UVF2B303 - Programmation système et réseaux
Année scolaire: 2007-2008
Session: Février 2008
Partie Unix/Linux

Nom : **Prénom :**

◯ 3A SLR
◯ TW3S filière brestoise en 1 an
◯ TW3S filière marocaine en 2 ans

21 février 2008

Modalitées

- Document de cours autorisés (polys, sujets de TP, listings de TP)
- Réponses à rendre sur copie séparée.
- Durrée :
  - 3A SLR : 40mn ; répondre à la première question (la seconde question est optionnelle),
  - TW3S en 1 an : 40mn ; répondre à première question (la seconde question est optionnelle),
  - TW3S en 2 ans : 60mn ; répondre aux deux questions.

1 Première question : réaliser `socket_exec()`

On souhaite implémenter une fonction, appelée `socket_exec()` qui lance l'exécution d'une commande en parallèle, et qui renvoie un descripteur de socket Unix de type flux d'octets correspondant à l'entrée standard et à la sortie standard de la commande.

La commande est interprétée par le shell `/bin/sh` avec l'option `-c`. Par exemple si la commande souhaitée est `"ls -l /etc"` , on ferait `/bin/sh -c "ls -l /etc"`.

Note : pour les connaisseurs, cela ressemble à la fonction `popen()` survolée en cours, à la différence que l'on travail avec une socket et non un pipe.

L'annexe A documente à la façon du man ce que serait cette fonction.

L'annexe B donne la trame d'un programme qui utiliserait cette fonction : grâce à `socket_exec()` on appelle la commande `wc` (*Word Count* : compter les octets, les mots, les lignes d'un flux) sur le messages `MSG`, puis on affiche le résultat.

Les annexes C D E donnent quelques pages de man qui peuvent (ou non) être intéressantes pour répondre au problème.

1. Proposez une implémentation de la fonction `socket_exec()`.
2. Proposez votre propre implémentation de l'appel système `sockepair()`. (Voir annexe C, mais ne traitez pas le `errno`)
3. Décrivez l'appel système `shutdown()` utilisé ligne 36 du code de l'annexe B (son utilité dans le cas général et dans le cas de notre exemple).

2 Seconde question : Linux et Unix sont démoniaques !

Les services qui fonctionnent sous Linux (et sur tout système Unix en général) sont en fait des processus lancés en «tâches de fond» lors de l'initialisation du système, juste après le «boot». Ces processus sont souvent un peu particuliers car ils fonctionnent comme des processus lancés en arrière plan sur un terminal (lorsque la ligne de commande est terminée par un caractère «&»). Ce sont des «démons» (daemon process). Leur nom est généralement terminé par un «d» : `cupsd`, `inetd`, `sshd`, etc.

Pour créer un démon il faut que le processus devienne «process group leader» (nous avons survolé ce thème en cours et il n'est pas utile ici d'en savoir beaucoup plus). Pour cela, le processus lancé en premier (celui qui résulte du lancement de la commande) doit créer un fils et se terminer aussitôt. Si rien n'est fait dans le fils à cet instant alors le fils se termine aussi car son père s'est terminé. Pour contrer cet effet, le fils devient «process-goup leader» en appelant l'appel système `setsid()`[1].

1. Donner le code mettant en œuvre le mécanisme de «démonisation» d'un exécutable, en respectant la syntaxe la plus exacte possible.
2. Le processus fils devient orphelin car son père s'est terminé, or il n'y a pas de processus orphelins sous Unix. Expliquez ce qui se passe.

[1] Voir annexe F

Annexes

A SOCKET_EXEC(3) Manuel du programmeur SLR-TW3S

NOM

`socket_exec` - Entrées-sorties pour un processus

A.1 SYNOPSIS

```
int socket_exec(const char *commande);
```

DESCRIPTION

La fonction `socket_exec()` engendre un processus en créant deux sockets Unix connectées, exécutant un fork(), et en invoquant le shell.

L'argument `commande` est un pointeur sur une chaîne de caractères, terminée par un caractère nul, et contenant une ligne de commande shell. Cette commande est transmise à `/bin/sh` en utilisant l'option `-c`. L'interprétation en est laissée au shell.

La valeur renvoyée par `socket_exec()` est un flux d'octets d'entrée-sortie normal (en fait, un descripteur de socket Unix). L'écriture dans le flux correspond à écrire sur l'entrée standard de la commande. Symétriquement, la lecture depuis un flux ouvert par `socket_exec()` correspond à lire la sortie standard de la commande.

VALEUR RENVOYÉE

La fonction `socket_exec()` renvoie un descripteur de socket Unix ou -1 en cas d'échec quelconque.

B Exemple de programme utilisant `socket_exec()`

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <sys/wait.h>
4  #include <stdio.h>
5  #include <unistd.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9
10 int socket_exec(const char *commande) {
11   /* Creation de deux sockets UNIX connectees */
12
13   /* Creation d'un processus fils
14    * associe son entree standard et sa sortie standard a la premiere socket
15    * puis invoque un shell qui va interpreter la ligne de commande.
16    * Ce shell est "/bin/sh" avec l'option "-c".
17    */
18
19   /* Le processus pere retourne la seconde socket */
20
21   /* En cas d'erreur quelconque, retourne -1 */
22 }
23
24
25 #define MSG     "coucou\ntout le monde\n"
26
27 int main() {
28   char buff[200];
29   ssize_t sz;
30
31   int se = socket_exec("wc");
32
33   strcpy(buff, MSG);
34   write(se, buff, strlen(MSG)+1);
35
36   shutdown(se, SHUT_WR);
37
38   sz = read(se, buff, 200);
39   buff[sz]='\0';
40   printf("Resultat %s", buff);
41
42   wait(0);
43   exit(EXIT_SUCCESS);
44 }
```

NOM

`socketpair` - Créer une paire de sockets connectées.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int d, int type, int protocol, int sv[2]);
```

DESCRIPTION

La fonction `socketpair()` crée une paire de sockets connectées, sans noms, dans le domaine de communication d, du type indiqué, en utilisant le protocole optionnel protocol. Les descripteurs correspondant aux deux sockets sont placés dans `sv[0]` et `sv[1]`.

Les deux sockets ne sont pas différenciables.

VALEUR RENVOYÉE

En cas de réussite, zéro est renvoyé, sinon -1 est renvoyé et errno contient le code d'erreur.

NOTES

Sous Linux, le seul domaine supportant cet appel est PF_UNIX (ou le synonyme, PF_LOCAL). (La plupart des implémentations ont la même restriction).

D SOCKET(2) Manuel du programmeur Linux

NOM

`socket` - Créer un point de communication

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION

`socket()` crée un point de communication, et renvoie un descripteur.

Le paramètre domain indique le domaine de communication pour le dialogue ; ceci sélectionne la famille de protocole à employer. Elles sont définies dans le fichier `<sys/socket.h>`. Les formats actuellement proposés sont :

Nom	Utilisation	Page
PF_UNIX, PF_LOCAL	Communication locale	unix(7)
PF_INET	Protocoles Internet IPv4	ip(7)
PF_INET6	Protocoles Internet IPv6	ipv6(7)
PF_IPX	IPX - Protocoles Novell	
PF_NETLINK	Interface utilisateur noyau	netlink(7)
PF_X25	Protocole ITU-T X.25 / ISO-8208	x25(7)
PF_AX25	Protocole AX.25 radio amateur	
PF_ATMPVC	Accès direct ATM PVCs	
PF_APPLETALK	Appletalk	ddp(7)
PF_PACKET	Interface paquet bas-niveau	packet(7)

La socket a le type indiqué, ce qui fixe la sémantique des communications. Les types définis actuellement sont :

SOCK_STREAM Support de dialogue garantissant l'intégrité, fournissant un flux de données binaires, et intégrant un mécanisme pour les transmissions de données hors-bande.

SOCK_DGRAM Transmissions sans connexion, non garantie, de datagrammes de longueur maximale fixe.

SOCK_SEQPACKET Dialogue garantissant l'intégrité, pour le transport de datagrammes de longueur fixe. Le lecteur doit lire le paquet de données complet à chaque appel système read.

SOCK_RAW Accès direct aux données réseau.

Le protocole à utiliser sur la socket est indiqué par l'argument protocol. Normalement, il n'y a qu'un seul protocole par type de socket pour une famille donnée, auquel cas l'argument protocol peut être nul.

VALEUR RENVOYÉE

`socket()` renvoie un descripteur référençant la socket créée en cas de réussite. En cas d'échec -1 est renvoyé, et errno contient le code d'erreur.

E EXEC(3) Manuel du programmeur Linux

NOM

`execl, execlp, execle, execv, execvp` - Exécuter un fichier

SYNOPSIS

```
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

DESCRIPTION

La famille des fonctions `exec()` remplace l'image du processus en cours par une nouvelle image du processus.

L'argument initial de toutes ces fonctions est le chemin d'accès du fichier à exécuter.

Les arguments `const char *arg` ainsi que les points de suspension des fonctions `execl()`, `execlp()`, et `execle()` peuvent être vues comme `arg0`, `arg1`, ..., `argn`. Ensemble, ils décrivent une liste d'un ou plusieurs pointeurs sur des chaînes de caractères terminées par des caractères nuls, qui constituent les arguments disponibles pour le programme à exécuter. Par convention, le premier argument doit pointer sur le nom du fichier associé au programme à exécuter. La liste des arguments doit se terminer par un pointeur `NULL`, et puisque ce sont des fonctions variadiques, ce pointeur doit être transtypé avec `(char *) NULL`.

Les fonctions `execv()` et `execvp()` utilisent un tableau de pointeurs sur des chaînes de caractères terminées par des caractères nuls, qui constituent les arguments disponibles pour le programme à exécuter. Par convention, le premier argument doit pointer sur le nom du fichier associé au programme à exécuter. Le tableau de pointeurs doit se terminer par un pointeur `NULL`.

VALEUR RENVOYÉE

Si l'une des fonctions `exec` revient, c'est qu'une erreur a eu lieu. La valeur de retour est -1, et errno contient le code d'erreur.

F SETSID(2) Manuel du programmeur Linux

F.1 NOM

`setsid` - Créer une session et fixer l'ID du groupe de processus.

F.2 SYNOPSIS

```
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

`setsid()` crée une nouvelle session si le processus appelant n'est pas un leader de groupe. Le processus appelant devient le leader du nouveau groupe, et n'a pas de terminal de contrôle. L'ID du groupe de processus et l'ID de session du processus appelant sont fixés à la valeur de PID du processus en cours. Le processus en cours sera le seul dans son groupe et sa session.

VALEUR RENVOYÉE

L'identifiant de session du processus en cours.

ERREURS

En cas d'erreur, la valeur de retour est -1, et errno est positionné. La seule erreur susceptible de se produire est EPERM. Elle est déclenchée si l'ID du groupe de processus d'un processus quelconque est égal au PID du processus appelant. En particulier `setsid()` échoue si le processus appelant est déjà leader d'un groupe.

NOTES

Un fils créé par `fork(2)` hérite de l'identifiant de session de son père. L'identifiant de session est conservé après un `execve(2)`.

Le leader d'un groupe est le processus dont le PID est égal à l'ID du groupe. Pour s'assurer que `setsid()` réussira, il faut effectuer un `fork(2)`, suivi d'un `exit(2)` pour le père, et le fils appellera `setsid()`.