Reseau

Fork

Pid_t pid; -> création d'une variable pid

Pid = fork(); -> -1 = erreur, 0 = fils, 0 > père (récupère pid du fils)

int status;

wait(&status); -> attends la fin du processus fils

Pipe

int descripteur[2]; -> création de la variable

pipe(descripteur); -> création du pipe (0 = lecture, 1 = ecriture)

close(descripteur[0]); -> Fermeture du tube en lecture

Write(descripteur[1], message, taille_message) -> Ecriture dans le pipe

Read(descripteur[0], message, taille_message) -> lecture dans le pipe

Thread

pthread_t st; -> création d'une variable thread

pthread_create(&st, NULL, fonction, &i ou &tab[i]) -> création d'un thread

pthread_join(st, (void **) &test ou &tab[i] ou NULL) -> Attente de la fin d'un thread

Le test ou tab sont des pointeurs dans ce cas

pthread_exit(res ou NULL); -> Valeur que l'on retourne et récupère avec pthread_join

Le res est un pointeur dans ce cas

Pthread_mutex_t Verrou = pthread_mutex_initializer; -> création d'un verrou

Pthread_mutex_lock(&Verrou); -> Bloque le verrou

Pthread_mutex_unlock(&Verrou); -> Débloque le verrou

Void *SommeP(Void* par){}; -> Fonction pour un thread

int *test = (int*) par; -> cast d'un paramètre Void *

pthread_cond_t cond; -> création d'une variable conditionnelle

pthread_cond_wait(&cond, &Verrou); -> Déblocage et mise en pause du thread

pthread_cond_broadcast(&cond); → Débloque les thread bloqué avec wait

Structure

```
typedef struct valeurs
{
   int val1;
   int val2;
} valeurs;
```

IPC

- → Un objet IPC peut être privé ou public
- → Le message lu disparait de la file
- → Vie jusqu'à sa destruction (au delà de la vie des processus accédant)
- → Si file pleine, le processus voulant déposer est endormi et sera réveiller quand il y aura de la place
- → Si file vide, le processus voulant extraire est endormi et sera réveiller dès qu'il y aura un message
- → Si file privé, le processus voulant y accéder doit connaitre la clé car pas possible de la récupérer avec ftok(). (pipe, Bouche d'oreille, file publique)

```
key_t cle = ftok("toto.txt", 'G');  → création d'une clé
int f_id = msgget(cle, IPC_CREAT | 0666);  → création d'une file
int res = msgctl(f_id, IPC_RMID, NULL);  → Destruction d'une file

typedef Struct message
{
   long monetiquette;
   long étiquetteRetour;
   int premier;
   ...
} message;
```

→ Structure type d'un message IPC avec toujours comme premier parametre l'étiquette du destinataire

```
message mes;
int retour = msgsnd(f_id, &mes, (sizeof(message) - sizeof(long)), 0);
```

↳ envoie un message dans la file

```
retour = msgrcv(f_id, &mes, (sizeof ...), pid, 0);  → reçois un message de la file
```

(Pid = identifiant du destinataire du message, peut être son pid ou un int quelconque)

mémoire partagé

```
Key_t cle = ftok("toto.txt", 'G'); -> création d'un clé
int id = shmget(cle, sizeof(int), IPC_CREAT | 0666); -> création mémoire
int *mbplace;
mbplace = (int *) shmat(id, NULL, 0); -> attachement à la mémoire partagé
*mbplace = 100; -> accès à la mémoire via le pointeur
shmdt((void *) mbplace); -> détachement de la mémoire partagé
shmctl(id, IPC_RMID, NULL); -> Suppression mémoire partagé
```

Sémaphore

```
Key_t cle = ftok("toto.txt", 'G'); -> création d'une clé
int id = semget(cle, 1, IPC_CREAT | 0666); -> création d'un sémaphore
semctl(id, 0, SETVAL, egctl); -> initialisation du sémaphore
                                 -> Possibilité de mettre directement une valeur
semctl(id, 0, IPC_RMID); -> destruction du sémaphore
union semun egctl; } -> Pour initialiser le sémaphore à 1
egctl.val = 1;

Struct sembuf opp; } -> Structure utilisé pour opéré sur le sémaphore
opp.sem_num = 0;
opp.sem_op = -1 si décrémente ou 1 si incrémente;

Semop(id, &opp, 1); -> opération sur le sémaphore
```

Socket

```
int descSock = socket(PF_INET, SOCK_DGRAM, 0); -> mode non connecter (buffer)
int descSock = socket(PF_INET, SOCK_STREAM, 0); -> mode connecter (buffer)
Struct sockaddr_in ad; -> création socket
ad.sin_family = AF_INET;
ad.sin_addr.s_addr = INADDR_ANY; } -> configuration socket
ad.sin_port = htons((short) 0);
int res = bind(descSock, (struct sockaddr*) &ad, sizeof(ad)); -> liaison buffer et socket
```

```
struct sockaddr_in client;
socklen_t lgclient = sizeof(client);
```
} → Structure du client pour le serveur

```
struct sockaddr_in serveur;
Serveur.sin_family = AF_INET;
Serveur.sin_addr.s_addr = inet_addr("127.0.0.1");
Serveur.sin_port = htons((short)31000);
```
} → Structure du serveur pour le client

mode non connecté :

```
int recv = recvfrom(ad, mes, sizeof(mes), 0, (struct sockaddr *)&client, &lgclient);
int retour = Sendto(ad, mes, sizeof(mes), 0, (struct sockaddr *)&client, lgclient);
```

mode connecté :

```
int acclient = accept(ad, (struct sockaddr *)&client, &lgclient); → coté serveur
connect(ad, (struct sockaddr *)&serveur, sizeof(serveur); → coté client

recv(acclient, mes, sizeof(mes), 0); → ad à la place de acclient pour le client
send(acclient, &mes, sizeof(mes), 0); → ad à la place de acclient pour le client
```