Annexe : UTILISATION DE LA BIBLIOTHEQUE DE SEMAPHORES LIBIPC

La bibliothèque de sémaphores, appelée `libIPC`, est un outil facilitant l'utilisation des sémaphores au-dessus d'Unix. Elle reprend l'interface classique des sémaphores.

1. LES PRIMITIVES

libIPC contient les primitives suivantes :

```
#include <libipc.h>
```

```
int creer_sem(int nb);
```

Fonction qui crée `nb` sémaphores non initialisés. Cette fonction retourne -1 en cas d'erreur. Les sémaphores sont numérotés à partir de 0. Par exemple l'appel à `creer_sem(3)` va créer 3 sémaphores numérotés 0, 1 et 2.

```
int init_un_sem(int sem, int val);
```

Fonction qui initialise le sémaphore numéro `sem` à la valeur val. Cette fonction retourne -1 en cas d'erreur.

```
void P(int sem);
```

Réalisation de la primitive P sur le sémaphore numéro `sem`.

```
void V(int sem);
```

Réalisation de la primitive V sur le sémaphore numéro `sem`.

```
int det_sem(void);
```

Fonction qui détruit tous les sémaphores. Elle doit impérativement être appelée à la fin du programme. Cette fonction retourne -1 en cas d'erreur.

libIPC permet également de créer des segments de mémoire partagée :

```
char* init_shm(int taille);
```

Fonction qui crée un segment de mémoire partagé de taille `taille`. Elle retourne l'adresse du segment créé. Cette fonction retourne NULL en cas d'erreur.

```
int det_shm(char *seg);
```

Fonction qui détruit le segment désigné par `seg`. Elle doit impérativement être appelée à la fin du programme. Cette fonction retourne -1 en cas d'erreur.

2. COMMANDES SHELL

Il existe quelques commandes Unix permettant d'intervenir sur les sémaphores en cas de problème (par exemple la non destruction des sémaphores) :

`ipcs`	permet d'afficher la liste des sémaphores et segments de mémoire partagée définis sur la machine.
`ipcrm sem id`	permet de détruire le sémaphore identifié par `id`.
`ipcrm shm id`	permet de détruire le segment de mémoire partagée identifié par `id`.

3. UTILISATION DE LA BIBLIOTHEQUE

`libIPC` est définie dans le répertoire

`/Infos/lmd/2023/licence/ue/LU3IN010-2024fev/libipc/libipc/lib`.

Lorsqu'un programme utilise les primitives de libIPC, il faut lier son exécutable à libIPC.

Le répertoire `/Infos/lmd/2023/licence/ue/LU3IN010-2024fev/libipc/libipc/demo/` contient un exemple de programme utilisant libIPC, ainsi que le Makefile permettant de lier son exécutable à libIPC.

Le plus simple est de copier le fichier `Makefile` contenu dans ce répertoire

```
$ cp /Infos/lmd/2024/licence/ue/LU3IN010-2025fev/libipc/libipc/demo/ Makefile .
```

puis de le modifier éventuellement en remplaçant `demo_ipc` par le nom de votre fichier. Vous pouvez alors générer l'exécutable.

```
$ make
```

4. POUR INSTALLER LA LIBRAIRIE CHEZ SOI

La librairie est disponible sous forme d'archive compressée sur le moodle de l'UE
`https://moodle-sciences-24.sorbonne-universite.fr/mod/folder/view.php?id=140727`

Pour la décompresser il suffit de taper sur votre machine :

```
$ tar xvfz libipc.tgz
```

Un répertoire libipc est créé contenant les deux répertoires `demo` et `src`. Il suffit de compiler la librairie :

```
$ cd libipc/src
$ make
```

Vous pouvez alors compiler l'exemple présenté en 5 et l'exécuter :

```
$ cd ../demo
$ make
```

```
$ ./demo-ipc
```

5. EXEMPLE D'UTILISATION

Le programme ci-dessous réalise une barrière à trois processus. Le processus père crée deux fils et les attend à la barrière.

```
#include <libipc.h>
/* Definition des semaphores */

#define SEM1    0
#define SEM2    1

/* Définition du format du segment de memoire partagée */

typedef struct {
     int a;
} t_segpart;

t_segpart *sp; /* Pointeur sur le segment */

/* fonction exécutée par le premier processus fils */
void fils1(void)
{
     sleep(10);
     sp->a++;
     V(SEM1);
     printf("fin1\n");
     exit(0);
}

/* fonction exécutée par le second processus fils */
void fils2(void)
{
     sleep(10);
     sp->a++;
     V(SEM2);
     printf("fin2\n");
     exit(0);
}

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

     /* Creer les semaphores */
     if (creer_sem(2) == -1) {
          perror("creer_sem");
          exit(1);
     }

     /* Initialiser les valeurs */
     init_un_sem(SEM1,0);
     init_un_sem(SEM2,0);

     /* Creer le segment de memoire partagee */
     if ( (sp = (t_segpart *) init_shm(sizeof(t_segpart))) == NULL) {
          perror("init_shm");
          exit(1);
     }
     sp->a = 0;
```

```
    /* Creer le premier processus fils */
    if ((pid = fork()) == -1) {
         perror("fork");
         exit(2);
    }
    if (pid == 0) {
         /* Premier processus fils */
         fils1();
    }

    /* Creer le second processus fils */
    if ((pid = fork()) == -1) {
         perror("fork");
         exit(2);
    }
    if (pid == 0) {
         /* Second processus fils */
         fils2();
    }

    /* Processus Pere */
    printf("le pere attend...\n");
    P(SEM1);
    printf("fin du fils 1\n");
    P(SEM2);
    printf("fin du fils 2\n");
    printf("valeur du compteur = %d\n", sp->a);

    /* Destruction des semaphores et du segment de mémoire */
    det_sem();
    det_shm(sp);

    return EXIT_SUCCESS;
}
```

Pour tout commentaire, ou rapport de « bug » : Pierre.Sens@lip6.fr