ProgSys

Tube anonyme :
<unistd.h>

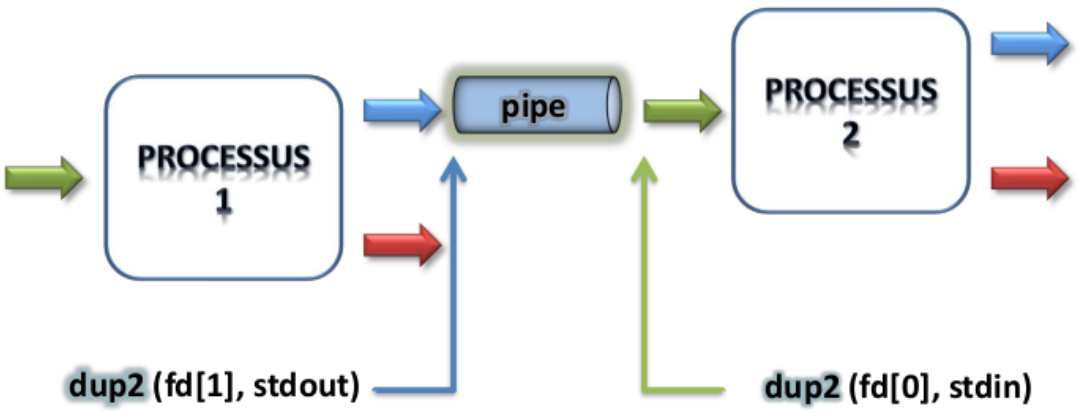
Code

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main()
 {char buffer[30]; int fd[2];
pipe( fd );
switch (fork( ))
 {
case -1 :      perror ("fork( )");          exit (-1);
case 0  :
                close (fd[1]);
                printf ( "Fils : Lecture tube \n");
                read (fd[0], buffer, sizeof (buffer));
                printf ( "Fils : %s \n " , buffer);
                break;
default :
                close (fd [0]);
                printf( "donner votre chaine de caractères" );
                scanf (" %s", buffer);
                printf ("Père : Écriture dans tube \n");
                write (fd [1], buffer, 20);
                wait (NULL);
                break;
}
return (0);}
```

```
int main ( )
{     int fd[2];
      pipe (fd);
            if (fork())
            {           if (fork()){
                              close fd[0];      close fd [1];
                              wait(NULL);    wait(NULL );

                        }else {
                              close (fd[1]);
                              dup2(fd[0], 0);
                              execlp("wc", "wc", "-c", NULL);

            }else {
                        close (fd[0]);
                        dup2(fd[1], 1);
                        execlp("ls", "ls", "-l", NULL);
            }
return 0;
}
```



Tube nommé
<sys/types.h>
<sys/stat.h>

Client.c

```
int main ( ) {
      char chaine1 [1024], chaine2 [1024];
      char * Question =  "/tmp/client";
      char * Reponse  =  "/tmp/serveur";

      int d_E_Clt, d_L_Clt;
      mkfifo (Question, 0644);

      d_E_Clt  = open (Question,  O_WRONLY);
      d_L_Clt  = open (Reponse,  O_RDONLY);

      printf("donner la question" );
      scanf (" %s", chaine1);
      write (d_E_Clt, chaine1, 30);

      read (d_L_Clt, chaine2, sizeof(chaine2));
      printf("la réponse est  : %s\n" , chaine2);

return 0; }
```

Serveur.c

```
int main ( )
{
      char chaine1 [1024], chaine2 [1024];
      char * Question =  "/tmp/client";
      char * Reponse  =  "/tmp/serveur";

      int d_E_Serv, d_L_Serv;
      mkfifo (Reponse, 0644);

      d_L_Serv = open (Question, O_RDONLY);
      d_E_Serv = open (Reponse, O_WRONLY);

      read (d_L_Serv, chaine1, sizeof(chaine1));
      printf("la question est  : %s\n" , chaine1);

      printf("donner la réponse  " );
      scanf (" %s", chaine2);
      write (d_E_Serv, chaine2, 30);

return  0; }
```

If you want to open in non block mode : open(fifoname, O_WRONLY | O_NONBLOCK)

Threads

Données communes entre les threads
Les ressources d'un processus sont partagées par tous ses threads :
• Code
• Variables globales
• Fichiers ouverts
• Signaux
• Droits Unix
• Environnement de shell
• Répertoire de travail
=> Contenu de la mémoire (programme, tas, état des entrées/sorties)

Données privées
• Données de la pile
• Valeurs des registres
• Informations sur l'ordonnancement du thread
• Gardées dans le Thread Control Block (TCB

#include<pthread.h>
Add -lpthread in gcc

int pthread_create(threadid, attr, fonc, arg);

// Libère l'ensemble de ressources allouées à la fin du thread au processus initial
int pthread_detach(pthread_t threadid);

int pthread_exit(void * retval);

int pthread_join(pthread_t threadid, void * * retour)
int pthread_cancel(pthread_t threadid);

```
void * start(void * ptr)
{ int x = *(int *) ptr ;
printf("[thread id=%d ] *** New thread has x =%d \n", pthread_self(), x) ;
pthread_exit(NULL);
}
int main()
{
int i; int x=3;
pthread_t tid[5];
void * ptr = (void *)& x;
for(i =0; i<5;i++)
pthread_create(&tid[i], NULL, start, ptr ) ;
printf( "Main thread is running \n" ) ;
for(i =0; i<5;i++)
pthread_join(&tid[i], NULL) ;
return EXIT_SUCCESS;
}
```

Mutex

```
pthread_mutex_t SB;
int pthread_mutex_init ( &SB, &att); // attr NULL pour valeurs par défaut
// Init rapide : pthread_mutex_t SB= THREAD_MUTEX_INITIALIZER;
int pthread_mutex_destroy( &SB);

int pthread_mutex_lock ( &SB);
int pthread_mutex_unlock (&SB);
//retournent -1 si erreur
Int pthread_mutex_trylock ( &SB);
//retorun EBUSY si busy
```

Moniteurs

```
pthread_cond_t cd;
int pthread_cond_init ( &cd, &att);
// pthread_cond_t cd= THREAD_COND_INITIALIZER
pthread_cond_destroy (&cd);
```

Exemple :
Void allouer(int n)
pthread_mutex_lock(&mutex)
While (n>nlibre){
pthread_cond_wait(&c,&mutex)
}
nlibre-=n
pthread_mutex_unlock(&mutex)
}

Void liberer(int n)
pthread_mutex_lock(&mutex)
nlibre+=n
pthread_cond_brodcast(&c)
pthread_mutex_unlock(&mutex)
}

Semaphore
Généralisation des mutex(Semaphore Binaire)
P(): qui attend que le sémaphore soit positif et le décrémente de : puis-je ?
V(): incrémente le sémaphore de 1, et réveille les threads/processus bloquant = vas-y

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_post(& sem); //V
int sem_wait(&sem); //P
int sem_trywait(&sem); // non bloquante
int sem_destroy(sem_t * sem);

Sockets

#include <sys/types.h>
#include <sys/socket.h> /* socket(), bind() ... */

int socket
(
int domaine, /* AF_UNIX | AF_INET */
int type, /* SOCK_DGRAM | SOCK_STREAM */
int protocole /* 0 : protocole par défaut */
);
Supprimer une socket :
close(int sock).

```
#include <stdio.h>
#include <stdlib.h>
#include <error.h>
#include <string.h> /* memcpy(), strcmp() */
#include <unistd.h>
#include <netdb.h> /* gethostbyname() */
#include <sys/types.h>
#include <sys/socket.h> /* socket(), connect(), bind(), ... */
#include <netinet/in.h> /* pour les sokets AF_INET, htonl(), etc. */
#include <arpa/inet.h> /* inet_ntoa() */
```

Client.c

```
{     int socketfd, portno, n;
      struct sockaddr_in serv_addr;
      struct hostent *serv;
      char buffer[256];
      if (argc<3)                    {fprintf(stderr,"specifier les arguments de %s \n ", argv[0]);exit(-1);}
      portno=atoi(argv[2]);
      socketfd=socket(AF_INET, SOCK_STREAM,0);
      if (socketfd <0)               { printf ("erreur de creation"); exit(-1);}
      server =gethostbyname(argv[1]);
      if (server==NULL)              {printf ("erreur de nom serveur"); exit(-1);}
      bzero ((char*)&serv_addr,sizeof(serv_addr));
      serv_addr.sin_family= AF_INET;
      serv_addr.sin_port=htons(portno);
      memcpy (&serv_addr.sin_addr, server->h_addr,server->h_length);
      if(connect(socketfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0) {printf ("erreur de connection\n"); exit(-1);}
      bzero (buffer,256);
      printf ("\nDonner la chaine svp : \n");
      fgets(buffer,255,stdin);
      n=write(socketfd, buffer, strlen(buffer));
      if (n<0)              {printf ("erreur de ecriture\n"); exit(-1);}
      bzero(buffer,256);
      n=read(socketfd, buffer, 255);
      if (n<0)                       {printf ("erreur de lecture\n"); exit(-1);}
      printf("\n La reponse du serveur est : %s  \n", buffer);
return 0; }
```

Serveur.c

```
{       int socketfd, newsocketfd, portno, clilen, n;
        struct sockaddr_in serv_addr, cli_addr;
        char buffer[256];
        if (argc<2)                      {fprintf(stderr,"specifier le port ");exit(-1);}
        portno=atoi(argv[1]);
        socketfd=socket(AF_INET, SOCK_STREAM,0);
        if (socketfd <0)                 { printf ("erreur de creation"); exit(-1);}
        bzero ((char*)&serv_addr,sizeof(serv_addr));
        serv_addr.sin_family= AF_INET;
        serv_addr.sin_addr.s_addr= htonl(INADDR_ANY);
        serv_addr.sin_port=htons(portno);
        if(bind(socketfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)       {printf ("erreur de bind\n"); exit(-1);}
        listen(socketfd,5);
        clilen=sizeof(cli_addr);
        newsocketfd=accept(socketfd, (struct sockaddr *) & cli_addr, &clilen);
        if (newsocketfd <0)              { printf ("erreur de acceptation\n"); exit(-1);}
        bzero (buffer,256);
        n=read(newsocketfd, buffer, 255);
        if (n<0)                         {printf ("erreur de ecriture\n"); exit(-1);}
        printf("\nvoici le message reçu :%s\n", buffer);
        bzero(buffer,256);
        n=write(newsocketfd, "SERVEUR : jai bien reçu la requete", 40);
        if (n<0)                         {printf ("erreur de lecture\n"); exit(0);}
        return 0;
```