Scripts - Processus

Cours 5

E. Paviot-Adet

Emmanuel.Paviot-Adet@parisdescartes.fr
Bureau B2-2

IUT Paris Descartes

DUT Informatique - 1ère année

Sommaire

Plan

Structure générale

Script

Fichier texte contenant des commandes

- Droits en exécution
  - Pour rendre le fichier exécutable
- Première ligne
  - Peut indiquer le shell utilisé pour exécuter le script
    - Syntaxe : #!*Nom_exécutable*
    - Exemple : *#!/bin/bash*
    - Nom : *shebang* ou *hashbang*
    - Optionnelle
- Ligne commençant par #
  - Commentaire
  - Exception : le shebang

Utilisation de la commande *set* dans un script

set -x Affiche les commandes et leurs arguments avant leur exécution

- Doit être mis dans le script
- Permet de suivre le déroulement du script
  - Utile pour trouver d'éventuelles erreurs

set -u Une erreur est retournée dès qu'une variable non définie est utilisée

- Par défaut, la variable est remplacée par la chaîne vide

- Retour à la normale
  - *set +x*
  - *set +u*

Paramètres

Paramètres

Informations textuelles ajoutées après le nom d'une commande

- Chaîne de caractères ayant une sémantique propre à chaque commande et chaque paramètre
- Permet à l'utilisateur de fournir des informations spécifiques à chaque exécution du script
- Exemples :
  - *ls /tmp*
    - */tmp* est un paramètre le répertoire à utiliser pour *ls*
  - *ls -l*
    - *-l* est un paramètre modifiant le comportement de *ls*
    - Fait partie de la syntaxe d'utilisation de la commande
- Vos scripts aussi peuvent utiliser des paramètres

Variables spécifiques

$* Liste des paramètres
- Sans le nom de la commande

$# Nombre de paramètres
- Sans compter la commande

$0 Le nom de la commande

$1, $2... $9 Paramètres 1, 2... 9

${10}... Paramètre 10...

Exemple de script

```
#!/bin/bash                                  # Shebang
#Nom du script : test_param                  # Commentaire
#Affichage des valeurs des paramètres        # Commentaire
echo Nombre de paramètres : $#
echo Paramètres : $*
echo Nom de la commande : $0
echo Premier paramètre : $1
```

Création et utilisation du script précédent

Création

Dans un éditeur de texte

- Pas d'information de mise en forme
- Attribution des droits en exécution
  - On suppose qu'il s'appelle *script*
  - *chmod u+x script*

Exécution

- Ligne de commande : *./script p1 p2*
  - *./* pour indiquer à l'invite de commande que le script se trouve dans le répertoire courant

Création et utilisation du script précédent

Création

Dans un éditeur de texte

- Pas d'information de mise en forme
- Attribution des droits en exécution
  - On suppose qu'il s'appelle *script*
  - *chmod u+x script*

Exécution

- Ligne de commande : *./script p1 p2*
  - *./* pour indiquer à l'invite de commande que le script se trouve dans le répertoire courant

Affichage du script précédent

Ligne de commande : *./script p1 p2*

```
Nombre de paramètres : 2
Paramètres : p1 p2
Nom de la commande : ./script
Premier paramètre : p1
```

Plan

Définition

Un *processus* est constitué de :

- Un programme à exécuter
  - En mémoire centrale
- Données
  - En mémoire centrale
- Descripteur de processus
  - Descripteurs de ressources allouées
    - Tables d'identification des zones mémoires utilisées
    - Table des fichiers ouverts
    - ...
  - Attributs de sécurité
    - Nom du propriétaire
    - Droits
  - Contexte
    - Image de l'état des registres

Création de processus

Sous Unix :

- Création à partir de processus existants
  - Par un système de clonage
    - Processus créé est appelé *processus fils*
    - Processus initiateur est appelé *processus père*
    - *fork* en C (valeur retournée permet de les distinguer)
  - Un processus peut choisir d'exécuter un nouveau programme
    - *exec* en C
- Certains processus n'ont pas de père
  - Processus *init* créé au lancement du système
    - Ne se termine pas et porte le numéro 1
  - Les *daemons*
    - Chargés de rendre différents services

Terminaison

Règle générale

A la terminaison, un processus retourne une valeur à son père.

Cas particuliers

- Le père termine avant le fils
  - Le processus *init* devient le père
- Le père ne récupère pas la valeur retournée par le fils
  - Le descripteur du fils reste dans le système
  - On appelle ça un processus *zombie*
  - Ce processus ne peut pas être tué directement
  - Seule solution : tuer le père

Terminaison

Règle générale

A la terminaison, un processus retourne une valeur à son père.

Cas particuliers

- Le père termine avant le fils
  - Le processus *init* devient le père
- Le père ne récupère pas la valeur retournée par le fils
  - Le descripteur du fils reste dans le système
  - On appelle ça un processus *zombie*
  - Ce processus ne peut pas être tué directement
  - Seule solution : tuer le père

Temps partagé

Définition

Un système est dit à *temps partagé* lorsqu'il partage la puissance de calcul entre différents processus pour donner l'impression qu'ils progressent simultanément

Mise en œuvre

- Allouer un core à un processus pour un temps déterminé
  - *Quantum de temps*
- Ce temps terminé, passage à un autre processus
  - *Commutation de processus* (ou *de tâches*)
- Le changement de processus prend du temps
  - *Temps de commutation*

Temps partagé

Définition

Un système est dit à *temps partagé* lorsqu'il partage la puissance de calcul entre différents processus pour donner l'impression qu'ils progressent simultanément

Mise en œuvre

- Allouer un core à un processus pour un temps déterminé
  - *Quantum de temps*
- Ce temps terminé, passage à un autre processus
  - *Commutation de processus* (ou *de tâches*)
- Le changement de processus prend du temps
  - *Temps de commutation*

Quantum et commutation

Principes à respecter

- Quantum de temps grand devant le temps de commutation
  - Sinon l'ordinateur passe trop de temps à changer de processus
- Quantum de temps petit aux yeux de l'utilisateur
  - Sinon ce dernier a l'impression que son processus avance par à-coups

Causes de commutation de tâches

- Fin du quantum de temps
- Attente sur un périphérique

Quantum et commutation

Principes à respecter

- Quantum de temps grand devant le temps de commutation
  - Sinon l'ordinateur passe trop de temps à changer de processus
- Quantum de temps petit aux yeux de l'utilisateur
  - Sinon ce dernier a l'impression que son processus avance par à-coups

Causes de commutation de tâches

- Fin du quantum de temps
- Attente sur un périphérique

Commutation de tâches

Problème

Comment interrompre, puis reprendre, un processus sans perdre d'information ?

Solution

La bonne exécution dépend des valeurs des registres. C'est le *contexte* d'exécution. Il faut donc le sauvegarder.

Si P_1 succède à P_2

- Sauvegarde du contexte de P_1
- Sélection de P_2
- Restauration du contexte de P_2
- Lancement de l'exécution de P_2

Commutation de tâches

Problème

Comment interrompre, puis reprendre, un processus sans perdre d'information ?

Solution

La bonne exécution dépend des valeurs des registres. C'est le *contexte* d'exécution. Il faut donc le sauvegarder.

Si P_1 succède à P_2

- Sauvegarde du contexte de P_1
- Sélection de P_2
- Restauration du contexte de P_2
- Lancement de l'exécution de P_2

Commutation de tâches

Problème

Comment interrompre, puis reprendre, un processus sans perdre d'information ?

Solution

La bonne exécution dépend des valeurs des registres. C'est le *contexte* d'exécution. Il faut donc le sauvegarder.

Si P_1 succède à P_2

- Sauvegarde du contexte de P_1
- Sélection de P_2
- Restauration du contexte de P_2
- Lancement de l'exécution de P_2

Commandes

Informations sur les processus sous Unix

ps Information instantannée sur les processus

Options :

- -a Tous les utilisateurs
- -u Tous les processus des utilisateurs spécifiés
- -x Même les processus non liés à un terminal

top Affichage continu d'informations sur les processus

Communication entre processus

Echange de données

- Par fichier
  - Ecritures et lectures simultannées par plusieurs processus
- Mémoire partagée
  - Les processus légers (threads) partagent la même zone mémoire
  - Différents processus peuvent partager une même zone mémoire

Synchronisation

Verrous Permettent de bloquer tout ou partie d'un fichier
Peuvent concerner la lecture et/ou l'écriture

Sémaphores Limitent l'exécution simultannée de plusieurs processus
Ne sont pas asociés à un type de ressource

Communication entre processus

Echange de données

- Par fichier
  - Ecritures et lectures simultannées par plusieurs processus
- Mémoire partagée
  - Les processus légers (threads) partagent la même zone mémoire
  - Différents processus peuvent partager une même zone mémoire

Synchronisation

Verrous Permettent de bloquer tout ou partie d'un fichier
Peuvent concerner la lecture et/ou l'écriture

Sémaphores Limitent l'exécution simultannée de plusieurs processus
Ne sont pas asociés à un type de ressource

Outils de communications

Echange de données et synchronisation

- Files d'attentes de messages
  - Un message déposé reste dans la boite jusqu'à ce que le destinataire vienne le chercher
- Les sockets
  - Communication à distance entre processus
  - Identification par numéro IP et numéro de port
  - Plusieurs protocoles supportés (TCP et UDP)
- Les tubes
  - Redirection de la sortie standard d'un processus vers l'entrée d'un autre

Signaux

Définition

La réception d'un signal interromp l'exécution d'un processus pour lancer l'exécution d'une routine de traitement

Concrètement

- Il existe plusieurs types de signaux correspondants à des situations différentes
  - Terminaison, erreurs ...
- Chaque signal a sa propre routine de traitement
- L'utilisateur peut définir ses propres routines de traitement
- Il est possible de masquer les signaux (à quelques excptions près)
- *kill* est la commande permettant d'envoyer un signal à un processus depuis un terminal

Signaux

Définition

La réception d'un signal interromp l'exécution d'un processus pour lancer l'exécution d'une routine de traitement

Concrètement

- Il existe plusieurs types de signaux correspondants à des situations différentes
  - Terminaison, erreurs ...
- Chaque signal a sa propre routine de traitement
- L'utilisateur peut définir ses propres routines de traitement
- Il est possible de masquer les signaux (à quelques excptions près)
- *kill* est la commande permettant d'envoyer un signal à un processus depuis un terminal

Modes d'exécution

Avant-plan

- Dans une fenêtre "Invite de commande" uniquement
- Monopolise l'invite de commande
  - L'utilisateur doit attendre la terminaison de la commande pour en lancer une autre
- Dans l'intervalle le processus peut afficher des messages et lire des données sur le clavier

Arrière-plan

- Processus lancé depuis une fenêtre "Invite de commande"
  - Exécution pendant que l'utilisateur peut lancer d'autres processus
  - Utile pour lancer un éditeur de texte par exemple
- Les daemons sont des processus systèmes d'arrière-plan

Modes d'exécution

Avant-plan

- Dans une fenêtre "Invite de commande" uniquement
- Monopolise l'invite de commande
  - L'utilisateur doit attendre la terminaison de la commande pour en lancer une autre
- Dans l'intervalle le processus peut afficher des messages et lire des données sur le clavier

Arrière-plan

- Processus lancé depuis une fenêtre "Invite de commande"
  - Exécution pendant que l'utilisateur peut lancer d'autres processus
  - Utile pour lancer un éditeur de texte par exemple
- Les daemons sont des processus systèmes d'arrière-plan

Syntaxe

Processus d'avant-plan

Il suffit de lancer la commande
ls, gedit ...

Processus d'arrière-plan

Il suffit de faire suivre la commande par le caractère "&"
gedit &

Transformer un processus d'avant-plan en processus d'arrière-plan

Utilisation de la commande *bg*. Voir aussi *fg*. Par exemple :

```
gedit      # Lancement de l'éditeur de texte en avant-plan
CTRL Z     # Interruption du processus
bg         # Reprise du processus en arrière-plan
```

Syntaxe

Processus d'avant-plan

Il suffit de lancer la commande
ls, gedit ...

Processus d'arrière-plan

Il suffit de faire suivre la commande par le caractère "&"
gedit &

Transformer un processus d'avant-plan en processus d'arrière-plan

Utilisation de la commande *bg*. Voir aussi *fg*. Par exemple :

```
gedit      # Lancement de l'éditeur de texte en avant-plan
CTRL Z     # Interruption du processus
bg         # Reprise du processus en arrière-plan
```

Syntaxe

Processus d'avant-plan

Il suffit de lancer la commande
ls, gedit ...

Processus d'arrière-plan

Il suffit de faire suivre la commande par le caractère "&"
gedit &

Transformer un processus d'avant-plan en processus d'arrière-plan

Utilisation de la commande *bg*. Voir aussi *fg*. Par exemple :

```
gedit     # Lancement de l'éditeur de texte en avant-plan
CTRL Z    # Interruption du processus
bg        # Reprise du processus en arrière-plan
```