; puis l'arbitre désigne un joueur au hasard et lui envoie la patate chaude ; dès qu'un joueur reçoit la patate, il incrémente son score de 1. Si le score est inférieur à 1000, il tire au hasard un n° de joueur (il ne doit pas se choisir lui-même) et lui envoie la patate chaude. Sinon, si le score est égal à 1000, il envoie la patate à l'arbitre ; quand l'arbitre reçoit finalement la patate (devenue froide), il demande à chaque joueur d'afficher son score, et la partie s'arrête.

Question 1): Écrire en C (ou éventuellement en pseudo-code) le jeu de la patate chaude, selon les indications suivantes :

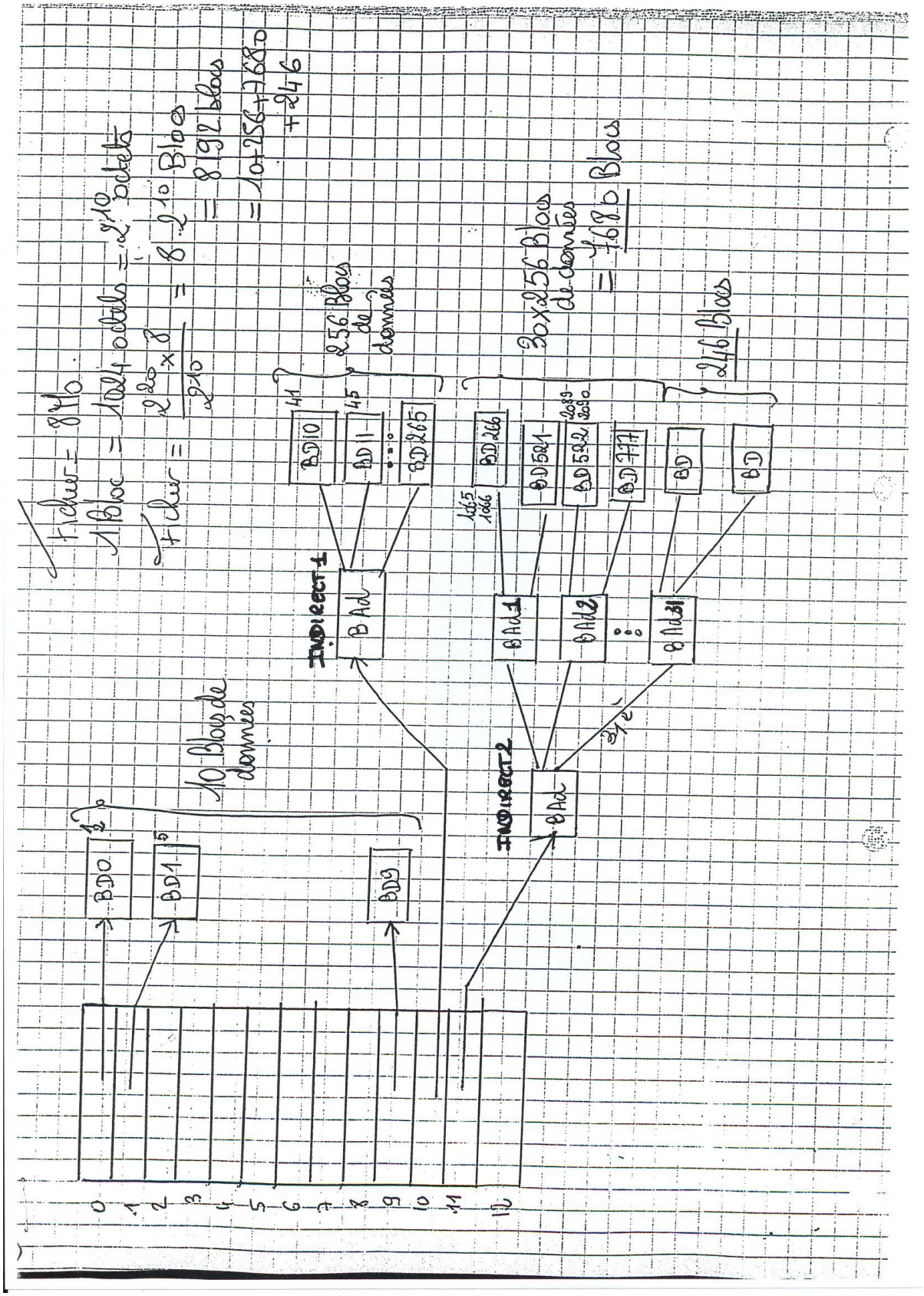
- les joueurs « J0 », « J1 », « J2 », et « J3 » sont des processus fils de l'arbitre « A » ;
- en début de partie, « A » communique le PID des autres joueurs via un pipe anonyme ;
- le tirage aléatoire d'un joueur se fait en utilisant la fonction int num_joueur_aleat() qui retourne un numéro aléatoire compris entre 0 et 3. À titre d'exemple, voici ce que pourrait être cette fonction :

```
#include <stdlib.h>
int num_joueur_aleat()
{
long int li;
li=lrand48()%4;
return((int)li);
}
```

- l'envoi de la patate chaude est simulé par l'envoi d'un signal utilisateur SIGUSR1 (pour les 4 joueurs comme pour le retour de la patate chaude à l'arbitre en fin de partie) ;
- en fin de partie, l'arbitre demande à chaque joueur d'afficher son score (ce qui sous-entend la fin de la partie) par l'envoi d'un signal SIGUSR2.

Corrige

Exercice 1 : (10 points)



Un processus lit séquentiellement un fichier de 8 Mo, à raison de 256 octets à la fois. On suppose que les blocs disque sont de 1024 octets et qu'un numéro de bloc occupe 4 octets. Par ailleurs, le temps d'accès moyen au disque est de 40 ms.
1/ Rappelez la structure d'une inode et d'un fichier Unix

2/ Le système ne gère pas de mécanisme de buffer cache.
Donnez le nombre total d'accès disque nécessaire et le temps d'attente en entrées/sorties

lecture des 10 premiers blocs : 4 * 10 accès disque
lecture des 256 blocs suivants (niveau d'indirection 1) : on a deux accès disque par lecture
8 * 256 accès disque
lecture des 7926 blocs restants (niveau d'indirection 2) : on a trois accès disque par lecture
12 * 7926 accès disque
soit un total de 97200 accès disque et une durée moyenne de lecture égale à 3888 s.

3/ Le système gère un mécanisme de buffer cache
a/ Rappelez le fonctionnement de ce mécanisme. Pourquoi la gestion de remplacement est-elle LRU plutôt que FIFO ?
b/ Donnez le nombre total d'accès disque nécessaire et le temps d'attente en entrées/sorties
c/ L'écriture physique des blocs modifiés ne se fait que lorsqu'un bloc du buffer cache doit être libéré. Quel avantage et inconvénient cela présente-t-il ?

3/ Le système gère un mécanisme de buffer cache
a/ question de cours
b/ on a un accès disque par blocs de données (lors de la lecture des 256 premiers octets du bloc) on a un total de 33 blocs d'adresse à lire.
soit un nombre d'accès disque égal à 8192 + 33 = 9224 et un temps moyen de lecture égal à seulement 329 s !!!!
c/ avantage : on économise les accès disque
inconvénient : risque de perte de données si plantage de la machine.
l'appel SYNC force le vidage du cache sur le disque.

Exercice 2 : (10 points)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
int tab_pid[4]; // contiendra le PID de chaque joueur
int p[4][2]; // les 4 pipes anonymes
int score; // le score du processus
int nb_joueur;
int num_joueur_aleat()
{
long int li;
li=lrand48()%4;
return((int)li);
```

```
}
void reception_patate_fils()
{
int prochain_joueur;
signal(SIGUSR1, reception_patate_fils); // on réarme
score++;
if (score < 1000)
{
// on retire un nombre aleatoire tant qu'on se choisit nous même
do {
prochain_joueur=num_joueur_aleat();
} while (prochain_joueur==nb_joueur);
kill(tab_pid[prochain_joueur], SIGUSR1);
}
else
{
// fin de partie, on envoie la patate au père
kill(getppid(), SIGUSR1);
}
}
void reception_patate_pere()
{
// quand le père reçoit la patate, il envoie le signal SIGUSR2 aux 4
// fils, puis il récupère leur valeur de fin pour éviter les zombies,
// puis s'en va...
int i;
for (i=0 ; i<4 ; i++)
kill(tab_pid[i], SIGUSR2);
// on récupère les valeurs de exit() des fils pour éviter les zombies...
for (i=0 ; i<4 ; i++)
wait(NULL);
exit(0);
}
void recoit_fin_partie()
{
printf("Joueur %d, score : %d\n", nb_joueur, score);
exit(0);
}
int main()
{
int i, j;
// Création de 4 pipes anonymes
for (i=0 ; i<4 ; i++)
pipe(p[i]);
// initialisation du score pour tous
score=0;
// Création des 4 joueurs
nb_joueur=0;
while ( (nb_joueur<4) && ((tab_pid[nb_joueur]=fork())>0) )
nb_joueur++;
if (nb_joueur==4) // on a reçu 4x un nbre >0 au fork() => on est le père
{
signal(SIGUSR1, reception_patate_pere);
// on ferme les pipes en réception :
for (i=0 ; i<4 ; i++)
close(p[i][0]);
// pour chacun des 4 fils, on envoie les 4 PID
for (i=0 ; i<4 ; i++)
for (j=0 ; j<4 ; j++)
write(p[i][1], &(tab_pid[j]), sizeof(tab_pid[j]));
// on ferme les pipes d'émission
```

```
for (i=0 ; i<4 ; i++)
close(p[i][1]);
// on envoie la patate ;-)
i=num_joueur_aleat();
kill(tab_pid[i], SIGUSR1);
for (;;) ;
}
else
{
// code des fils (NB: chaque fils est le joueur nb_joueur)
signal(SIGUSR1, reception_patate_fils);
signal(SIGUSR2, recoit_fin_partie);
// on ferme les pipes d'émission
for (i=0 ; i<4 ; i++)
close(p[i][1]);
// on reçoit les pid des joueurs
for (i=0 ; i<4 ; i++)
read(p[nb_joueur][0], &(tab_pid[i]), sizeof(tab_pid[i]));
// on ferme les pipes de réception :
for (i=0 ; i<4 ; i++)
close(p[i][0]);
for (;;) ;
}
}
```