

PRÉPARATION A L'EXAMEN D'UNIX

Système d'exploitation :

Sous Unix, tous les processus sont des processus

- Démons
- **Réentrants**
- Zombies

Si un processus possède les droits d'accès 0555, son propriétaire est « root » et est dans le répertoire « /usr/bin »,

- **Tous les utilisateurs peuvent l'exécuter**
- Seul le SU peut l'exécuter
- Seul les utilisateurs appartenant au groupe « système » peuvent l'exécuter

La fonction wait() permet


- D'attendre la fin d'un fils bien particulier
- **D'attendre la fin d'un fils quelconque**
- La frappe d'un caractère ou la réception d'un signal quelconque

Lorsqu'un signal est envoyé à un processus père par un de ses fils,


- Tous les processus (père et fils) reçoivent le signal
- **Seul le père reçoit le signal**
- Tous les processus (père et fils) reçoivent le signal, sauf du celui qui la émit

Sous UNIX, tous les processus passent par l'état :

- Démon
- Réentrant
- **Zombie**

 Pour rechercher une chaîne de caractères « xyz » dans un ensemble de fichiers, il faut utiliser la commande suivante :

- find . -name « xyz » - print
- **grep xyz *.***

 Pour obtenir l'aide de la fonction fork (), il faut utiliser la commande suivante :

- **man fork**
- man fork ()
- man 2 fork

Le code de retour de la fonction fork () est, pour le processus père :

- 0
- **Le pid du fils**
- Son propre pid

Après un exec (), il faut :

- Tester le code de retour de la fonction, pour s'assurer qu'il n'y a pas de problème
- Rien, car soit elle s'est bien déroulée, soit le programme s'arrête
- **Traiter l'erreur**

Un i-noeud est associé à un fichier de façon unique :

- **Vrai**
- Faux, il peut y en avoir plusieurs (Obtenu par un lien)
- Vrai, mais il faut le préciser à la création

Les fichiers :

Soit le fichier Examen.txt. Son contenu est :

```
abcdefghijklmnopqrstuvwxyz
```

Il sera utilisé pour les exemples de la suite.

Soit le programme suivant :

```
...
hd = open("Examen.txt",O_RDWR) ;
hd1 = open("Examen.txt",O_RDWR) ;
write(hd,"123",3);
write(hd1,"456",3);
...
```

Le fichier Examen.txt est

- 123456ghijklmnopqrstuvwxyz
- **456defghijklmnopqrstuvwxyz**
- 123defghijklmnopqrstuvwxyz

Soit le programme suivant :

```
...
hd = open("Examen.txt",O_RDWR) ;
lseek(hd,100,0) ;
rc = write(hd,"123",3);
...
```

rc = ?

- **3**
- -1 (cas d'erreur)
- 0

Soit le programme suivant :

```
...
hd = open("Examen.txt",O_RDWR|O_APPEND) ;
lseek(hd,3,0) ;
write(hd,"XYZ",3);
...
```

Le fichier Examen.txt est

- abcXYZghijklmnopqrstuvwxyz
- abcdefghijklmnopqrstuvwxyz (car il y a une erreur)
- **abcdefghijklmnopqrstuvwxyzXYZ** //O_APPEND : écriture en fin de fichier

Soit le programme suivant :

```
...
hd = open ("Examen.txt",O_RDWR);
lseek (hd,100,0);
rc = read (hd ,szBuffer,3);
...
```

rc = ?

- 3.
- 1. (Cas d'erreur)
- L'arrêt du programme.
- **0**

Soit le programme suivant :

```
...
hd = open ("Examen.txt",O_RDWR);
idFils = fork ();
if (idFils)
{
    write (hd,"XYZ",3);
    exit (1);
}
write (hd, " ABC " ,3 );
exit (0);
```

Le fichier Examen.txt est:

- **ABCXYZghijklmnopqrstuvwxyz (ou XYZABCghijklmnopqrstuvwxyz)**
- abcdefghijklmnopqrstuvwxyzABCXYZ
- XYZdefghijklmnopqrstuvwxyz

Soit le programme suivant :

```
...
hd = open ("Examen.txt",O_RDWR);
hd1 = dup (hd);
lseek (hd,3 ,0);
read (hd ,szBuffer,3);
read (hd1, szBuffer1,3);
exit (0);
```

szBuffer = ?

- **szBuffer = "def" szBuffer1 = "ghi"**
- szBuffer = "def" szBuffer1 = "def"
- szBuffer = "def" szBuffer1 = "abc"

Soit le programme suivant :

```
...
hd = open ("Examen.txt",O_RDWR);
lseek (hd,20,0);
rc = read (hd ,szBuffer, 10);
```

rc = ?

- 1 (Cas d'erreur)
- 10
- **6**

Un processus a bloqué 1 fichier avec 1 verrou partagé et il se termine suite à 1 SIGQUIT. Le fichier garde son verrou :

- Vrai.
- **Faux**
- Le verrou est levé uniquement si le programme l'a prévu

Un fichier ordinaire sous UNIX est :

- Un fichier source
- Un fichier exécutable
- Un fichier de données
- Aucun des trois
- **Les trois**

Un fichier spécial, sous UNIX, caractérise :

- Un exécutable
- **Un fichier associé à un périphérique**
- Un fichier système

Un fichier FIFO (un tube) sert :

- À stocker des données
- À mémoriser des informations utiles au noyau
- **A la communication entre processus**

Un fichier peut être supprimé sur disque par :

- Uniquement le SU
- Uniquement le propriétaire
- Le propriétaire et le SU
- **Par toutes personnes ayant les droits d'accès en écriture sur le fichier**

Etant donné 1 fichier ouvert en O_RDWR, il est possible de se positionner au-delà de la fin de fichier via la fonction lseek() :

- Jamais
- **Sans problème**

Un verrou exclusif posé sur un enregistrement empêche le processus propriétaire de manipuler cet enregistrement :

- **Faux**
- Vrai c'est en fait une sécurité
- Vrai, dans certains cas

Lorsqu'un processus veut modifier un verrou partagé en un verrou exclusif, il peut le faire directement :

- Vrai, il fait ce qu'il veut de ce qui lui appartient
- **Faux, car dans ce cas il crée une situation de dead lock**
- Vrai, mais il doit être sûr qu'un autre processus ne manipule pas cet enregistrement

Sur un même enregistrement, un processus peut placer un verrou partagé et un verrou exclusif :

- Vrai
- **Faux, car il ne peut exister 2 verrous différents sur le même enregistrement**
- Vrai mais seul le dernier verrou posé existe

La pose d'un verrou exclusif sur un enregistrement empêche un autre processus de manipuler cet enregistrement :

- Toujours vrai
- **Vrai, uniquement si l'autre processus teste l'existence du verrou**
- Faux, l'autre processus fait ce qu'il veut

On peut placer sur un enregistrement un verrou partagé et un verrou exclusif :

- Vrai, par définition d'un verrou partagé
- **Faux, en aucun cas //mais plusieurs verrous de ce type (partage) peuvent être mis**

La fonction dup () alloue un nouveau descripteur dans la table des fichiers ouverts, ce dernier :

- **Il aura la même entrée dans la table des fichiers ouverts //Les 2 descripteurs pointent sur la même entrée dans la table des fichiers ouverts. Mais le descripteur a une nouvelle entrée dans la table des descripteurs**
- Il aura une entrée différente dans la table des fichiers ouverts, mais accèdera au même fichier
- Ça n'aura aucun effet, car le fichier est déjà ouvert une fois

Lors de la création d'un fichier avec les droits d'accès 0777, on obtient, sur disque, les droits suivants:

- 0777
- **0755**
- 0700

Dans le cadre de l'institut, un étudiant du même groupe que vous peut :

- Lire et écrire dans vos fichiers dont vous avez donné les droits 0660
- **Ne peut pas lire vos fichiers même si vous avez donné les droits 0660. Car les droits d'accès de votre répertoire lui interdisent)**

Les terminaux sont des périphériques dont on peut modifier :

- Tous les paramètres indépendamment les uns des autres
- ~~Certains paramètres à conditions qu'ils soient compatibles avec les autres~~
- Aucuns paramètres, sauf si l'on est SU

Lors du login, votre terminal se trouve :

- **En mode canonique**
- En mode non canonique

Lors du login, un terminal vous est attribué :

- De façon aléatoire mais vous l'aurez toujours, car vous êtes connu du système
- De façon fixe, car il est réservé dans le fichier des mots de passe et vous obtiendrez toujours le même
- **De façon aléatoire, en fonction des ressources du système**

Les Pipes :

Un processus reçoit un signal SIGPIPE :

- Lorsqu'il tente d'écrire dans un pipe plein
- **Lorsqu'il tente d'écrire dans un pipe non ouvert en lecture**
- Jamais

Un processus sait qu'il n'y a rien à lire dans un tube et par conséquent ne reste pas sur un read () car :

- Il n'y a plus assez de caractères dans le tube
- **Le tube est fermé en écriture par tous les processus qui le manipulent**
- Il ne le sait pas, il faut lui envoyer un signal

Un processus qui tente d'écrire dans un pipe plein :

- Est interrompu, car il y a erreur d'écriture, et il reçoit un SIGPIPE
- **Est bloqué sur l'instruction write**
- Continue son exécution, mais n'a pas écrit dans le tube

2 processus indépendants peuvent communiquer par pipe :


- **Faux, si c'est un pipe classique // Vrai si c'est un pipe nommé**
- Vrai, il suffit d'ouvrir le pipe par la fonction fcntl ()
- Vrai, le pipe étant propriété du système, on peut toujours l'ouvrir

Un processus peut ouvrir un pipe après l'avoir fermé :

- Vrai grâce à la fonction fcntl ()
- **Faux, en aucun cas**
- Vrai grâce à la fonction open ()

Un processus exécute l'instruction :

```
read (hdPipe,szBuffer, 10); //alors qu'il n'y a que 5 caractères dans le pipe
```

- 
- **~~Le processus est en attente, car la lecture est bloquante~~**
 - **Le processus lit les 5 caractères sans problème et continue son exécution**
 - Le processus s'interrompt, car il y a une erreur

Un étudiant de l'institut pourra communiquer par tube nommé avec son coéquipier :

- Toujours, si le tube nommé a les droits correspondants
- **Oui, à condition que le tube possède les droits d'accès et que le tube soit créé dans un répertoire commun**
- Jamais, car c'est voulu ainsi dans l'institut

Les I.P.C :

Les I.P.C. sont supprimés automatiquement à la fin du processus :

- Vrai
- **Faux**
- Vrai mais uniquement lorsque tous les processus qui les manipulent sont terminés

On peut manipuler un ensemble de sémaphores en une seule opération :

- Faux, il faut boucler sur tous les éléments de l'ensemble
- Vrai, mais si le programme est interrompu par la réception d'un signal, l'ensemble peut être incohérent
- **Vrai, et toutes les valeurs de l'ensemble seront correctes, car l'opération est atomique**

Si l'on manipule 1 ensemble de sémaphores avec semop () & que sur 1 d'entre eux on ne peut effectuer l'opération :

- Les autres sémaphores seront modifiés
- **L'opération est bloquante jusqu'à la possibilité de manipuler l'ensemble complet**
- L'opération ne sera parfaite, et le programme continue son exécution, il faut prévoir le cas

On peut ouvrir un certain nombre de files de messages, ce nombre est :

- Illimité, il suffit que les clés soient différentes
- **Ce nombre est limité par le système**
- Ce nombre est limité par le système et est le même pour tous les utilisateurs

Le propriétaire peut supprimer une file de messages à tout moment :

- **Vrai**
- Faux
- Vrai, mais uniquement lorsque tous les processus qui les manipulent sont terminés

Le propriétaire peut supprimer une mémoire partagée à tout moment :

- **Vrai, mais uniquement lorsque tous les processus qui lui sont attachés sont terminés**
- Faux
- Vrai

Un sémaphore peut avoir :


- **Une valeur entière positive (ou = 0)**
- Une valeur entière quelconque
- Une valeur réelle

Si un processus endormit sur un appel système reçoit un signal :

- Il ne fait rien, le signal est suspendu pendant l'appel système
- Il interprète le signal, et renvoie une erreur
- **Il interprète le signal, renvoie une erreur et positionne errno**

Lorsqu'un processus reçoit un signal :

- **Il interprète le signal**
- Il interrompt la fonction, interprète le signal et positionne errno à EINTR
- Il positionne errno à EINTR et continue son exécution

 Un proc exécute le handler assigné à un signal, le comportement du signal n'est pas modifié, il reçoit encore le signal :

- Ce 2ème signal est suspendu
- ~~Ce 2ème signal est perdu~~
- Ce 2ème signal directement interprété

Lorsqu'un processus reçoit un signal, il peut déterminer le processus qui l'a émis :

- Toujours vrai
- **Impossible**