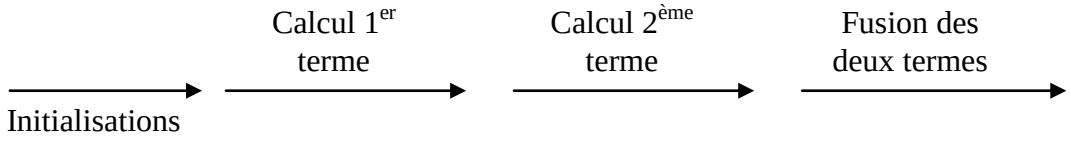
IV. MEMOIRES PARTAGEES

1. INTRODUCTION

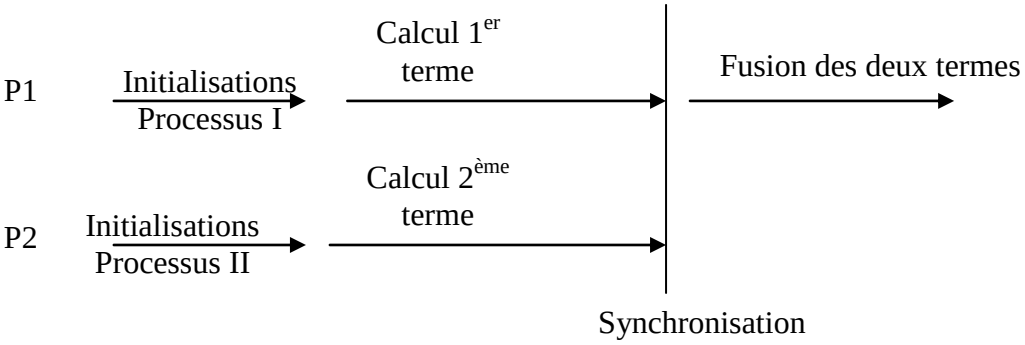
Travailler en parallèle n'a d'intérêt que si les processus peuvent échanger des données. En effet, si les processus travaillent uniquement sur des zones mémoires différentes, autant écrire directement un programme linéaire classique.

Or si nous pouvons affecter à chaque processus une tâche indépendante du résultat d'un autre processus, le gain de temps d'exécution sera intéressant :

Programmation linéaire :



Programmation parallèle



Pour réaliser un tel programme, il faudra bien que le processus qui calcule le deuxième terme puisse retransmettre au processus qui fera le calcul de premier terme puis la fusion des résultats, la valeur qu'il a calculé.

Nous sommes confrontés à des fonctionnalités permettant à différents processus d'échanger des informations en d'autres terme de communiquer.

Nous allons donc gérer une mémoire partagée. Le fait qu'elle soit déclarée comme partagée, ne signifie pas que les problèmes d'exclusion mutuelle sont gérés. Cela signifie que la zone mémoire est la même. Par conséquent, si les processus y accèdent de manière normale, les processus pourront travailler sur les mêmes données.

Les méthodes de communication les plus connues sont les fichiers, les tubes, tubes nommés et les sockets (pour des processus qui s'exécutent dans des machines différentes). Ces méthodes font appel au système de fichier. Ce qui signifie des mécanismes de communication qui peuvent être lourds et lents (localisation chargement en mémoire, accès, ...)

Les IPC (Inter Process Communication), apparus avec le Système V d'UNIX et repris dans linux, ne sont pas des fichiers mais des éléments mémoires gérés par le noyau et offrent de meilleures performances de gestion. Il existe 3 sortes d'IPC :
- Les files de messages
- Les segments de mémoire partagés
- Les sémaphores

Les IPC sont gérées par le noyau par l'intermédiaire de tables une pour chaque type. Ils sont identifiés par :

- En interne : chaque objet de type IPC est identifié au sein du système par un identificateur (positif ou nul), qui joue un rôle similaire aux identificateurs de fichier. Chaque processus qui souhaite utiliser un des ces objets doit connaître son identificateur.

- En externe, les IPC sont identifiés par une clé, qui joue le rôle des références pour les fichiers.

Ce cours est consacré aux segments de mémoires partagées

2. PRINCIPE

Une mémoire partagée est repérée par le système par l'intermédiaire d'un code numérique nommée la clé. Lorsqu'un processus demande l'accès à une certaine zone mémoire, il doit passer en paramètre cette clé qui devra être la même dans chaque processus utilisant la même mémoire. C'est au programmeur de gérer ces appels. Il conviendra également de vérifier que cet appel fonctionne bien.

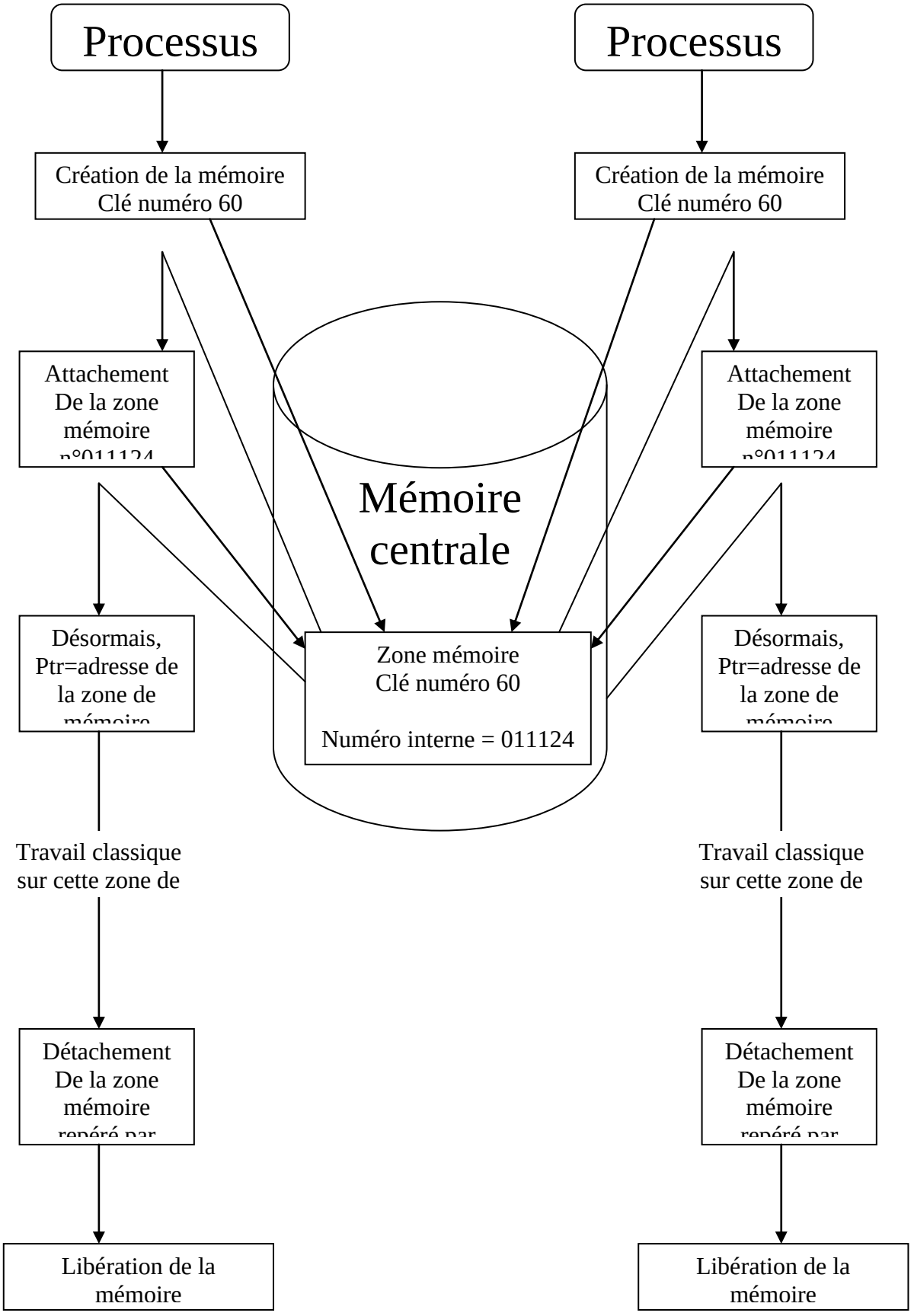
Chaque processus qui a besoin d'utiliser une mémoire partagée avec d'autres processus devra observer une certaine méthode :

En premier il faudra que le processus demande la création de cette zone si elle n'existe pas, ou demande l'accès à cette zone sinon. Cette procédure nous retournera un numéro de mémoire qui n'a rien à voir avec la clé.

En second, il faudra que chaque processus demande de rattacher cette zone à son espace d'adresses . Il s'agit d'une « conversion » de ce numéro en un pointeur. Cela permet d'accéder à cette zone mémoire de manière simple.

A partir de maintenant, cette mémoire partagée s'utilise comme une variable classique. A la fin de l'exécution de ce programme, il faudra provoquer le détachement.

Puis, il faudra informer le système que ce processus n'utilise plus cette mémoire et si plus aucun processus ne l'utilise, elle sera détruite.


Processus
Processus
Création de la mémoire
Clé numéro 60
Création de la mémoire
Clé numéro 60
Attachement
De la zone
mémoire
Attachement
De la zone
mémoire
Mémoire
centrale
Désormais,
Ptr=adresse de
la zone de
Désormais,
Ptr=adresse de
la zone de
Zone mémoire
Clé numéro 60
Numéro interne = 011124
Travail classique
sur cette zone de
Travail classique
sur cette zone de
Détachement
De la zone
mémoire
Détachement
De la zone
mémoire
Libération de la
mémoire
Libération de la
mémoire

3. CREATION SHMGET

Elle permet la création d'un segment de mémoire partagé entre différents processus.

Syntaxe : *int* shmget (*int* cle, *int* taille, *int* flag)

- *clé* correspondant à une valeur d'identification commune à tous les processus.
- *taille* représente la taille de la mémoire partagée
- *flag* correspond aux options de création :

  -IPC_CREAT : pour créer un nouveau segment. Sinon **shmget**() recherchera le segment associé à *clé*, vérifiera que l'appelant a la permission d'y accéder

  -IPC_EXCL : utilisée en conjonction avec IPC_CREAT, permet l'émission d'une erreur si l'IPC existe déjà.

Ses options peuvent êtres combiner avec les permissions d'accès (Lecture, Ecriture)

shmget() renvoie l'identifiant du segment de mémoire partagée associé à la valeur de l'argument clé, -1 en cas d'erreur.

Exemple :

Création d'une mémoire partagée permettant de manipuler un caractère :

```
#define CLE_MEM 60

main()
{int shmid;

shmid = shmget(CLE_MEM, sizeof(char), IPC_CREAT | 0666) ;
if (shmid == -1)
    {
      perror("shmget");
     exit(0);
    }
.....
}
```

4. ATTACHEMENT : SHMAT

shmat() : Elle permet l'attachement d'un segment de mémoire partagé à l'espace d'adresse d'un processus. En effet, les seules adresses accessibles à un programme sont celles appartenant à son espace d'adresses. L'opération d'attachement consiste à agrandir l'espace d'adressage d'un processus en intégrant le segment de mémoire partagée permettant ainsi l'accès aux données qu'il contient.

Syntaxe : *void* *shmat (*int* shmid, *void* *adresse, *int* flag)

Elle attache le segment de mémoire partagée identifié par shmid au segment de données du processus appelant.

shmid : identificateur du segment de mémoire partagée que l'on désire attaché

adresse : Adresse à laquelle on souhaite faire l'attachement du segment de mémoire partagée. Si adresse vaut NULL, le système essaye de trouver une adresse libre pour attacher le segment

flag : option d'attachement, peu d'options sont réellement disponibles, la plus utile étant SHM_RDONLY qui indique que le segment est attaché en mode lecture seulement. Par défaut elle prend la valeur 0.

Valeur de retour : Cette fonction retourne un pointeur sur la zone mémoire qui est attachée. Il faut ainsi, penser à faire une conversion de type sur cette variable (faire un cast, de manière identique à la fonction malloc()).

Exemple :

```
#define CLE_MEM 60
char *ptr ;
int shmid;

shmid = shmget(CLE_MEM, sizeof(char), IPC_CREAT | 0666) ;
if (shmid == -1)
   {
     perror("shmget");
    exit(0);
   }

ptr = (char*) shmat(shmid, 0, 0);

*ptr='A'
.............
```

5. DETTACHEMENT : SHMDT

Elle permet de supprimer l'attachement d'une mémoire partagée à l'espace adressable d'un processus.

Syntaxe : int shmdt(const void *adresse)

adresse : est le pointeur obtenu lors de l'attachement.

shmdt() : retourne 0 si tout s'est bien passé -1 en cas d'erreur

```
#define CLE_MEM 60
char *ptr ;
int shmid;

shmid = shmget(CLE_MEM, sizeof(char), IPC_CREAT | 0666) ;
ptr = (char*) shmat(shmid, 0, 0);

/*Utilisation de la variable repérée par ptr */
*ptr='A'
.....
/* Détachement */
shmdt(ptr) ;
```

6. OPERATIONS DE CONTROLE : SHMCTL

Elle permet d'exécuter des opérations de contrôle sur un segment de mémoire partagée.

Syntaxe : shmctl (*int shmid* , *int* cmd, *struct shmid_ds* *buf)

shmctl() : effectue l'opération de contrôle indiquée dans **cmd** sur le segment de mémoire partagée dont l'identifiant est fourni dans shmid. L'argument buf est un pointeur sur une structure shmid_ds définie dans <sys/shm.h> de la façon suivante :

```
struct shmid_ds {
   struct ipc_perm shm_perm;    /* Appartenace et permissions       */
   size_t          shm_segsz;   /* Taille segment en octets         */
   time_t          shm_atime;   /* Heure dernier attachement        */
   time_t          shm_dtime;   /* Heure dernier détachement        */
   time_t          shm_ctime;   /* Heure dernière modification      */
   pid_t           shm_cpid;    /* PID du créateur                  */
   pid_t           shm_lpid;    /* PID du dernier shmat(2)/shmdt(2) */
   shmatt_t        shm_nattch;  /* Nombre d'attachements actuels    */
   ... };
```

Les valeurs possibles pour cmd sont :

IPC_RMID : suppression du segment identifié par shmid. Cette destruction est différée si le segment est encore attaché à d'autres processus.

IPC_STAT : demande d'information sur le segment par le remplissage de la structure pointée par buf.

IPC_SET : demande de modification des caractéristiques de l'entrée identifiée par shmid avec les informations contenues dans la structure pointée par buf.

Exemple : suppression d'une mémoire partagée

```
#define CLE_MEM 60
char *ptr ;
int shmid;

shmid = shmget(CLE_MEM, sizeof(char), IPC_CREAT | 0666) ;
ptr = (char*) shmat(shmid, 0, 0);

/*Utilisation de la variable repérée par ptr */
.....

/*Détachement */
shmdt(ptr) ;

/*suppression de la mémoire partagée */
shmctl(shmid, IPC_RMID, NULL) ;
```

7. EXEMPLE

```
main(int argc, char *argv[])
{
 int idM, *N, pid;

 pid=fork();
 if(pid==0)
 {
 idM=shmget(key,sizeof(int),IPC_CREAT|0666);
 if(idM==-1)
        {
        perror("shmget");
        exit(0);
        }
 N=(int *)shmat(idM,0,0);
 *N=2;
 printf("Fils : %d\n",*N);
 shmdt(N);
}
else {
wait();
 idM=shmget(key,sizeof(int),IPC_CREAT|0666);
 if(idM==-1)
        {
        perror("shmget");
        exit(0);
        }
 N=(int *)shmat(idM,0,0);
 *N=*N +3;
 printf("Pere : %d\n",*N);
 shmdt(N);
 shmctl(idM, IPC_RMID, NULL);
}
}
```

Deux processus avec lien de parenté (Père/Fils) partagent une variable entière **n.**
L'affichage donne :
Fils : 2
Père : 5