R3.05 Programmation système

#include

De base	Client	Serveur
#include <stdio.h> #include <unistd.h> #include <stdlib.h> #include <errno.h> #include <sys/wait.h> #define TAILLE 1 000 000	#include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> #include <netdb.h>	*Inclusion du client +* #include <signal.h> #include <sys.ioctl.h>

Inclusions annexes

#include <ctype.h> **(fonctions toupper())**
#include <pthread.h> **(Thread)**

printf :

%d → int
%c → char
%s → string (char*)
%lu → long unsigned int

TP1 Processus

Exercice 1

```
void exo1 (){
  printf("Je suis le processus numero %d\n", getpid()); // getpid --> permet d'obtenir le PID du processus en cours

  int fils; // variable contenant le PID du processus fils
  fils = fork();

  printf("Fork m'a renvoye la valeur : %d\n", fils);
  // A partir d'ici, les lignes sont exécutés par le processus fils créé par le fork

  printf("Je suis le processus numero %d %s %d\n ", getpid(), "et mon pere est le processus numero \n ", getppid()); // getppid --> permet d'obtenir le PID du pere
```

```
}
```

Exercice 2

```
pid_t id;

  int compteur; // Variable compteur déclaré une seule fois
                // Chaque processus "redéclare la sienne"

  //PROCESSUS
  id = fork();
  switch(id){
     case -1 :
        printf("Il y a une erreur\n");
        exit(EXIT_SUCCESS);
        break;
     case 0: //Processus fils
        for(compteur = 0; compteur <= 20; compteur +=2){
           printf("Les nombres pairs %d\n", compteur*2);
           sleep(1);
        }
        break;
     default: //Processus père
        for (compteur = 1; compteur <= 19; compteur += 2){
           printf("Les nombres impairs %d\n", compteur);
        }
        break;
  }
```

Exercice 3 & 4

```
  //VARIABLES
  pid_t id;
  int status;

  //PROCESSUS
  id = fork();

  switch(id){
     case -1 :
        printf("Il y a une erreur\n");
        exit(EXIT_FAILURE);
        break;
```

```
    case 0: //Processus fils
        printf("processus fils\n");
        //execlp("ps", "ps", NULL); // (relative path, commande, NULL)
        execl("/bin/ps", "ps", NULL); // execl sans p nécessite le chemin complet (path, commande, NULL)
        exit(EXIT_SUCCESS);
        break;
    default: //Processus père
        waitpid(id, &status,0);
        printf("La valeur de retour du fils est : %d\n", WEXITSTATUS(status));
}
```

Exercice 5

```
int main(int argc, char *argv[]){
  // argv c'est les parametres en console en gros

  if(argc < 2){ // si plus de 2 paramètres... (on choisit de l'interdire)
     printf("usage : %s commande\n", argv[0]);
     exit(EXIT_FAILURE);
  }
  //VARIABLES
  pid_t id;
  int status;

  //PROCESSUS
  id = fork();

  switch(id){
     case -1 :
        printf("Il y a une erreur\n");
        exit(EXIT_FAILURE);
        break;
     case 0: //Processus fils
        execlp(argv[1],argv[1], argv[2], argv[2],  NULL);
        // Ici on passe argv[1] : la commande
        // On passe argv[2] : les éventuels paramètres
        // On pourrait passer argv[3]...

        exit(EXIT_SUCCESS);
        break;
     default: //Processus père
        waitpid(id, &status,0);
```

```
        printf("La valeur de retour du fils est : %d\n", WEXITSTATUS(status));
    }
}
```

TP2 Threads

Exercice 2 & 3

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>
#include <pthread.h>

void thread1(){
    setbuf(stdout,NULL); // On débufeurise les sorties standards
    // Cela permet d'enlever \n par exemple.
    do{
        printf(".");
        sleep(1);
    }while(1);
    pthread_exit(NULL); //retourne null car osef de la valeur
}
void thread2(){
    char c;
    do{
    printf("Tapez un caractère : \n");
    c = getchar();
    }while(c != 'a');
    pthread_exit(NULL);
}

int main(){
    pthread_attr_t attr; //attributs de création

    pthread_t id1, id2; //identifiant des threads

    pthread_attr_init(&attr); //initialise la structure des threads
    pthread_create(&id1, &attr, (void*) thread1, NULL);
    pthread_create(&id2, &attr, (void*) thread2, NULL);
```

```
  //pthread_join(id1,NULL);
  pthread_join(id2,NULL); // Pthread_join permet d'attendre la fin d'exécution d'un thread

  printf("On va s'arrêter la\n");
  exit(EXIT_SUCCESS);
}
```

Rq : Ici on attend la fin du thread 2 (condition d'arrêt)
Si on attendait la fin du thread 1, le thread 2 s'arrêterait quand il remplirait la condition, mais le thread 1 continuerait de boucler indéfiniment et il ne s'arrêterait jamais (hors ctrl+c...)

Exercice 3

Meme code que exercice 2 + l'avant dernière ligne a modifier comme tel

```
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
  // Permet de lancer les threads dans l'état "détachés"
```

L'état détaché assure la libération des ressources instantanément a la fin de l'exécution d'un thread.
Cela empêche le pthread_join, donc tout se stop direct.

TP3 Mutex

Exercice 1

Exo de threads classiques
Pas de sémaphore, les threads écrivent en même temps dans une même variable, bref, ça marche pas.

Exercice 2

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>
#include <pthread.h>
```

```c
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // page 14 du manuel C

void thread(unsigned long *cpt){ // on récupère le contenu de la variable dans l'adresse de cpt
  // VARIABLES
  unsigned long tmp; // tampon du compteur
  unsigned long i;   // incrémentateur

  // TRAITEMENT
  tmp = 0;
  // compteur
  for(i = 0; i < 10000000; i++){
     pthread_mutex_lock(&mutex); // ON BLOQUE ICI LA VARIABLE
     tmp = *cpt; // *cpt : contenu de la variable à l'adresse de cpt
     tmp ++;
     *cpt = tmp;
     pthread_mutex_unlock(&mutex); // ICI ON DÉBLOQUE LA VARIABLE
  }
 pthread_mutex_destroy(&mutex); // Ici, comme on en a plus besoin, on détruit le mutex
  // afficher le contenu de cpt
  printf("Cpt = %lu\n", *cpt);
}


int main(){
  // VARIABLES
  pthread_attr_t attr; // attributs de création
  pthread_t id1, id2;  // identifiant des threads

  unsigned long cpt;   // variable compteur

  // TRAITEMENTS
  cpt = 0;

  pthread_attr_init(&attr);
  pthread_create(&id1, &attr, (void*) thread, &cpt); // On passe l'adresse de cpt dans le thread
  pthread_create(&id2, &attr, (void*) thread, &cpt);

  printf("Les 2 threads sont lancés\n");

  pthread_join(id1, NULL);
  pthread_join(id2, NULL);
```

```
  exit(EXIT_SUCCESS);
}
```

Exercice 3

Le temps d'exécution est plus long avec les sémaphores car chaque thread doit attendre son tour.

TP4 Pipes

Exercice 1

```
int main(){
  // VARIABLES
  pid_t id; // pid du fils
  int tube[2];   // tableau à 2 case représentant le tube
  char caractere;// caractère saisi au clavier par l'utilisateur

  // TRAITEMENTS
  setbuf(stdout, NULL); // On débufeurise les sorties standards (on est pas obligé)
  id = fork();

  if(pipe(tube)==-1){ //tentative ouverture du pipe
     printf("erreur\n");
     exit(EXIT_FAILURE);
  }

  switch(id){
     case -1 :
        printf("Il y a une erreur\n");
        exit(EXIT_SUCCESS);
        break;

     case 0: //Processus fils
        close(tube[1]); // fermeture du sens ecriture

        while((read(tube[0],&caractere, sizeof(int))!=-1)){
           // read renvoie -1 si erreur, dans ce cas on arrete la boucle
           if(caractere != '$'){
              printf("Caractère = %c\n", toupper(caractere));
           }
           else{
```

```
                break; // Fin de la boucle
            }
        }
        close(tube[0]); // fermeture du sens lecture
        break;

    default : //Processus père
        close(tube[0]); // Fermeture du sens lecture
        while((caractere = getchar()) != '$'){
            caractere = getchar(); // Lecture du caractere
            write(tube[1], &caractere, sizeof(int));
            if (caractere=='$'){
              break;
            }
        }

        close(tube[1]); // Fermeture du sens ecriture
        break;
  }
  exit(EXIT_SUCCESS);
}
```

Exercice 2

Modifications à ajouter :
Déclarer une variable “int état”.

Devant la fonction read, ajouter "état = read(...)”

```
if (etat = 0){
  break;
}
```

read renvoie 0 s’il n’y a plus de tube ouvert. Le père fermant le tube lorsqu’il reçoit $ ...

Exercice 3

Modifier la condition du père “char == ‘$’” par “char == EOF”
Ajouter printf(“fin de transfert”); après la fermeture du tube
Pour exécuter le programme avec une redirection, il faut utiliser la commande en console unix : ./main < fichier.txt

Le message “fin de transfert” apparaît à la fin du programme. (question du tp)

TP5 Signaux

Exercice 1

```
int main(){
  while(1){
     printf(".\n");
     sleep(1);
  }
  exit(0);
}
```

Exercice 2

```
#include <signal.h> // manipulations de signaux (il faudra inclure le reste aussi évidemment)
void rien(){
//ne fait rien, revient directement au programme principal
}

int main(){
  signal(SIGINT, rien);
  //Intercepte le signal donné en paramètre(1), exécute la fonction donné en paramètre(2)
  //Fonctionne en parallèle par nature, pas besoin de l'intégrer à un processus autre ou à un thread
  while(1){
     printf(".\n");
     sleep(1);
  }
  exit(0);
}
```

Exercice 3

```
void loupe(){
  printf("Loupé !\n");
  while(1){
     //vide
  }
}

int main(){
  signal(SIGTSTP, loupe);
  //Intercepte le signal donné en parametre(1), exécute la fonction donné en paramètre(2)
  //Fonctionne en parallèle par nature, pas besoin de l'intégrer à un processus autre ou à un thread
  while(1){
     printf(".\n");
     sleep(1);
  }
  exit(0);
}
```

Question TP :
Lorsqu'on tape Ctrl+Z la première fois, le programme affiche "Loupé !" puis n'affiche plus rien mais continue de tourner. Les autres fois il ne se passe rien. On peut le tuer avec un Ctrl+Z

Exercice 4

```
void fin(){
  // Important, il faut déclarer fin avant loupé, sinon le compilateur ne connaîtra pas fin dans la fonction loupé
  exit(EXIT_SUCCESS); // termine l'exécution du programme
}

void loupe(){
  signal(SIGTSTP, fin);
  printf("Loupé !\n");
  while(1){
  }
}
```

Même main que pour l'exercice 3

Exercice 5

```
int main(){

  pid_t id;
  int etat;

  id = fork();
  switch(id){
     case -1:
        printf("Erreur lors de la création du processus fils\n");
        exit(EXIT_FAILURE);
        break;
     case 0:
        //Configuration de la réception du signal
        sig_pere.sa_handler = signal_pere;
        sig_pere.sa_flags = SA_RESTART; // Non interruption des fonctions système (obligatoire)
        sigemptyset(&sig_pere.sa_mask); // masque de signaux
        sigaction(SIGUSR1, &sig_pere, NULL); // Mise en place de la fonction

        //CODE
           attente = 0;
           printf("Fils : En cours d'exécution...");
           while(attente ==0){
              sleep(1);
              printf("Fils : Dans la boucle...\n");
           }
        printf("Fils : fin de la boucle et retour\n");
        exit(EXIT_SUCCESS);
        break;
     default:
        printf("Pere : Attente des 5 secondes.\n");
        sleep(5);

        printf("Pere : Envoie du signal...\n");
        kill(id, SIGUSR1); // envoie du signal au fils

        //Attente de terminaison du processus fils pour éviter les zombies
        wait(&etat);
        exit(EXIT_SUCCESS);
        break;
  }
```

```
  exit(0);
}
```

TP6 Clients / Serveurs

Client

Rappel du sujet : se connecter au serveur lakartxela (localhost) sur le port 64100. Lui envoyer un chiffre, vérifier qu'il nous renvoie bien le carré de ce chiffre.

```
int main(){
  // VARIABLES

  // Déclaration de la socket locale depuis laquelle on se connectera au serveur
  int fd; // File descriptor (on déclare un socket)
  fd = socket(AF_INET, SOCK_STREAM, 0); // On initialise socket
  if (fd < 0){
     printf("Erreur lors de la création du socket\n");
     exit(EXIT_FAILURE);
  }

  // Paramétrage de l'adresse à laquelle on souhaite se connecter
  struct sockaddr_in adresse; // On déclare une structure d'adresse
  struct hostent *serveur; // Contiendra l'adresse du serveur distant
  serveur = gethostbyname("iparla.iutbayonne.univ-pau.fr");

  adresse.sin_family = AF_INET;
  adresse.sin_port = htons(64100); //htons convertie un entier en "port"

  adresse.sin_addr = *(struct in_addr *) serveur -> h_addr;
  bzero((char*)&adresse, sizeof(serveur));
  //serveur -> h_addr : un membre de la structure hostent.
  //Contient l'adresse IP du serveur convertie au format réseau et récupéré par la fonction
gethostbyname
  //On n'oubliera pas de l'initialiser avec la fonction bzero tel que déclaré

  if(connect(fd, (struct sockaddr *)&adresse, sizeof(adresse)) < 0){
     //connect renvoie une valeur inférieure à 0 en cas d'erreur
     printf("connexion impossible\n");
  }
  else{
     printf("Connexion établie\n");
```

```
    int msg = 4;

    //Envoi du message au serveur
    write(fd, &msg, sizeof(int));

    //Lecture du message reçu par le serveur sur le socket
    read(fd, &msg, sizeof(int));
    printf("Message reçu : %d\n", msg);

    // On oublie pas de fermer ce qu'on ouvre
    close(fd);
  }

   exit(EXIT_SUCCESS);
}
```

Serveur

Rappel du sujet : Créer un serveur qui reçoit une connexion, reçoit un caractère, le renvoie en majuscule via la fonction toupper().

```
int main(){

  int fd; // File descriptor (on déclare un socket)
  struct sockaddr_in serveur; // On déclare une structure d'adresse pour notre serveur


  // Création du socket
  fd = socket(AF_INET, SOCK_STREAM, 0);
  if (fd < 0){
     printf(("Erreur lors de la création du socket"));
     exit(EXIT_FAILURE);
  }

  // Paramétrage de l'adresse serveur (la notre)
  serveur.sin_family = AF_INET;
  serveur.sin_port = htons(9876);
  serveur.sin_addr.s_addr = INADDR_ANY; //On définit que tlm peut se connecter a notre serveur (any adresse in)

  // ici, on tente de lier le socket et le port
  if((bind(fd, (struct sockaddr *)&serveur, sizeof(struct sockaddr))) < 0){
     printf("Impossible de lier le socket et le port\n");
     exit(EXIT_FAILURE);
```

```
  }

  // On attend une connexion
  listen(fd, 5); // le 2eme parametre est le nombre d'utilisateur simultanés pouvant se connecter
  // remarqque: si c'est full et qu'un client veut se connecter, il y aura un probleme coté client, coté serveur c'est
bon

  // On peut maintenant dialoguer avec le client
  int fa, size;
  struct sockaddr_in client;
  size = sizeof(struct sockaddr); // La taille de la structure adresse client et serveur
  while(1){
     if((fa = accept(fd, (struct sockaddr*)&client, &size)) < 0){
        printf("Impossible d'accepter la socket distante\n");
        exit(EXIT_FAILURE);
     }
     else{
        char *message;
        read(fa, &message, sizeof(char));
        message = toupper(message);
        write(fa, &message, sizeof(char));
     }
  }

  //Fermeture du socket
  close(fd);
  exit(EXIT_SUCCESS);
}
```