IIE++ : noyau et shell

Sylvain Maret

12 décembre 2018

Plan

Plan

Vous savez quoi ?

Avant de commencer la ++ j'en profite pour préciser les choses :

- Ceci ne remplace pas le cours
- C'est juste pour vous aider
- Les slides seront disponibles sur mon perso maret.iiens.net/ppdenoyau.pdf
- posez des questions dès que possible
- Je me suis basé sur les slides de Yoliste et de Titch pour faire ce ++

Vrai intro

Alors pour commencer, les appels noyaux ça sert à :

- Manipuler des fichiers/flux.
- créer/gérer des processus
- Comuniquer entre différents processus

Vrai intro

le shell c'est :

- l'outil de base pour communiquer avec la machine
- très puissant et très utile

Vrai intro

Deux trois conseils :

- vous avez le droit à une feuille recto pour le cours (je crois)
- lisez le man pour faire votre fiche et notez ce qui vous parait important (ou en lien avec vos TPs précédents) et assurez vous d'être sur la fonction en C.
- refaire les TPs.

La fiche

Pour le fiche pensez aussi à :

- mettre des exemples de code
- mettre le type des arguments d'une fonction

Plan

Protips

- Pour un script :
  # !/bin/bash
- Ensuite :
  chmod +700 mon_joli_script.sh
  ./mon_joli_script.sh
- Accéder aux arguments d'un script :
  ./mon_joli_script.sh arg1 arg2 arg3
  $0 mon_joli_script.sh
  $1 arg1
  $2 arg2
  ...
  $# : Nombre d'arguments $* : listes des arguments

Protips

Entrée :

- a=2 #pas d'espace
- echo a
- echo $a
- echo "$a"
- echo '$a'

Sortie :

- (rien)
- a
- 2
- 2
- $a

Protips

Récupérer le résultat d'un commande :
a='ls'
Évaluer une expression : compteur++ ?
compteur=$(($compteur+1))

Conditions

Syntaxe :
if [[cond]] ;
then
machin
else
autre machin
fi

Evaluation :
eq : equal
ne : not equal
gt : greater than
lt : lower than
ge : greater equal
le : lower equal

Exemple :
```
if[[$# -ne 3]] ;
then
echo "fail !"
else
echo "success"
fi
```

Boucles

While :

```
a=$#
while [[ $a -gt 0 ]] ;
do
echo $a
a=$(($a))
done
```

For :

```
for f in $*
do
echo $f
done
```

Redirection

Trois grandes possibilités : < > et |

- cmd > fichier
  écrit la sortie de cmd dans le fichier au lieu de stdout
  > : écrase le fichier
  » : ajoute à la fin du fichier
- cmd < fichier
  passe le contenu de fichier en argument de cmd
- cmd | cmd2
  passe la sortie de cmd en argument de cmd2
  echo bonjour | cat

Que fait cette commande ?
ls -R | grep "\."c$ > toto.txt

Plan

Appels système

Deux moyens de les réaliser :

- directement.
- en passant par les instructions de la libc(fopen, fread...).

A quoi ça sert ?

- Passer du mode user au mode système.
- exécuter des fonctions systèmes

Les fonctions utiles

- open(path, flags) : ouvre le fichier donné dans l'argument "path" avec les options données par les flags, renvoie le file descriptor assigné au fichier.
- close(file_descriptor) : ferme le fichier ayant le file descriptor donné en argument
- read(fd, buffer, count) : lit jusqu'à count caractères dans le fichier désigné par fd, et stocke les caractères lus dans buffer
- write(fd, buffer, count) : écrit count caractères depuis buffer dans le fichier désigné par fd.
- lseek(fd, offset, whence) : permet deplacerle curseur du flux à une position whence selon un offset(exemple lire un fichier en commencant à la ligne 4).

Il existe bien sur des fonctions de la libc pour réaliser les mêmes commandes : fileno, fflush,...

Les descripteurs de flux

- décrit le flux crée (par des fonctions tels que open).
- sert à réutiliser les flux plus tard.

Les flux les plus connus sont 0,1 et 2.
Pensez à libérer vos flux à la fin de votre code.

Fonctions utiles

- fork() : créé un nouveau processus, renvoie le PID de ce processus (Process IDentifier) dans le processus père, et renvoie 0 dans le processus fils.
- kill(pid, sig) : envoie le signal sig au processus spécifié par le PID
- signal(sig, fonction) : permet de définir un gestionnaire de signal : lorsque le programme recevra le signal sig, il appellera fonction pour traiter le signal.
- getpid() : renvoie le PID du processus qui appelle cette fonction.

Définition

Principe : utiliser un fichier pour la communication entre deux processus. L'un lit dans le pipe, l'autre lit ce qui est écrit par le premier. On écrit/lit dans les pipes avec read() et write()

Définition

Deux types de pipes différents :

- Pipes nommés : fichiers créés sur le système avec la fonction mkfifo(nom,droits sur le fichier) en C ou la commande bash mkfifo.
  - Avantage : accessible depuis n'importe quel processus.
  - Inconvénient : créé un fichier sur le système.
- Pipes non nommés : créés avec la fonction pipe(int fd[2]). L'argument est un tableau d'int de taille (au moins) 2, dans lequel seront mis 2 file descriptors : un pour la lecture du pipe (fd[0]) et un pour l'écriture dans le pipe (fd[1]).
  - Avantage : tout se passe en mémoire, pas de création de fichier
  - Inconvénient : accessible uniquement par deux processus liés entre eux.

Plan

Des questions ? Just Google it ! ! ! !

On va refaire quelques exos basique de vos TPs.