Systèmes d'exploitation

Cours no. 11

Jean-Sébastien Coron

Université du Luxembourg

November 22, 2009

Programmation en C sous UNIX

- Opérations sur les fichiers.
  - Les commandes `fwrite`, `fread`
- Manipulation de processus.
  - La commande `fork()`

Les fichiers

- Ouverture d'un fichier: création d'un flux:
  - `FILE *f;`
  - `f=fopen("nom.ext","r");` en lecture.
  - `f=fopen("nom.ext","w");` en écriture.
- Ecriture dans un fichier:
  - `fprintf(f,"bonjour");` écriture dans un fichier.
- Lecture dans un fichier:
  - `fscanf(f,"%d",&i);` lecture d'un entier dans un fichier.

Manipulation de fichier

- `int fputc(char c,FILE *f)`
  - écrit le caractère `c` dans le ficher `f`, et incrémente la position.
- `int fgetc(FILE *f)`
  - lit un caractère, et incrémente la position.
  - Si fin du fichier, on obtient `EOF`.
- `int feof(FILE *f)`
  - teste si on a atteint la fin du fichier (0 pour faux, non-nul pour vrai)
- `int fclose(FILE *f)`
  - ferme le fichier

La commande `fwrite()`

- La commande `fwrite()`:
  - Permet de stocker des données binaires dans un fichier.
  - `int fwrite(void *ptr, int size, int nbelem, FILE *f)`
  - Ecrit `nbelem` éléments de taille `size` contenus dans le tableau `ptr`.
  - Le nombre d'octets écrits est `size*nbelem`.
  - La fonction renvoie le nombre d'éléments stockés.
- La commande `fflush()`
  - `fprintf` et `fwrite` sont bufférisées.
  - `fflush()` permet de vider le buffer.

Exemple

```
/* exemple avec fwrite */
#include <stdio.h>

int main () {
  FILE *f;
  char buf[] = "Hello world";
  f = fopen ("monfich.txt" , "w");
  int n=sizeof(buf); // 12
  fwrite (buf , 1 , n , f);
  fclose (f);
  return 0;
}
```

La commande `fread()`

- La commande `fread()`:
  - Permet de lire des données binaires dans un fichier.
  - `int fread(void *ptr,int size, int nbelem, FILE *f)`
  - Lit `nbelem` éléments de taille `size` et les stocke dans le tableau `ptr`.
  - La fonction renvoie le nombre d'éléments lus.
  - Si la fin du fichier est atteinte, le nombre d'éléments lus peut être inférieur à `size*nbelem`.

Exemple

```
/* exemple avec fread */
#include <stdio.h>

int main () {
  FILE *f;
  char buf[256];
  f = fopen ("monfich.txt" , "r");
  int n=fread (buf , 1 , 256 , f);
  printf("%s\n",buf);
  printf("Nb de carac lus: %d\n",n);
  fclose (f);
  return 0;
}
```

Les processus

- Création d'un processus.
  - Lorsqu'on entre une commande, le shell lance un processus pour l'exécuter.
  - Le shell attend la fin du processus, puis attend la commande suivante.
- Arborescence de processus.
  - Chaque processus a un *père*, celui qui l'a lancé.
  - Le numéro du processus du père est le PPID.
  - Un processus père peut avoir plusieurs processus *fils*
  - Sous UNIX, un premier processus `init` est créé avec un PID de 1. Ancètre de tous les processus.

Création de processus

- Duplication d'un processus.
  - Seule façon de créer un processus à bas niveau.
  - Appel système `fork()`
  - Crée une copie complête du processus, fils de ce processus.
  - Le fils change de programme en appelant `exec()`.
  - La fonction `exec()` remplace le code et ses données par celui d'une nouvelle commande.

Exécution d'une commande

- Exécution d'une commande: `tar -zcvf file *`
  - Le shell se duplique avec `fork()`. On a alors deux processus shell identiques.
  - Le shell père attend la fin du processus fils avec `wait()`.
  - Le shell fils remplace son exécutable par celui de la commande `tar` avec `exec()`
  - La commande `tar` s'exécute. Lorsqu'elle se termine, le processus fils disparait.
  - Le shell père est réactivé et attend la commande suivante.

La primitive `int fork()`

- L'appel a `fork()` duplique le processus.
  - L'exécution continue dans les deux processus.
- Valeur retournée par `fork()`:
  - Dans le processus père, retourne le PID du fils.
  - Dans le processus fils, retourne 0.
  - Si le fork échoue, renvoie -1.
- `getpid()` et `getppid()`
  - `getpid()` retourne le PID du processus.
  - `getppid()` retourne le PID du père du processus.

Exemple avec fork()

```
#include <stdio.h>
#include <unistd.h>
int main()
{
  int pidfils=fork();
  if(pidfils!=0)
  {
    printf("Je suis le père.");
    printf("PID de mon fils:%d\n",pidfils);
  } else
  {
    printf("Je suis le fils.");
    printf("Mon pid est:%d\n",getpid());
  }
}
```

Exemple avec `fork()`

- Résultat:
  - ```
    $ forksimple
    Je suis le père.PID de mon fils:2864
    Je suis le fils.Mon PID est:2864
    $
    ```
- Peut aussi apparaître dans l'ordre inverse.
    ```