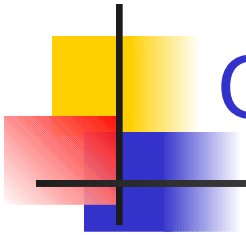




Luc Courtrai

# CONCURRENCE

**Université de Bretagne SUD**
**UFR SSI - Departement MIS**

# Plan

Notion de processus

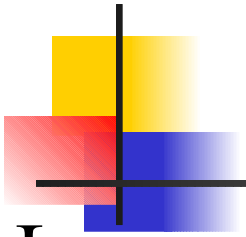
Les appels système UNIX

La synchronisation entre processus

**La communication entre processus**

Les threads JAVA

Les Posix threads

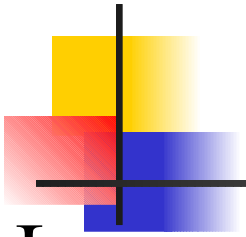


# Concurrence : communication

La communication entre processus.

Si les processus n’ont pas de mémoire partagée, il leur faut un moyen de communication pour échanger de l ’information.

L’outil le plus simple sur UNIX/LINUX est le tube.



# Concurrence : communication

## La communication entre processus. les tubes

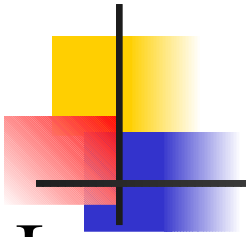
Un tube (pipe) permet une communication unidirectionnelle entre deux processus.

Un processus écrit dans le tube alors que le deuxième lit les information venant du tube. Le lecteur se synchronise sur l'arrivé d'information (lecture bloquante).

Le tube est un flot de données qui doit être contrôlé (découpé en message) par le programmeur.

# Concurrence : communication

La communication entre processus : les tubes



# Concurrence : communication

## La communication entre processus : les tubes

Une fois ouvert, le tube se manipule comme un fichier (flot)

Par exemple : les deux appels système  write, read permettent d ’écrire et de lire dans le tube (la lecture est bloquante).

La gestion des octets dans le tube est FIFO.

Si deux processus écrivent en même temps dans un tube, les info peuvent s'entrelacer,  il faut alors éventuellement gérer l'exclusion mutuelle sur le tube (le système gère des buffers cache).

# Concurrence : communication

La communication entre processus : les tubes

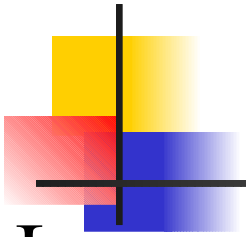
**Les tubes ordinaires.**

**Communication entre processus affiliés.**

```
#include<unistd.h>
int pipe(int p[2])
```

L ’appel système *pipe* permet la création d'un tube; la fonction retourne -1 en cas d ’échec.

Les tubes utilisent les descripteurs de fichiers des processus et les deux processus doivent posséder les même descripteurs (par héritage ex père fils). Un tube est accessible par le tableau *p* de deux descripteurs (*p[0]* lecture et *p[1]* écriture).



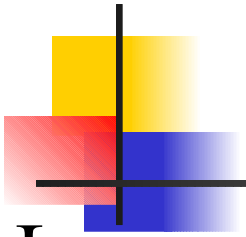
# Concurrence : communication

La communication entre processus : les tubes

**Les tubes ordinaires.**

```
int close(int fd)
```

L ’appel système *close* ferme un tube. La fermeture du descripteur d’écriture provoque une erreur sur le read effectué par l'autre processus.



# Concurrence : communication

La communication entre processus : les tubes

**Les tubes ordinaires : exemple.**

Un processus crée un tube puis crée un processus fils. Il lui envoie via le tube une suite de caractères saisie sur l ’entrée standard.

# Concurrence : communication

La communication entre processus : les tubes

```
#include<unistd.h>
int main ( ) {
  pid_t pid;
  int tube[2];// 2 descriteurs pour le tube

  // creation du tube
  if (pipe(tube) == -1) {
     perror( "pipe");
     exit(1);
  }
```

# Concurrence : communication

La communication entre processus : les tubes

```
// creation d 'un fils
if ((pid=fork())> 0) {
   char c;
   int nb;
   // processus pere
   close(tube[0]); // ferme la lecture
   while((nb=read(0,&c,1)) >0)
     write(tube[1],&c,1); // ecrit dans le tube
   close(tube[1]);
 } else {
   ...
```

# Concurrence : communication

La communication entre processus : les tubes

```
 } else {
    //  processus fils
    char c;
   int nb;
    close(tube[1]); // ferme l'écriture
    // lit dans le tube
    while((nb=read(tube[0],&c,1)) >0)
          write(1,&c,1); // sortie standard
    close(tube[0]);
    exit(0);
 } //else
} // main
```

# Concurrence : communication

## La communication entre processus : les tubes

**père**

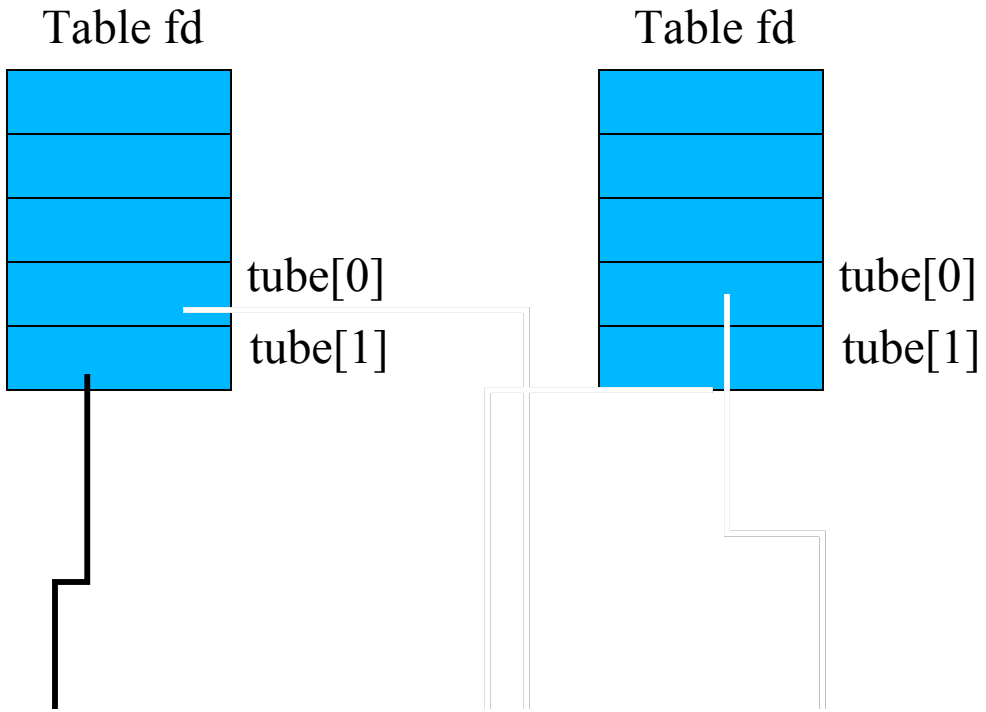
**pipe(tube)**

**fork()**

**close(tube[0])**

**read(0.. //bonjour**
**write(tube[1]);**

**read(0) // ctrl d**
**close(tube[1])**




**fils**

**close(tube[1])**

**read(tube[0]**
**write(1.. // bonjou**

**read(tube[0]) -1**

**close(tube[0]);**

# Concurrence : communication

La communication entre processus : les tubes

**Les tubes ordinaires : ps aux | wc - l**

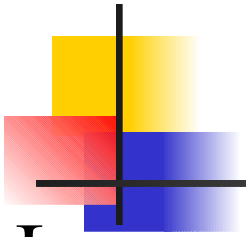
```
Les tubes d'un interpréteur de commande (shell)
exemple  : ps aux | wc -l
include<unistd.h>
int main ( ) {
  int tube[2];
  // creation du tube
  if (pipe(tube) == -1) {
     perror( "pipe");
     exit(1);
  }
```

# Concurrence : communication

## La communication entre processus : les tubes

**Les tubes ordinaires : le shell ps aux | wc - l**

```
if (fork()== 0) {  // pour le wc -l
   close(tube[1]); // ferme l'ecriture
   dup2(tube[0],0); // redirige l 'entree std
   execlp("wc","wc","-l",NULL)
   Exit(1) // Inutile sauf probleme du execlp
}
// père ps aux
close(tube[0]);
dup2(tube[1],1);
execlp("ps","ps","aux",NULL)
exit(1); // Inutile
}
```



# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés

Les tubes nommés (POSIX) permettent de nommer les tubes.

Les processus ne sont pas forcément affiliés (ce qui est le cas des tubes ordinaires).

Les tubes nommés sont des fichiers spéciaux dans le système de fichiers.

Les processus doivent donc connaître la même entrée UNIX.

*NB : Le fichier doit être créé sur le système de fichier de la machine où sont les deux processus*.

# Concurrence : communication

La communication entre processus : les tubes

**Les tubes nommés**

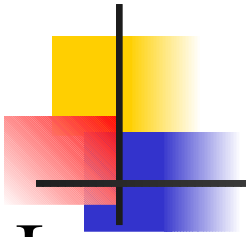
1 -- l 'appel système

```
#include<sys/types.h>
#include<sys/stat.h>
int mkfifo(char * path,mode_t mode);
```

crée un tube nommé avec les droits mode.

2 -- ou par la commande unix

```
> mkfifo pathfileTube
> chmod droit pathfileTube
```



# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés

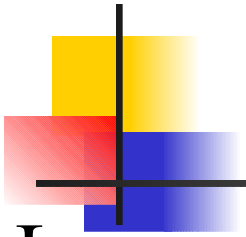
Une fois créé, le tube se manipule comme un fichier :

Il doit être ouvert en lecture ou en écriture par l'appel système open.

**Les deux processus se synchronisent sur le open (appel bloquant)**.

Les lectures (écritures) s ’effectuent par les primitives *read, write* (le read est bloquant). (appel système)

La fermeture s'effectue par la primitive close (pas de destruction du tubes)



# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés

Leur destructions s'effectuent via le système de fichiers par :

- la commande rm
- ou l'appel système unlink

# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés : exemple

```
>     mkfifo /tmp/tube; chmod 640 /tmp/tube
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
int main ( int argc , char **argv ) {
  mode_t mode=S_IRUSR|S_IWUSR|S_IRGRP;
  umask(0);
  if (mkfifo("/tmp/tube",mode) == -1) {
     perror("/tmp/tube");
     exit(1);
}
```

# Concurrence : communication

La communication entre processus : les tubes

**Les tubes nommés : exemple**

```
>       cat  > /tmp/tube
#include<fcntl.h>
int main ( int argc , char **argv ) {
  int t;char c;
  if ((t=open("/tmp/tube",O_WRONLY))< 0) {
    exit(1);
  }
  while(read(0,&c,1) >0)
    write(t,&c,1);
  close(t);
}
```

# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés : exemple

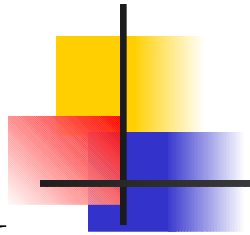
```
>       cat  /tmp/tube
#include<fcntl.h>
int main ( int argc , char **argv ) {
  int t;char c;
  if ((t=open("/tmp/tube",O_RDONLY))< 0) {
     exit(1);
  }
  while(read(t,&c,1) >0)
    write(1,&c,1);
  close(t);
}
```

# Concurrence : communication

## La communication entre processus : les tubes

### Les tubes nommés : exemple

```
>       rm  /tmp/tube

#include<unistd.h>
int main ( int argc , char **argv ) {
  if (unlink("/tmp/tube")== -1) {
     exit(1);
  }
}
```



# Concurrence : communication

La communication entre processus :

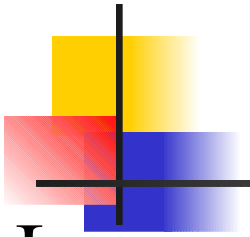
Les messages IPC (Inter Process Communication)

Communication par messages.

Avantages

**limite des messages** : Un message possède sa propre structure (le système préserve les limites)

**multiplexage** : Les processus se partage une file d'attente de messages permettant la communication entre plus de 2 processus (le système garantit la synchronisation).



# Concurrence : communication

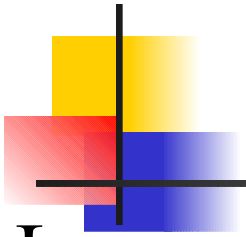
## La communication entre processus : Messages IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t cle,int option);
```

crée ou attache à une file de messages

Le champs clé : IPC_PRIVATE ou ftok

option : 0 ou IPC_CREAT IPC_EXCL et droits

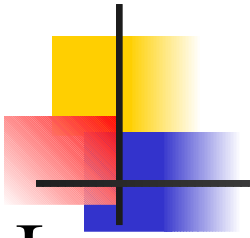


# Concurrence : communication

## La communication entre processus : Messages IPC

destruction d'une file

```
msgctl(msgid,IPC_RMID,NULL);
```



# Concurrence : communication

La communication entre processus : Messages IPC

Structure d'un message à définir par le programmeur de l 'application

```
struct message {
    long type; /* obligatoire */
     ...
}
```

Le type permet de filtrer les messages

Les données sont définies après le champ type

# Concurrence : communication

La communication entre processus : Messages IPC

primitive d'envoi de message :

```
int msgsnd(int msgid, void * mes,
    int lg,int option)
```

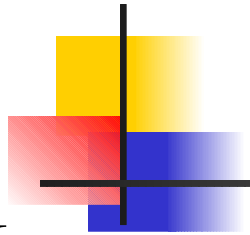
**msgid** : nom de la file

**mes** : pointeur sur la structure du message

**lg** : taille de la structure (sans le champs type)

**option** : 0 ou IPC_NOWAIT (échoue si la file est pleine, vérifier le code de retour -1 et errno EAGAIN)

La primitive retourne 0 en cas de succès.



# Concurrence : communication

La communication entre processus : Messages IPC

primitive de réception de message :

```
int msgrcv(int msgid, void * mes,int
  lg,long type,int option)
```

**msgid** : nom de la file

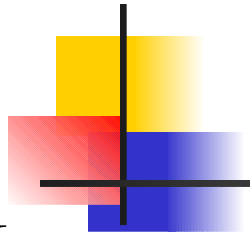
**mes** : pointeur sur la structure du message

**lg** : taille maxi de la structure (sans le champs type)

**type** : filtre

**option** : 0 ou IPC_NOWAIT (IPC_NOWAIT non bloquant probe, vérifier le code de retour -1 et errno EAGAIN)

La primitive retourne 0 en cas de succès



# Concurrence : communication

## La communication entre processus : Messages IPC

le champs `type pour msgrcv`

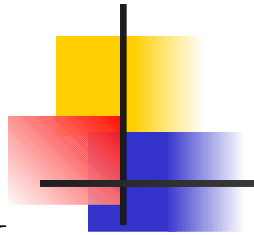
>0 le message le plus vieux correspondant au type est retourné

0 le message le plus vieux

<0 le message le plus vieux du type le plus petit ou égal à la valeur absolue de l'entier négatif est retourné

exemple type = -5 (filtre sur 1 2 3 4 5)

option : IPC_EXPERT le premier message de type différent de N, avec N > 0



# Concurrence : communication

## La communication entre processus : Messages IPC

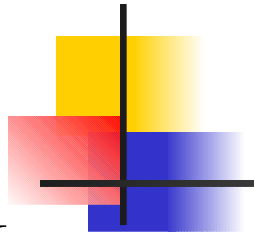
Exemple serveur de date

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define   APPLI   "/usr/local/bin/serveur"
    // cle pour ftok (binaire du serveur)
#define VERSION  '\0'
    // version pour ftok
#define SERVEUR 1 // type pour les messages
  destines au serveur (pid  impossible pour les
  clients)
```

# Concurrence : communication

La communication entre processus : Messages IPC

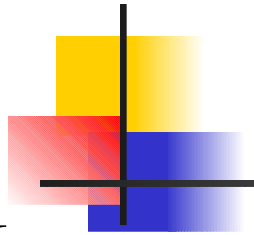
```
// type  message de requete
struct requete {
    long  type;  // serveur
    long  expe;  // data : pid de l 'expediteur
};
#define DATESIZE 64
// type message de reponse
struct reponse {
   long  type;          //  destinataire
   char  date[DATESIZE]; // data : heure systeme
};
```



# Concurrence : communication

La communication entre processus : Messages IPC

```
// serveur.c
//
// client serveur avec les messages IPC
//   partie serveur
//         le serveur fournit l'heure systeme
//         il cree la file de message
//   le nom de l executable doit s appeler
// /usr/bin/local/serveur pour la ftok
#include <stdio.h>
#include <time.h>
```



# Concurrence : communication

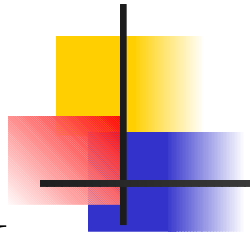
La communication entre processus : Messages IPC

```
int main ( int argc , char **argv ) {
  key_t cle; // cle ipc
  int    msgid; // file de message
  struct requete m;
  struct reponse rep;
  // genere le cle IPC
  if ((cle=ftok(APPLI,VERSION)) == -1 ) {
    fprintf(stderr,"Probleme sur ftoks\n");
    exit(1);
  }
```

# Concurrence : communication

La communication entre processus : Messages IPC

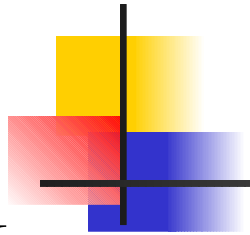
```
// creation d'une nouvelle file d'attente
 if ((msgid=msgget(cle,IPC_CREAT|IPC_EXCL|0666))
        ==-1) {
    fprintf(stderr,"Probleme de creation \
                    de la file\n");
    perror("msgget");
    exit(2);
 }
```



# Concurrence : communication

La communication entre processus : Messages IPC

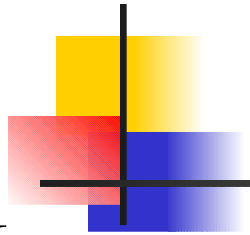
```
while (1) {
    time_t tps;
    // attente une requete (appel bloquant)
    if (msgrcv(msgid,&m, sizeof(long),SERVEUR,0)
                              == -1) {
      fprintf(stderr,"Probleme sur msgrcv\n");
      exit(3);
    }
    ...
```



# Concurrence : communication

La communication entre processus : Messages IPC

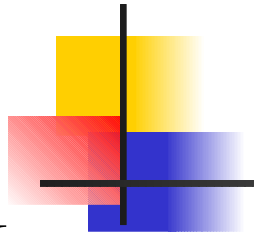
```
...
    // prepare la reponse
     rep.type=m.expe;
     tps = time(0); // recupere la date systeme
     strcpy(rep.date,ctime(&tps));

     // retour une valeur
     if (msgsnd(msgid,&rep, DATESIZE ,0) == -1) {
       fprintf(stderr,"Probleme sur msgsnd\n");
       exit(3);
       } //if
  } // while
} // main
```



# Concurrence : communication

## La communication entre processus : Messages IPC

```
// détruit la file
// soit a la reception d 'un signal
//    ou
// dans un programme spécifique
msgctl(msgid,IPC_RMID,NULL);
```



# Concurrence : communication

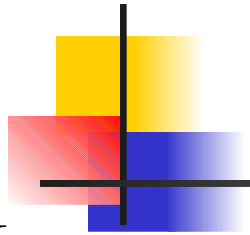
La communication entre processus : Messages IPC

```
// getDate.c
//  application client serveur avec les message
//   partie client
#include <stdio.h>
//programme principal
int main ( int argc , char **argv ) {
  key_t cle; /* cle ipc */
  int msgid;
  struct requete m;
  struct reponse r;
```

# Concurrence : communication

La communication entre processus : Messages IPC

```
// genere la meme  cle IPC que le serveur
 if ((cle=ftok(APPLI,VERSION)) == -1 ) {
    fprintf(stderr,"Probleme sur ftoks\n");
    exit(1);
  }
  // acces a la file d'attente file d'attente
  if ((msgid=msgget(cle,0))==-1) {
    fprintf(stderr,"Probleme sur la  file\n");
    exit(2);
  }
```



# Concurrence : communication

La communication entre processus : Messages IPC

```
m.expe = (long) getpid();
m.type = SERVEUR; /*code pour le servuer*/
// envoie la requete
if (msgsnd(msgid,&m,  sizeof(long),0) == -1) {
    fprintf(stderr,"Probleme sur msgsnd\n");
    exit(3);
}
```

# Concurrence : communication

La communication entre processus : Messages IPC

```
  // attente la reponse
  if (msgrcv(msgid,&r, DATESIZE,(long)getpid(),0)
                              == -1) {
     fprintf(stderr,"Probleme sur msgrcv  \
                    reponse\n");
     exit(3);
   }
   printf("Date %s  \n",r.date);
}
```