TP2 ET 3

SEMAPHORE ET MEMOIRE PARTAGEE

CS366 : Systèmes d’exploitations

## Binôme :

## Jessy GIACOMONI

## Adrien ROGIER

## Enseignant :

## Stéphanie CHOLLET

## Eric SIMON

# Sommaire

# Avant propos :

## But du Projet

Ce projet a pour but de nous familiariser avec la gestion des processus, les sémaphores et que les mémoires partagées. Dans ce projet, nous devrons gérer la création et l’ordonnancement des processus entre père et fils, la gestion des accès mémoires et fichier, la protection des données.

## Rappel des objectifs

L’objectif de ce TP est de réaliser une chaîne d’acquisition de données. Le résumé en schéma :

Processus

Acquisition

Processus

Stockage

Processus

Traitement

Mémoire

Data\_1\_X.txt

Data\_2\_X.txt

Figure 1 : Chaîne d'acquisition

Nous avons 4 processus à gérer dans cette chaîne d’acquisition :

* **Processus père :** Créer les 3 processus fils de la chaîne d’acquisition. Il veille à ce que chaque fils exécute bien son code.
* **Processus d’acquisition :** Génère des données aléatoires, puis stockent les valeurs dans une mémoire allouée par l’ordinateur.
* **Processus de stockage :** Ce processus récupère les valeurs stocké dans la mémoire alloué par le processus d’acquisition. Puis stockent ces valeurs dans un fichier texte *Data\_1\_X.txt*.
* **Processus de traitement :** Récupère les valeurs dans le fichier Data\_1\_X.txt, y effectuent un traitement, et stockent les résultats dans un fichier texte Data\_1\_X.txt.

# Contenu :

Ce projet est composé de 8 fichiers (+2 avec le bonus):

## Un fichier main.c

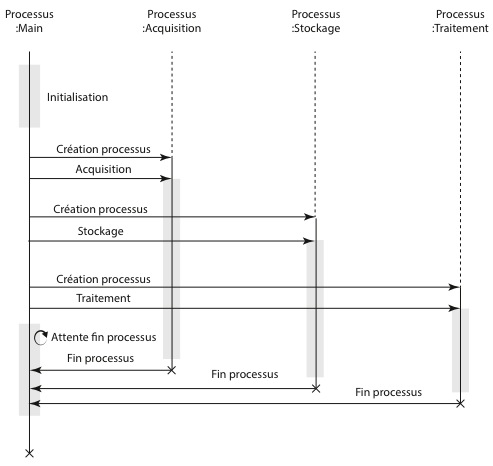
Ce fichier contient le code du processus père.

### Algorithme

\_

|  |
| --- |
| Fonction main(nbrSerie, delaiEntreSerie, nbrAcquisition, delaiAcquisition)  Initialisation (Sémaphore + Mémoire partagée)  Création processus acquisition  Création processus stockage  Création processus traitement  Attente de la fin de tous les processus  Destruction de la mémoire partagée et des sémaphores  Fin Fonction |

### Diagramme de séquence :



## Fichier acquisition.c/h :

L'acquisition des données va être simulée : les données seront l'heure actuelles du système.

La fonction gettimeofday() permet de récupérer cette information par l'intermédiaire d'une structure de donnée timeval :

### Algorithme

### Organigramme

### Méthodes

|  |
| --- |
| #include <sys/types.h>  #include <unistd.h>  struct timeval {  int tv\_sec; // seconde  int tv\_usec; // Microsecondes  }; |

Les données aléatoires seront générées par cette formule :

**valeur = µsecode\*10e6 + seconde**

## Un programme stockage.c/h :

Cette fonction prends les données passées en paramètre par la variable Mesure[ ] de type tableau. Elle copie ensuite les valeurs du tableau dans un fichier et supprime les valeurs du tableau copié dans le fichier.

### Algorithme

### Organigramme

### Méthodes

## Un programme traitement.c/h :

Cette fonction récupère les valeurs présentes dans le fichier data\_1\_X.txt. Puis traite les données du fichier et stocke le résultat dans un fichier data\_2\_X.txt. X étant le numéro de la série.

### Algorithme

### Organigramme

### Méthodes

## Programme courbe.c/h

Fonction bonus. Cette fonction nécessite l'installation du logiciel Gnuplot. Ce logiciel permet de tracer des courbes. Cette fonction s'intégrera dans la fonction traitement. Elle dessinera les courbes en fonction des valeurs de data\_2\_X.txt.

### Algorithme

### Organigramme

### Méthodes

## Un Makefile

### Algorithme

### Organigramme

### Méthodes

|  |
| --- |
| Est-ce qu’il faut mettre cette partie ? Transition Acqusition/Stockage Nous allons tout d'abord récupérer une série d'acquisition. Lorsque celui-ci est fait alors nous allons stocker. Durant, le délai entre les séries nous allons donc en profiter pour donner la main au stockage. Lors de la terminaison de celui-ci nous allons redonner la main à l'acquisition pour la prochaine série.   * Remarque: Cependant, un problème peut persister. En effet, si le temps de stockage est supérieur au délai entre les séries alors le délai de sera donc pas respecté. |

# Communication avec les IPCs Système V

## Principes

* Les segments de mémoire partagée, qui sont accessibles simultanément par deux processus ou plus, avec éventuellement des restrictions telles que la lecture seule.
* Les sémaphores, qui permettent de synchroniser l'accès à des ressources partagées.

Dans tous les cas, ces outils de communication peuvent être partagés entre des processus n'ayant pas immédiatement d'ancêtre commun. Pour cela, les IPC introduisent le concept de clé.

Une ressource IPC partagée est accessible par l'intermédiaire d'un nombre entier servant d'identificateur (l'ensemble de sémaphores, identifiant du segment mémoire), qui est commun aux processus désirant l'utiliser.

Pour partager ce numéro d'identification, consiste à demander au système de créer lui-même une clé, fondée sur des références communes pour tous les processus. La clé est constituée en employant un nom de fichier et un identificateur de projet. De cette manière, tous les processus d'un ensemble donné pourront choisir de créer leur clé commune en utilisant le chemin d'accès du fichier exécutable de l'application principale, ainsi qu'un numéro de version par exemple.

Une clé est fournie par le système sous forme d'un objet de type key\_t, défini dans <sys/type.h>. La constante symbolique IPC\_PRIVATE, définie dans <sys/ipc.h> représente une clé privée, demandant sans condition la création d'une nouvelle ressource IPC.

Pour créer une nouvelle clé à partir d'un nom de fichier et d'un identificateur de projet, on emploie la fonction ftok( ). Déclarée ainsi dans <sys/ipc.h> :

*key\_t ftok (char \* nom\_fichier, char projet);*

La clé créée emploie une partie du numéro d'i-noeud du fichier indiqué, le numéro mineur du périphérique sur lequel il se trouve et la valeur transmise en second argument pour faire une clé sur 32 bits :

|  |  |  |
| --- | --- | --- |
| 31...24 | 23...16 | 15...0 |
| Numéro projet & OxFF | Mineur périphérique & OxFF | Numéro i-noeud & OxFFFF |

La fonction ftok() ne garantit pas réellement l'unicité de la clé, car plusieurs liens matériels sur le même fichier renvoient le même numéro d'i-noeud. De plus, la restriction au numéro mineur de périphérique ainsi que l'utilisation seulement des 16 bits de poids faibles de 1'i-nœud rendent possible l'existence de fichiers différents renvoyant la même clé.

## Ouverture de l'IPC

L'obtention de la ressource IPC se fait à l'aide de l'une des deux commandes shmget( ) et semget( ). Les détails d'appel seront précisés plus bas, mais ces fonctions demandent au système de créer éventuellement la ressource si elle n'existe pas, puis de renvoyer un numéro d'identification. Si la ressource existe déjà et si le processus appelant n'a pas les autorisations nécessaires pour y accéder, les routines échouent en renvoyant -1.

À partir de l'identifiant ainsi obtenu, il sera possible respectivement :

* D'attacher puis de détacher un segment de mémoire partagée dans l'espace d'adressage du processus avec shmat( ) ou shmdt( ) ;
* De lever de manière bloquante ou non un sémaphore, puis de le relâcher avec la fonction commune semop( ).

## Contrôle et paramétrage

Les IPC proposent quelques options de paramétrage spécifiques au type de communication, ou générales. Pour cela, il existe deux fonctions, shmctl( ) et semctl( ), qui permettent de consulter des attributs regroupés dans des structures shmid\_ds et semid\_ds. Dans tous les cas, ces structures permettent l'accès à un objet de type struct ipcperm, défini ainsi dans <sys/ipc.h> :

|  |  |  |
| --- | --- | --- |
| Nom | Type | Signification |
| \_\_key | key\_t | Clé associée à la ressource IPC |
| \_\_seq | unsigned short | Numéro de séquence, utilisé de manière interne par le système, à ne pas toucher |
| mode | unsigned short | Autorisations d'accès à la ressource, comme pour les permissions des fichiers |
| uid | uid\_t | UID effectif de la ressource IPC |
| gid | gid\_t | GID effectif de la ressource IPC |
| cuid | uid\_t | UID du créateur de la ressource |
| cgid | gid\_t | GID du créateur de la ressource |

Les fonctions de contrôle permettent également de détruire une ressource IPC. En effet, un ensemble de sémaphores ou une zone de mémoire partagée restent présents dans le noyau même s'il n'y a plus de processus qui les utilisent. Ceci présente l'avantage d'une persistance des données entre deux lancements de la même application mais pose aussi l'inconvénient d'une utilisation croissante de la mémoire du noyau sans libération automatique. Il est donc possible de demander explicitement la destruction d'une ressource IPC. Les processus en train de l'employer recevront une indication d'erreur lors de la tentative d'accès suivante.

## Mémoire partagée

Le système de la mémoire partagée offert par les IPC Système V est le suivant:

* Une fonction shmget( ) permet à partir d'une clé key\_t d'obtenir l'identifiant d'un segment de mémoire partagée existant ou d'en créer un au besoin.
* L'appel-système shmat( ) permet d'attacher le segment dans l'espace d'adressage du processus.
* La fonction shmdt( ) sert à détacher le segment si on ne l'utilise plus.
* Enfin, l'appel-système shmctl( ) permet de paramétrer ou de supprimer un segment partagé.

Les prototypes de ces routines sont déclarés dans <sys/shm.h> ainsi :

int shmget (key\_t key, int taille, int attributs);

char \* shmat (int identifiant, char \* adresse, int attributs);

int shmdt (char \* adresse);

int shmctl (int identifiant, int commande, struct shmid\_ds \* attributs);

L'appel-système shmget( ) fonctionne comme il suit, en employant la clé transmise en premier argument pour rechercher ou créer un bloc de mémoire partagée. Les attributs indiqués en dernière position comportent les 9 bits de poids faibles de l'autorisation d'accès, et éventuellement les constantes IPC\_CREAT et IPC\_EXCL.

Le second argument de cette routine est la taille du segment désiré, en octets. Cette taille sert lors de la création d'une nouvelle zone de mémoire partagée. La valeur indiquée est arrondie au multiple supérieur de la taille des pages mémoire sur le système (4 Ko sur un PC). Si la taille demandée lors de la création est inférieure à la valeur SHMMIN ou supérieure à SHMMAX, une erreur se produit. Pour accéder à une zone mémoire déjà existante, il faut demander une valeur inférieure ou égale à la taille effective du segment. On emploie généralement zéro dans ce cas, car le système ne réduit pas la taille de la projection d'un segment existant.

Une fois obtenu l'identifiant d'un segment partagé, on doit l'attacher dans l'espace mémoire du processus à l'aide de la fonction shmat(). On indique en second argument l'adresse désirée pour l'attachement. Si cette adresse est nulle, le noyau recherche un emplacement libre dans l'espace d'adressage du processus, y réalise la projection, et l'appel-système shmat() renvoie l'adresse du premier octet de la zone partagée. L'attachement peut être réalisé en lecture seule si l'attribut SHM\_RDONLY est passé en troisième argument de shmat(), sinon la projection est réalisée en lecture et écriture.

La fonction shmctl( ) permet, d'agir sur un segment partagé. La commande employée en seconde position peut être :

* IPC\_STAT : pour remplir la structure shmi dds que nous allons détailler ci-dessous.
* IPC\_SET : pour modifier l'appartenance ou les autorisations d'accès au segment.
* IPC\_RMID : pour supprimer le segment. Ce dernier est alors marqué comme «prêt pour la suppression », mais ne sera effectivement détruit qu'une fois qu'il aura été détaché par le dernier processus qui l'utilise. Cela signifie aussi que tant qu'un processus conserve le segment attaché, il est toujours possible de le lier à nouveau avec shmat(), même s'il a été marqué pour la destruction.
* SHM\_LOCK : permet de verrouiller le segment en mémoire pour s'assurer qu'il ne sera pas envoyé sur le périphérique de swap. Cette opération réduisant la mémoire vive disponible pour les autres processus, elle est privilégiée et nécessite un UID nul ou la capacité CAP\_IPC\_LOCK.
* SHM\_UNLOCK : permet symétriquement de déverrouiller une page de la mémoire, autorisant à nouveau son transfert en mémoire secondaire.

La structure shmidds contenant les paramètres associés au segment de mémoire partagée comprend notamment les membres suivants :

|  |  |  |
| --- | --- | --- |
| Nom | Type | Signification |
| shm\_perm | struct ipc\_perm | Autorisation d'accès au segment de mémoire |
| shm\_segsz | size\_t | Taille en octets du segment |
| shm\_atime | time\_t | Heure du dernier attachement |
| shm\_dtie | time\_t | Heure du dernier détachement |
| shm\_ctime | time\_t | Heure de la dernière modification des autorisations |
| shm\_cpid | pid\_t | PID du processus créateur du segment |
| shm\_lpid | pid\_t | PID du processus ayant réalisé la dernière intervention |
| shm\_nattch | unsigned short | Nombre actuel d'attachements en mémoire |

L'utilisation des segments de mémoire partagée est le mécanisme de communication entre processus le plus rapide, car il n'y a pas de copie des données transmises. On évite notamment le transfert des informations entre l'espace mémoire de l'utilisateur et l'espace mémoire du noyau, à la différence de msgsnd( ) sur les files de messages, ou même de write( ) lors de l'utilisation de sockets BSD. Ce procédé de communication est donc parfaitement adapté au partage de gros volumes de données entre processus distincts.

Il est indispensable, lors de l'accès à des ressources communes, de synchroniser les différents acteurs, pour éviter les interférences regrettables. Pour cela, on dispose d'un dernier mécanisme IPC servant à organiser l'utilisation des mémoires partagées : les sémaphores.

## Sémaphores

Un sémaphore est dans sa forme la plus simple un drapeau qui peut être levé ou baissé. Il sert à contrôler l'accès à une ressource critique grâce à deux opérations :

* Avant l'accès, un processus attend que le drapeau soit levé, puis il le baisse.
* Après avoir utilisé la ressource protégée, le processus relève le drapeau, et le noyau réveille les autres processus bloqués dans l'opération précédente.

Le test qui intervient dans la première de ces opérations est atomiquement lié à la modification qui le suit. Ceci garantit qu'en aucun cas deux processus ne verront simultanément le drapeau baissé et se l'attribueront.

Les routines servant à manipuler les sémaphores sont semop( ), qui regroupe les opérations Pn( ) et Vn( ) et semctl( ) , qui permet entre autres de configurer ou de supprimer un ensemble de sémaphores.

Leurs prototypes sont déclarés dans <sys/sem.h> ainsi :

int semget (key\_t key, int nombre, int attributs);

int semop (int identifiant, struct sembuf \* operation, unsigned nombre);

int semctl (int identifiant, int numero, int commande, union semun attributs);

L'appel-système semget( ) fonctionne comme ses confrères msgget( ) et shmget( ), avec simplement en second argument le nombre de sémaphores dans l'ensemble. Cette valeur n'est prise en compte que lors de la création de la ressource, pas au moment de l'accès à un ensemble existant. Le troisième argument peut contenir comme d'habitude IPC\_CREAT, IPC\_EXCL et les autorisations d'accès.

La routine semop( ) sert à la fois pour les opérations Pn( ) et Vn( ) sur de multiples sémaphores appartenant au jeu indiqué en premier argument. Chaque opération est décrite par une structure sembuf, définie dans <sys/sem.h> ainsi :

|  |  |  |
| --- | --- | --- |
| Nom | Type | Signification |
| sem\_num | short int | Numéro du sémaphore concerné dans l'ensemble. La  numérotation débute à zéro. |
| sem\_op | short int | Valeur numérique correspondant à l'opération à réaliser |
| sem\_flg | short int | Attributs pour l'opération. |

L'opération effectuée est déterminée ainsi :

* Lorsque le champ sem\_op d'une structure sembuf est strictement positif, le noyau incrémente le compteur interne associé au sémaphore de la valeur indiquée et réveille les processus en attente.

Quand sembuf.sem\_op = n, avec n > 0, alors l'opération est Vn().

* Lorsque le champ sem\_op est strictement négatif, le noyau endort le processus jusqu'à ce que le compteur associé au sémaphore soit supérieur à sem\_op, puis il décrémente le compteur de cette valeur avant de continuer l'exécution du processus.

Quand sembuf.sem\_op = n, avec n < 0, alors l'opération est Pn( ).

* Lorsque le champ sem\_op est nul, le noyau endort le processus jusqu'à ce que le compteur associé au sémaphore soit nul, puis il continue l'exécution du programme. Cette fonctionnalité permet de synchroniser les processus.

Il existe deux options possibles pour le membre sem\_flg :

* IPC\_NOWAIT : l'opération ne sera pas bloquante, même si le champ sem\_op est négatif ou nul, mais l'appel-système indiquera l'erreur EAGAIN dans errno si l'opération n'est pas réalisable.
* SEM\_UNDO : pour être sûr que le sémaphore retrouvera un état correct même en cas d'arrêt intempestif du programme, le noyau va mémoriser l'opération inverse de celle qui a été réalisée et l'effectuera automatiquement à la fin du processus.

La routine semop( ) prend en second argument une table de structures sembuf. Le nombre d'éléments dans cette table est indiqué en dernière position. Le noyau garantit que les opérations seront atomiquement liées, ce qui signifie qu'elles seront toutes réalisées ou qu'aucune ne le sera. Il suffit qu'une seule opération avec sem\_op négatif ou nul échoue avec l'attribut IPC\_NOWAIT pour que toutes les modifications soient annulées.

Pour implémenter les fonctions P() et V() définies par Dijkstra, on peut donc employer un ensemble avec un seul sémaphore, qu'on manipulera ainsi :

P (int identifiant)

{

struct sembuf buffer;

buffer.sem\_num = 0;

buffer.sem\_op = -1;

buffer.sem\_flg = IPC\_UNDO;

return (semop (identifiant, & buffer, 1));

}

V (int identifiant)

{

struct sembuf buffer;

buffer.sem\_num = 0;

buffer.sem\_op = 1;

buffer.sem\_flg = IPC\_UNDO;

return (semop (identifiant, & buffer, 1));

}

L'option SEM\_UNDO employée lors d'une opération permet au processus de s'assurer qu'en cas de terminaison impromptue alors qu'il bloque un sémaphore le noyau en restituera l'état initial. Ceci est réalisé en utilisant un compteur par sémaphore et par processus qui a demandé un accès à l'ensemble. Ce mécanisme est donc coûteux en mémoire. Le noyau modifie l'état de ce compteur à chaque opération sur le processus en y inscrivant l'opération inverse. Si par exemple le processus effectue une opération Pn( ), le noyau le bloque jusqu'à ce que le compteur du sémaphore soit supérieur à n, puis il diminue le compteur de cette valeur, et il augmente le compteur d'annulation du sémaphore pour ce processus de la valeur n. Lorsque le processus réalisera Vn( ) le noyau augmentera le compteur du sémaphore et réduira le compteur d'annulation.

Lorsque le processus se termine, le noyau ajoute le compteur d'annulation à celui du sémaphore. Si le processus a bien libéré le sémaphore, le compteur d'annulation est nul, et rien ne se passe. Si par contre le processus s'est terminé après avoir effectué Pn( ), mais sans avoir réalisé Vn( ), le compteur d'annulation vaut +n et le noyau libère ainsi automatiquement le sémaphore. Si le noyau doit décrémenter le compteur du sémaphore lors de la fin d'un processus. L'implémentation actuelle sous Linux consiste à diminuer immédiatement le compteur, mais à limiter ce dernier à zéro. D'autres systèmes peuvent préférer bloquer indéfiniment pour garantir l'annulation de n'importe quelle opération.

L'emploi des sémaphores doit autant que possible être restreint aux opérations P() et V() sur un seul sémaphore à la fois. On limitera également le compteur du sémaphore à la valeur 1. Si le processus risque de bloquer – ou d'être tué – durant la portion critique où il tient un sémaphore, on emploiera l'option SEM\_UNDO. Bien sûr, si on utilise une fois cette option, on prendra la précaution de le faire à chaque opération sur le sémaphore.

La fonction semctl( ) permet de consulter ou de modifier le paramétrage d'un jeu de sémaphore, mais également de fixer l'état du compteur. Cette routine utilise traditionnellement en dernier argument une union définie ainsi :

union semun {

int valeur

struct semid\_ds \* buffer

unsigned short int \* table

}

En fait, cette union n'est pas définie dans les fichiers d'en-tête système, elle doit être déclarée manuellement dans le programme utilisateur. En réalité, le prototype de semctl( ), vu par le compilateur, est en substance le suivant :

int semctl (int identifiant, int numero, int commande, ...) ;

Pour garder une certaine homogénéité aux appels semctl( ), on préfère généralement regrouper les diverses possibilités dans une union, qui permet quand même une vérification minimale.

La structure semid\_ds qui représente le paramétrage d'un jeu de sémaphore contient notamment les membres suivants :

|  |  |  |
| --- | --- | --- |
| Nom | Type | Signification |
| sem\_perm | Struct ipc\_perm | Autorisations d'accès à l'ensemble de sémaphores |
| sem otime | time\_t | Heure de la dernière opération semop( ) |
| sem ctime | time\_t | Heure de la dernière modification de sem\_perm |
| sem\_nsems | unsigned short | Nombre de sémaphores dans l'ensemble |

Lorsqu'un ensemble de sémaphores est créé, les compteurs sont initialement vides. Aucun processus ne peut donc se les attribuer. Il faut donc leur donner une valeur initiale à l'aide de la commande SETVAL.