UE Programmation Unix

Contrôle Continu 2019

1 Partie 1
CONNAISSANCE DU COURS

Question 1 Vrai / Faux — 3 points

- **Le système gère une table `u_ofile` par processus**

Vrai

- **`u_ofile`:** table des descripteurs de fichiers d'un processus
  - Chaque processus possède la sienne
    - Vision PAR PROCESSUS des fichiers ouverts

- **Il n'est pas possible d'utiliser une socket avant de l'avoir nommée.**

Faut

Apres l'appel à la primitive socket, je me retrouve avec une socket qui a été créé mais quant à l'utilisation de cette socket, appart des processus de la même filiation, personne ne peut utiliser cette socket.

Création d'une socket

- La primitive `socket()`

```
#include<sys/types.h>
#include<sys/socket.h>
     int socket(int domain, int type, int protocol);
```

– à ce niveau, aucun processus d'une autre filiation ne peut atteindre la socket, il faut lui donner un nom.

- **La primitive `open` a pour unique fonction d'ouvrir un fichier.**

Faut

Autres fonctions de la primitive open :

Available Values for `oflag`

Value	Meaning
O_RDONLY	Open the file so that it is read only.
O_WRONLY	Open the file so that it is write only.
O_RDWR	Open the file so that it can be read from and written to.
O_APPEND	Append new information to the end of the file.
O_TRUNC	Initially clear all data from the file.
O_CREAT	If the file does not exist, create it. If the O_CREAT option is used, then you must include the third parameter.
O_EXCL	Combined with the O_CREAT option, it ensures that the caller *must* create the file. If the file already exists, the call will fail.

Question 2 Questions de cours 2 points

- **Soient deux fichiers appartenant à Francois du groupe users tels que décrits ci-dessous. `a.out` est un programme qui ouvre en lecture/écriture (`O_RDWR`) le fichier `Donnees`.**

```
-rwsr-xr-x    1      francois     useres 11687  déc    2      14:12  a.out
----rw-r--    1      francois     useres    44  déc    2      14:09  Donnees
```

Francois et Pierre sont du même groupe. Le programme `a.out` peut-il ouvrir le fichier `Donnees` s'il est exécuté par :

a. **Francois ? Non car les droit de Francois pour `Donnees` sont `---`**

b. **Pierre ? Non car même si les droit de Pierre pour `Donnees` sont `rw-` ,le droit SUID est ajouter à l'exectuable `a.out` et les droit du propriétaire (Francois) sont `---`**

Le mode d'accès (O_RDWR) ne correspond pas aux droits d'accès du propriétaire.

Utilisation [modifier | modifier le code]

Pour voir quels droits sont attribués à un fichier, il suffit de taper la commande ls -l nom_du_fichier :

```
# ls -l toto
-rwxr-xr--    1 user     group     12345 Nov 15 09:19 toto
```

La sortie signifie que le fichier toto (de taille 12345) appartient à « user », qu'on lui a attribué le groupe « group », et que les droits sont rwxr-xr--. On remarque qu'il y a en fait 10 caractères sur la zone de droits. Le premier - n'est pas un droit, c'est un caractère réservé pour indiquer le type de fichier. Il peut prendre les valeurs suivantes :

- `d` : répertoire
- `l` : lien symbolique
- `c` : périphérique de type caractère
- `b` : périphérique de type bloc
- `p` : pipe (FIFO) pour "tube" ou "tuyau" en anglais ou pipeline aussi en 'français'.
- `s` : socket
- `-` : fichier classique

Droit SUID

Ce droit s'applique aux fichiers exécutables, il permet d'allouer temporairement à un utilisateur les droits du propriétaire du fichier, durant son exécution. En effet, lorsqu'un programme est exécuté par un utilisateur, les tâches qu'il accomplira seront restreintes par ses propres droits, qui s'appliquent donc au programme. Lorsque le droit SUID est appliqué à un exécutable et qu'un utilisateur quelconque l'exécute, le programme détiendra alors les droits du propriétaire du fichier durant son exécution. Bien sûr, un utilisateur ne peut jouir du droit SUID que s'il détient par ailleurs les droits d'exécution du programme.

Notation [modifier | modifier le code]

Son flag est la lettre s ou S qui vient remplacer le x du propriétaire. La majuscule ou la minuscule du 's' permet de connaitre l'état du flag x (droit d'exécution du propriétaire) qui est donc masqué par le droit SUID 's' ou 'S': C'est un s si le droit d'exécution du propriétaire est présent, ou un S sinon. Il se place donc comme ceci :

```
---s------  ou  ---S------
```

Un fichier avec les droits `-rwxr-xr-x` auquel on ajoute le droit SUID aura donc la notation `-rwsr-xr-x`

Question 3 Questions de cours 2 points

Citez les types de fichier gérés par les systèmes de la famille UNIX ?

Types de fichiers (cours p. 24 – 25)

Le type du fichier détermine les opérations possibles.
Le type de fichier est encodé dans le champs `st_mode` de la structure `stat`.

→**Fichier régulier ou ordinaire**

non structuré

contenu : texte, binaire, image, document, etc.

→**Répertoire**

nœud de l'arborescence

Contenu : fichiers réguliers, répertoires

→**FIFO (tube)**

communication unidirectionnelle entre processus d'une même machine.

→**Socket AF_UNIX**

→**Lien symbolique**

Contenu : un nom de fichier.

→**Fichier spécial**

- Périphérique

- **Mode bloc : disque**
- **Mode caractère : clavier, écran**

Que demande l'utilisateur lorsqu'il écrit

```
sd = socket(AF_INET, SOCK_STREAM, 0)
```

dans un programme ?

La primitive socket

Tous processus qui aura besoin de communiquer avec un processus distant qui se trouve sur une autre machine (machine qui est connecté bien sûr à un réseau qui permet de l'atteindre), va devoir créé une socket (socket c'est une prise en anglais, une prise a laquelle on se branche..) et donc la création d'une socket passe par une primitive du même nom qui permet de le faire.

- La primitive `socket()`

```
#include<sys/types.h>
#include<sys/socket.h>
     int socket(int domain, int type, int protocol);
```

  - retourne un descripteur de fichier,
  - si `protocol` = 0 → le système choisit le protocole

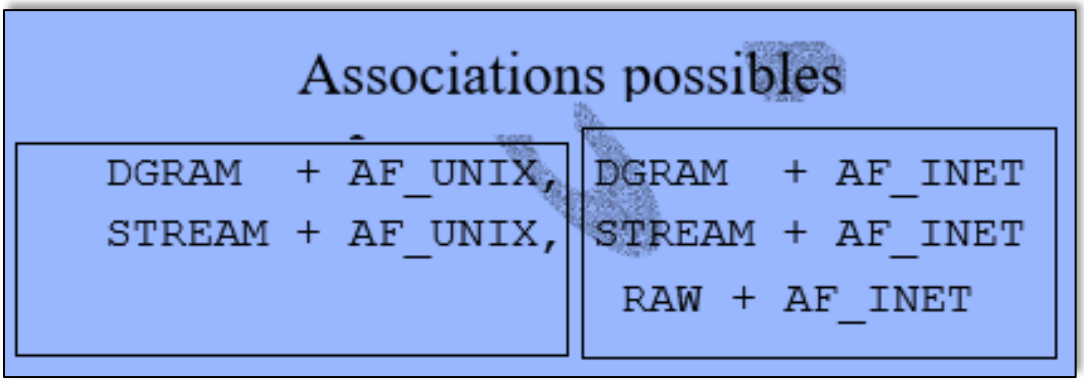
On a 3 paramètres :

1. **Domaine** - **le domaine ça va faire référence en fait au type d'adresse : adresses internet, adresses UNIX, adresse IPX (existe plus ajd)..**

Domain	Description
AF_INET	IPv4 Internet domain
AF_INET6	IPv6 Internet domain (optional in POSIX.1)
AF_UNIX	UNIX domain
AF_UNSPEC	unspecified

2. **Type** – **de quelle type de socket on a besoin ? ce Type il fait référence a la qualité de service qu'on a besoin pour notre application. En gros, ajd dans l'architecture TCP/IP y'a 2 type de service : le service connecté et le service non-connecté. A nous de voir laquelle des 2 correspond au besoin de notre application.**

Type	Description
SOCK_DGRAM	fixed-length, connectionless, unreliable messages
SOCK_RAW	datagram interface to IP (optional in POSIX.1)
SOCK_SEQPACKET	fixed-length, sequenced, reliable, connection-oriented messages
SOCK_STREAM	sequenced, reliable, bidirectional, connection-oriented byte streams



3. **Protocol** - **Le Protocol qui va être utiliser pour la communication entre les processus qui vont communiquer à travers les sockets. On peut mettre une valeur si on veut, mais si on mais 0 c'est le system qui choisit pour nous le Protocol qui correspond au type de service qu'on a demander (2em paramètre). En pratique on met toujours 0.**

Par exemple : si on mai SOCK_STREAM, le Protocol qui va être choisi automatiquement par le system ça va être TCP.

Quand je fais appel a la primitive socket je récupère un entier. Cette entier on l'appelle « un descripteur de socket ».

Quand on crée une socket, le descripteur qui va être retourner si tous va bien c'est un entier et cette entier a la même sémantique que l'entier qui est retourner par un `open()`. c'est le numéro de l'entrée de la `u_ofile` du processus qui a été utiliser pour la création de cette socket.

2 Partie 2
APPLICATION DU COURS

Question en plus sur les tubes – TP 3.12

Question pas corrigé en TP – peut être c une question du CC de mardi ??

Ecrire un programme qui CALCULE et affiche la capacité d'un tube ordinaire* en nombre de caractères.

*M.Soto : « mais ça change pas le problème a mon avis si c'était un tube nommée » .

On insiste sur le mot « calcule » car y'a une constante dans le système qui permet de connaitre la taille d'un tube. Cette constante est mentionnée dans le cours. Donc, pour cette exercice FAUT PAS afficher la valeur de cette constante. Cette valeur doit être calculer.

Tube ordinaire – le tube qui fonctionne que par héritage.

Indice : Vous n'arriverai pas sans les signaux

CC 2019 – 3 points

Sachant que le temps d'écriture d'un octet dans un tube ordinaire est inférieur à 1 seconde, écrire un programme qui calcule et affiche la capacité maximale d'un tube ordinaire puis se termine

```c
#include <stdio.h> // printf()
#include <stdlib.h> // exit(), EXIT_SUCCESS
#include <signal.h> // sigaction(), sigemptyset(), alarm()
#include <unistd.h> // pipe(), write()

long count = 0;
void handler(int signo);

int main(int argc, char *argv[]) {
    //Structure pour la mise en place des gestionnaires
    struct sigaction action;

    /* Remplissage de la structure */

    // Adresse du gestionnaire
    action.sa_handler = handler;

    // Mise a zero du champ sa_flags théoriquement ignoré
    action.sa_flags = 0;

    /* int sigemptyset(sigset_t *set);
     * initialise a VIDE l'ensemble de signaux pointé par set
     * retourne 0 en cas de succès ou -1 en cas d'erreur
     */

    // On ne bloque pas de signaux spécifiques
    sigemptyset(&action.sa_mask);

    /* int sigaction(int sigo, const struct sigaction *act,
       struct sigaction *oldact); */
    // Mise en place du gestionnaire pour le signal SIGALRM
    sigaction(SIGALRM, &action, NULL);

    int fd[2];
    pipe(fd);
    alarm(1);
    while(1){
        if( write(fd[1], "i", 1) >= 0 ) {
           count++;
           alarm(1);
       }
    }

    return 0;
}

void handler(int signo) {
    printf("Capacite maximale du tube : %ld\n", count);
    exit(EXIT_SUCCESS);
}
```

CC 2019 – 4 points

Ecrire un programme `auto_exec.c` qui se recouvre N fois lui-même.
A chaque recouvrement le programme affiche :

```
./auto_exec : recouvrement i.
```

Avant de se terminer, il affiche :

```
./auto_exec : terminé
```

Par exemple : `./auto_exec 3` aura pour résultat :

```
./auto_exec recouvrement 1.
./auto_exec recouvrement 2.
./auto_exec recouvrement 3.
./auto_exec : terminé
```

```c
#include <stdio.h> // fprintf(), printf(), sprint()
#include <stdlib.h> // atoi(), exit(), EXIT_FAILURE
#include <unistd.h> // execl()

int main(int argc, char *argv[]) {
    if(argc < 2){
       fprintf(stderr, "Usage : %s N \n", argv[0]);
       exit(EXIT_FAILURE);
    }

    if( argc == 2 )
    {
       execl(argv[0], argv[0], argv[1], "1", NULL);
    }
    else
    {
       int N = atoi( argv[1] );
       int i = atoi( argv[2] );

       printf("%s recouvrement %d.\n", argv[0], i);

       if( i != N)
       {
          char next_i[10];
          sprintf(next_i, "%d", i + 1);
          execl(argv[0], argv[0], argv[1], next_i, NULL);
       }
       else
       {
          printf("%s : terminé \n", argv[0]);
       }
    }

    return 0;
} /* main */
```

s