Système et réseaux (SR1)

Programmation système en C sous Unix.

Attention, en C :

- pas de booléen : 0 pour faux, 1 pour vrai
- l'adresse d'une variable est indiquée par & devant le nom de la variable (les chaînes et tableaux sont toujours manipulés par leur adresse de début)

1) Utilisation des fichiers et autres utilitaires

De nombreuses instructions renvoient -1 comme code erreur et utilisent la variable prédéfinie `errno` pour fournir un code d'erreur détaillé.

Quelques utilitaires

librairies :
```
#include <string.h> pour strcpy et strcat
#include <strings.h> pour bzero
#include <stdio.h> pour perror
#include <unistd.h> pour getpass,sleep, access
```

- les codes erreur contenus dans la variable `errno` peuvent être affichées « en clair » par la fonction `perror` (qui a comme unique paramètre un message). Cette fonction affiche le message passé en paramètre puis l'intitulé de l'erreur
- les zones buffer peuvent être réinitialisées avec les fonctions `memset`(buffer, 0, taille) ou `bzero`(buffer, taille)
- l'affectation de chaînes de caractères se fait avec la fonction `strcpy`(destination, source)
- la concaténation de chaînes de caractères se fait avec la fonction `strcat`(destination, source)
- la saisie de mots de passe se fait avec la fonction `getpass`(prompt), le résultat renvoyé est le mot de passe (type char*) ou NULL en cas d'erreur
- un pipe peut être créé entre une commande et un processus avec `popen`(commande, mode), le résultat renvoyé est un flux dans lequel le processus devra lire ou écrire. Le mode est `r` ou `w`, il indique ce que doit faire le processus (*fread* ou *fwrite* sur le pipe).
- pour placer en attente un processus, la fonction `sleep`(durée) peut être utilisée. La durée est donnée en secondes. Le code retour est 0 ou le nombre de secondes restant à attendre si l'instruction `sleep` a été interrompue
- la fonction `access`(chemin, mode) permet de tester vos droits d'accès sur un fichier ainsi que son existence. Le paramètre mode prend les valeurs `R_OK, W_OK, XR_OK, F_OK`. Le code retour est 0 si l'accès est autorisé
- sur les chaînes de caractères : `sprintf` plus efficace que `strcat, strcpy`

Les fichiers

Ils peuvent être décrits dans les programmes C par un **descripteur** entier (que vous retrouvez dans le répertoire `/proc/pid/fd` pour le processus qui exécute votre programme) ou par un **flux** (variable de type FILE * dont la valeur est NULL en cas d'erreur). Un flux est traité au niveau système par un descripteur :

```
descripteur pour le fichier parDescripteur 4
commande ls -l /proc/2385/fd
total 0
lrwx------ 1 mnt mnt 64  1 oct.  17:55 0 -> /dev/pts/0
lrwx------ 1 mnt mnt 64  1 oct.  17:55 1 -> /dev/pts/0
lrwx------ 1 mnt mnt 64  1 oct.  17:55 2 -> /dev/pts/0
l-wx------ 1 mnt mnt 64  1 oct.  17:55 3 -> /home/mnt/Documents/cours-SR1/pgC/parFlux
l-wx------ 1 mnt mnt 64  1 oct.  17:55 4 -> /home/mnt/Documents/cours-SR1/pgC/parDescripteur
```

Sur les flux, la fonction `ferror(flux)` affiche le message associé au code erreur (s'il est non nul) de la dernière opération effectuée. La fonction `fileno(flux)` affiche le descripteur associé à un flux.

Les fichiers manipulés par descripteur permettent un déplacement direct à une position relative dans le fichier (vocabulaire *offset* et *base* qui peut être `SEEK_SET` (depuis le début du fichier), `SEEK_CUR` (depuis la position courante), `SEEK_END` (depuis la fin de fichier).

Les fichiers par flux

librairies :
```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

`fread`
paramètres : buffer, taille, nombre, flux
code retour : nombre d'élements lus, en cas d'erreur il faut utiliser `feof()` et `ferror()`

`fwrite`
paramètres : buffer, taille, nombre, flux
code retour : nombre d'élements écrits, en cas d'erreur il faut utiliser `ferror()`

`fopen`
paramètres : nom du fichier et mode d'utilisation du fichier (`r, r+, w, w+, a, a+`)
code retour : pointeur de flux, en cas d'erreur `NULL` et variable `errno` positionnée

`fclose`
paramètres : flux

code retour : en cas d'erreur la variable `errno` est positionnée

`feof`

paramètres : flux
code retour : vrai ou faux (sous la forme 0 ou non nul)

Pour lire avec un flux, ligne par ligne dans un fichier texte, vous pouvez utiliser `fgets`

*char *fgets(char *s, int size, FILE *stream);*

Les fichiers par descripteurs

librairie :
```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

`read`

paramètres : descripteur, le buffer et le nombre d'octets à lire (il est possible d'utiliser la fonction `sizeof`())
code retour : nombre d'octets lus, 0 en fin de fichier

`write`

paramètres : descripteur, le buffer et le nombre d'octets à écrire
code retour : nombre d'octets écrits, en cas d'erreur -1 et variable `errno` positionnée

`open`

paramètres : nom du fichier et drapeaux ainsi que les droits pour la création (un drapeau parmi `O_RDONLY`, `O_WRONLY`, `O_RDWR`, combinables par | avec `O_APPEND`, `O_CREAT` ; les droits sont optionnels et en notation octale sur 4 chiffres)
code retour : le numéro du descripteur, en cas d'erreur -1 et variable `errno` positionnée

`close`

paramètres : descripteur
code retour : 0 en cas de succès, en cas d'erreur -1 et variable `errno` positionnée

`lseek`

paramètres : descripteur, offset en octets, origine. Pour l'origine `SEEK_SET`, `SEEK_CUR`, `SEEK_END`
code retour : position en octets depuis le début de fichier, en cas d'erreur -1 et variable `errno` positionnée

2) Le mode non bloquant

Le passage en mode non bloquant peut se faire en utilisant une des requêtes de la fonction de paramétrage des descripteurs de fichiers ouverts (`fcntl`) ou une fonction plus sophistiquée de surveillance de plusieurs descripteurs, appelée `select`.

La fonction `fcntl`(descripteur, requête, drapeau) permet de rendre non bloquantes les entrées-sorties sur le fichier dont le descripteur est donné. La requête à utiliser est `F_SETFL`, le drapeau `O_NONBLOCK`. Pour revenir en mode bloquant, utiliser le drapeau inverse (`~O_NONBLOCK`). Le code retour est 0 ou -1 en cas d'erreur (avec la variable `errno` positionnée).

```
drapeau=fcntl(fd, F_GETFL);
fcntl(fd, F_SETFL, drapeau | O_NONBLOCK);
```

```
drapeau=fcntl(fd, F_GETFL);
fcntl(fd, F_SETFL, drapeau & ~O_NONBLOCK);
```

L'utilisation de `select` est plus complexe :

- il faut définir une table de descripteurs à surveiller :
  - type prédéfini `fd_set`
  - `FD_ZERO`(adresse de la table)
  - `FD_SET`(descripteur, adresse de la table)
  - `FD_CLR`(descripteur, adresse de la table)
  - `FD_ISSET`(descripteur, adresse de la table), le résultat est un entier (0 pour faux, 1 pour vrai)

  cette table doit être reconstruite après chaque appel à `select`
- il faut définir la temporisation :
  - type prédéfini `struct timeval`
  - champ `tv_sec` pour des secondes
  - champ `tv_usec` pour des micro-secondes
- instruction proprement dite :

  `select(nombre, tableLec, tableEcr, tableExcep, temporisation)`

  Le paramètre *nombre* détermine la surveillance des entrées d'indices *0* à *nombre-1* dans les tables fournies en paramètres. Si le code retour est 0, une des voies surveillée a été activée.

3) Les verrous

Les verrous mis en place avec `fcntl` sont consultatifs[1] (ou « *advisory* », ils ne protègent que des accès par des processus

1 *Les verrous impératifs ne figurent pas dans la norme POSIX (sous Linux, il faut utiliser l'option « -o mand » au montage du disque pour disposer de verrous impératifs).*

utilisant eux aussi des verrous). Il faut utiliser :

- une structure `struct flock` avec :
  - la définition du type de verrou
    champ `l_type` qui prend les valeurs `F_RDLCK, F_WRLCK, F_UNLCK`
    `F_RDLCK` est un verrou partagé et `F_WRLCK` un verrou exclusif
  - la définition de la zone sous verrou
    champ `l_whence` : comment comprendre la position de début (`SEEK_SET, SEEK_CUR, SEEK_END`)
    champ `l_start` : la position de début
    champ `l_len` : longueur (en octets)
  - le pid du processus, champ `l_pid`
- des opérations de manipulation
  `F_SETLK (struct flock *)` permet d'acquérir ou de libérer un verrou (code erreur -1 et errno positionnée)
  `F_SETLKW (struct flock *)` idem mais en attendant si un verrou existe déjà (code erreur -1 et errno positionnée)
  `F_GETLK (struct flock *)` renvoie `F_UNLOCK` si le fichier peut être verrouillé, ou les informations connues sur le verrou existant (`l_type, l_whence, l_start, l_len, l_pid`)

Exemple :

```
fic = open(fichier, O_RDWR);
lck.l_type = mode;              // F_RDLCK ou F_WRLCK
lck.l_whence = SEEK_SET;
lck.l_start = 200;
lck.l_len = 100;
res=fcntl(fic, F_GETLK, &lck);
                                // arrêt si res==-1
switch (lck.l_type)
  {
    case F_RDLCK : printf("verrou partagé posé par %d\n", lck.l_pid);
                   printf("entre %d et %d\n\n", lck.l_start, lck.l_start + lck.l_len);
                   break;
    case F_WRLCK : printf("verrou exclusif posé par %d\n", lck.l_pid);
                   printf("entre %d et %d\n\n", lck.l_start, lck.l_start + lck.l_len);
                   break;
    case F_UNLCK : // on peut poser le verrou
                   lck.l_type = mode;               // F_RDLCK ou F_WRLCK
                   res=fcntl(fic, F_SETLK, &lck);
                                                    // arrêt si res==-1
  }
```

4) Les signaux

La fonction `kill(pid, signal)` permet d'envoyer un signal depuis un programme C. Le code erreur est -1 avec la variable `errno` positionnée.

La fonction `sigaction(signal, nouveau, ancien)` permet de remplacer l'ancien comportement (handler) associé au signal par un nouveau. L'ancien comportement sera sauvegardé dans `ancien`, sauf si ce paramètre est à `NULL`. Pour la description des comportements, une structure prédéfinie est utilisée :

```
struct sigaction
{
  void       (*sa_handler)();
  sigset_t   sa_mask;
  int        sa_flags;
  ...
```

La fonction associée à `nouveau->sa_handler` est le comportement mis en œuvre pour le `signal`. L'ensemble de signaux `nouveau->sa_mask` regroupe les signaux qui doivent être bloqués pendant l'exécution de `nouveau->sa_handler`. Les drapeaux permettent de configurer le comportement du handler. Un seul drapeau étant reconnu par POSIX, nous ne les utiliserons pas (`nouveau->sa_flags=0`).
Le type prédéfini `sigset_t` permet de définir un ensemble des signaux à l'aide de primitives : `sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember()`. Nous utiliserons `sigfillset(ensemble)` qui collecte tous les signaux système et renvoie -1 en cas d'erreur.

