Cours linux

Gestion des processus

alan.simon@free.fr

Présentation 1/5

Time sharing:

Le Kernel ordonnance l'execution de processus

- Processus kernel (thread du kernel)
- Processus système (services)
- Processus utilisateurs

Chaque processus vas disposer du CPU à son tour pour être executé

Mis a part init chaque processus est initié par un autre processus (son père) par la fonction fork

Présentation 2/5

Fonction fork :

- Créer un processus fils identique au processus courant

Fonction exec :

- Remplace l'executable courant par un autre

Au lancement d'un processus on appel ces deux fonctions successivement

- Ainsi le fils hérite du contexte d'exécution du père sauf pour la valeur du PID et du PPID : Process identity.

Présentation 3/5

Contexte d'execution

Zone mémoire :

- Binaire
- Donnée

Variable du processus :

- Répertoire courant
- Uid, gid
- Termial de ratachement
- Priorité
- Etc....

L'environment Unix : les variables hérité (exportées)

Présentation 4/5

Caractéristiques notable

- PID : identifiant du processus.
- PPID :identité du processus père (créateur).
- UID, EUID, SUID : identifiant(s) du (des) propriétaire(s).
- GID, EGID, SGID : identifiant(s) du groupe.
- tty : terminal d'attachement.
- pri :priorité
- Nice :prorité utilisateur
- s : statut du processus (endormi, en attente, en exécution, etc.)
- stime :date de lancement du processus
- time :temps écoulé d'utilisation d'unité centrale
- size :taille de la mémoire allouée.

Présentation 5/5

Statut du processus:

- Running
- Zombie : processus inactif voulant rendre la main a son père qui ne l'attend pas
- Waiting : Processus en attente d'une ressource
- Dying (en train d'écrire un fichier core)
- Swapped (placé en swap)
- Stopped : [Ctrl D]

Les entrées sorties

Chaque processus dispose au minimum des flux de données suivants

- Une entrée standard (0)
- Une sortie standard (1)
- Une sortie d'erreur (2)

Chaque flux dispose d'un descripteur de fichier (un numérique)

Gestion sous shell :

- > fichier (ou descripteur de fichier : >&2)
- | processus
- < fichier
- 2>&1

Les signaux

Kill est la commande permattant de transmettre un signal au processus

Cmd “man kill”

1 HUP (ton père est mort ou relis ta conf)

2 INT (Contrôle C)

3 QUIT

6 ABRT

9 KILL (envoyé directement au kernel)

15 TERM

18 SIGSTP (Contrôle Z)

Un signal est envoyé au processus que celui-ci traite comme une interuption (sauf kill -9)

Trap en shell permet de traiter ces signaux

Cmd “man trap”

Cmd “man kill”

Attention “Contrôle D” n'est pas un signal : [EOF]

Commandes shell : Lancer des processus

cmd : execute la commande : cmd

- C'est un processus fils
- Il hérite de l'environement du shell courant
- Il utilisent les même sortie que le shell courant

“&” à la fin d'une commande

- lance la commande en fond de tache
- Attention a bien redirigé les sorties std et err

Cmd “exec” lance un processus à la place du processus courant (sans fork)

Cmd “nohup cmd”

- lance la commande cmd en trappant le signal HUP
- Détache la sortie standard du termibal courant (fichier nohup.out si non définie)

Cmd “(cmd ; cmd)” lance un sous shell exécutant les commandes cmd puis cmd

Commandes shell
Contrôle

“ps” : liste les processus en cours “ps -ef” tout les processes “ps aux” ou “ps ajx” sous bsd

“jobs” list les jobs en fond de tache

“fg $i” ForeGroung remet un processus au premier plan

“bg $i” BackGround relance un processus stoppé en fond de tache

“wait” attend la fin du tout les processus fils

“exit n” fin d'exécution avec renvois le code retour d'execution au processus parent

“nice” “renice” permet de definir/redefinir la priorité du processus

En pratique : gestion des jobs

Tappez : “(sleep 50 ; echo fin 1)” puis “[ctrl]z”

Tappez : “jobs”

Tappez “(seep 45 ; echo fin 2)” puis “[ctrl]z”

Tappez : “jobs”

```
[1]-  Stopped                 ( sleep 30; echo fin sleep 30 )
[2]+  Stopped                 ( sleep 25; echo fin sleep 25 )
```

Tappez succesivement : “bg” “bg” “wait”

/proc

Pseudo filesystem présentant l'état de la tables des processus

- Tappez : “ps” récupèrer le pid de votre shell courant
- Tappez : “ls -ald /proc/$$/cwd”
- Tappez : “cd /etc ; ls -ald /proc/$$/cwd”
- Tappez : “env”
- comparrez avec : “cat /proc/$$/environ”

- Tappez : “ls -ald /proc/$$/exe”

Scripting présentation

Un script est un fichier texte exécutable

- Première ligne : le sha-bang : #!/usr/bin/$interpreteur $argument
- Suite de commandes ou le programme transmis à l'interpreteur

Environement : variable courante et variable interne au shell

- $$ = PID
- $0 = commande courante (l'appel du script)
- $1... les arguments du script
- $PPID
- $PWD
- $? : code retour dernière commande
- $! : pid dernière commande

En pratique “nohup”

Créez un fichier tst.sh et rendez le executable

```
#!/bin/bash
sleep 500
echo fin
exit 0
```

Ouvrez 2 terminaux

Sur le premier relancez bash une seconde fois et identifier son Pid “ps -f”

puis la commande “tst.sh”

Sur le second kill -1 du pid du second bash

Le script tst.sh tourne-t-il toujours ?

“ps -ef | grep tst.sh”

Refaite le même test avec la commande “nohup tst.sh”

Contstat ?

Killez le processus sleep avec le signal SIGINT (-2)

TD

Faire un script launch qui lance un processus en tache de fond

- Ce script prendra en argument, le répertoire de lancement, le programme à lancer, et ses arguments
- Ce script redirigera les sorties std et erreur vers un fichier de log dans /var/log/batch-log/$nom-du-processus-YYMMDDHHMM