STR : TP : Threads, mutex et sémaphores

Chapitre 1 : Mutex

Cette présentation contient des extraits du code, les programmes sources complets ainsi qu'un fichier Makefile pour les compiler peuvent être retrouvés dans l'archive rendu.

Question 1 :

On reprend le programme précédent.
On commence par ajouter un mutex global. Ensuite, on verrouille ce mutex au début de la fonction monThread afin d'y avoir un accès unique, et on le déverrouille à la fin de la

fonction pour permettre une éventuelle réutilisation par d'autres autres threads. Programme:

```c
/* ---------------------------------------------------------------------- */
void  monThread (long  no)
{
    double * retVal = (double *) calloc (1, sizeof(double));
    struct timespec delai = { .tv_sec = 0, .tv_nsec = 0};
    long n = 0;
    double diviseur = 1.;
    const char *p;
    /*  Attente initiale du thread                                        */
    delai.tv_nsec = TEMPO_10ms * (10 + rand() % 31);
    nanosleep(&delai, NULL);

    CHECK_T(pthread_mutex_lock(&mutex), "pthread_mutex_lock()") ;
    for (p = msgPi; *p != '\0'; p++) {
        if (isalpha(*p)) {
            n++;
            if (no == 1)
                printf("%c", tolower(*p));
            else
                printf("%c", toupper(*p));
        } else {
            if (n > 0) {
                *retVal += n / diviseur;
                n = 0;
                diviseur *= 10.;
            }
            printf("%c", *p);
            fflush(stdout);
        }
        /* Attente entre le traitement de chaque caractère                */
        delai.tv_nsec = TEMPO_10ms * (5 + rand() % 16);
        nanosleep(&delai, NULL);
    }
    CHECK_T(pthread_mutex_unlock(&mutex), "pthread_mutex_unlock()");
    /* traitement du dernier mot                                          */
    if (n > 0) *retVal += n / diviseur;

    /* terminaison du thread                                              */
    pthread_exit((void *) retVal);
}

/* ---------------------------------------------------------------------- */
void bye (void)
{
    printf("Fin du processus %d (tid = 0x%lx)\n",
           getpid(), pthread_self());
}
/* ---------------------------------------------------------------------- */
```

- Exécution, de cette manière les deux ressources ont été consultées chacune pendant son temps, les deux messages ont été donc imprimés correctement.

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./mutexThread1
Le processus (pid = 13206, tid = 0x7f6d19f0a740) va créer 2 threads
Attente de la terminaison des 2 threads

QUE J'AIME A FAIRE APPRENDRE UN NOMBRE UTILE AUX SAGESq
Fin du thread n°1 avec le code 3.1415926535
ue j'aime a faire apprendre un nombre utile aux sages
Fin du thread n°2 avec le code 3.1415926535
Fin du processus 13206 (tid = 0x7f6d19f0a740)
```

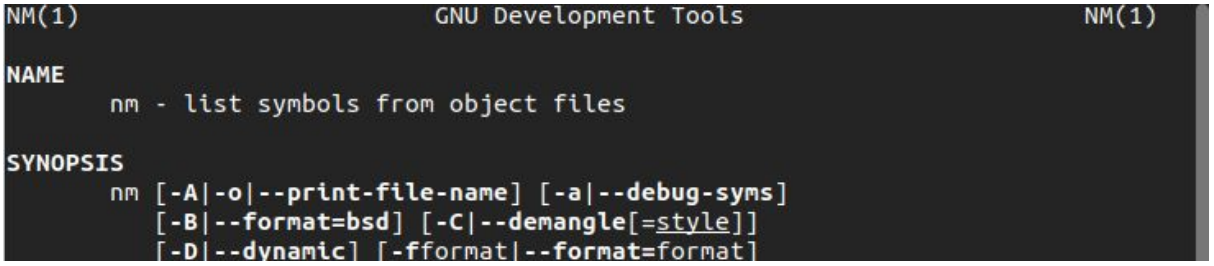
Chapitre 2 : Sémaphores

Question 2 :

a - Dans la commande :

nm -D /lib/x86_64-linux-gnu/libpthread.so.0 | grep sem_

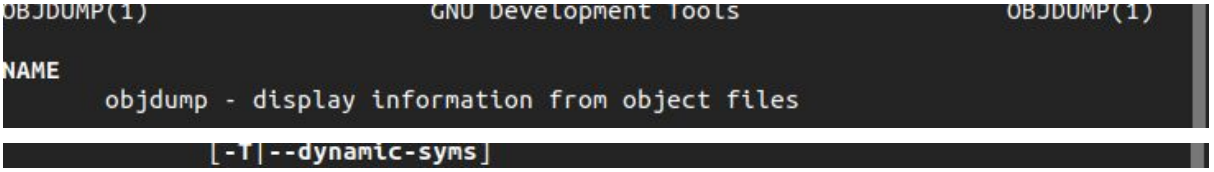
Selon la documentation, la commande ***nm*** permet d'afficher les occurrences d'un pattern d'un symbole inclus dans des fichiers objets, binaires, ou librairie. ***-D*** permet d'afficher les symboles dynamiques.



b - Dans la commande :

objdump -T /lib/x86_64-linux-gnu/libpthread.so.0 | grep sem_

Le rôle de ***objdump*** et d'afficher les informations sur ces fichiers. Le ***-T*** permet d'afficher les symboles dynamiques.



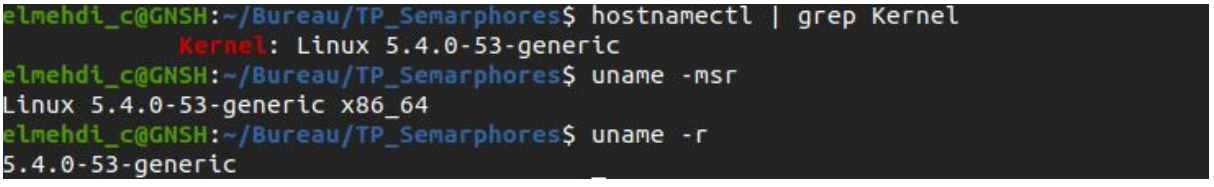
Question 3 :

Afin de d'afficher le nom et la version du noyau, on peut utiliser une des trois commandes suivantes :

hostnamectl | grep Kernel

uname -msr

uname -r



Question 4 :

- On reprend le code de m*utexThread1.c* sous le nom *mutexThread2.c.* On remplace l'usage de mutex par un usage des sémaphores.

```c
    double * status;

    CHECK_S(sem_init(&sema, 0, 1), "sem_init()");

    srand(time(NULL));  /* Init. du générateur de nombres pseudo-aléatoires */
    atexit(bye);
    printf("Le processus (pid = %d, tid = 0x%lx) va créer %d threads\n",
           getpid(), (long)pthread_self(), NBMAX_THREADS);
    /* Création des NBMAX_THREADS                                          */
    for (i = 0; i < NBMAX_THREADS; i++)
        CHECK_T(pthread_create (&tid[i], NULL, (pf_t)monThread,
                                (void *)i), "pthread_create()");

    printf("Attente de la terminaison des %d threads\n\n", NBMAX_THREADS);
    for (i = 0; i < NBMAX_THREADS; i++) {
        CHECK_T(pthread_join (tid[i], (void **) & status),"pthread_join()");
        printf("\nFin du thread n°%ld avec le code %.10f\n", i + 1, *status);
        free(status);
    }

    CHECK_S(sem_destroy(&sema ) ,"sem_destroy()");
    return EXIT_SUCCESS;
}

/* ------------------------------------------------------------------------ */
void  monThread (long  no)
{
    double * retVal = (double *) calloc (1, sizeof(double));
    struct timespec delai = { .tv_sec = 0, .tv_nsec = 0};
    long n = 0;
    double diviseur = 1.;
    const char *p;
    /*  Attente initiale du thread                                          */
    delai.tv_nsec = TEMPO_10ms * (10 + rand() % 31);
    nanosleep(&delai, NULL);

    //Accès à la ressource
    CHECK_S(sem_wait(&sema), "sem_wait()") ;
    for (p = msgPi; *p != '\0'; p++) {
        if (isalpha(*p)) {
            n++;
            if (no == 1)
                printf("%c", tolower(*p));
            else
                printf("%c", toupper(*p));
        } else {
            if (n > 0) {
                *retVal += n / diviseur;
                n = 0;
                diviseur *= 10.;
            }
            printf("%c", *p);
            fflush(stdout);
        }
        /* Attente entre le traitement de chaque caractère                 */
        delai.tv_nsec = TEMPO_10ms * (5 + rand() % 16);
        nanosleep(&delai, NULL);
    }
    CHECK_S(sem_post(&sema), "sem_post()");
    /* traitement du dernier mot                                           */
    if (n > 0) *retVal += n / diviseur;
```

- Exécution :

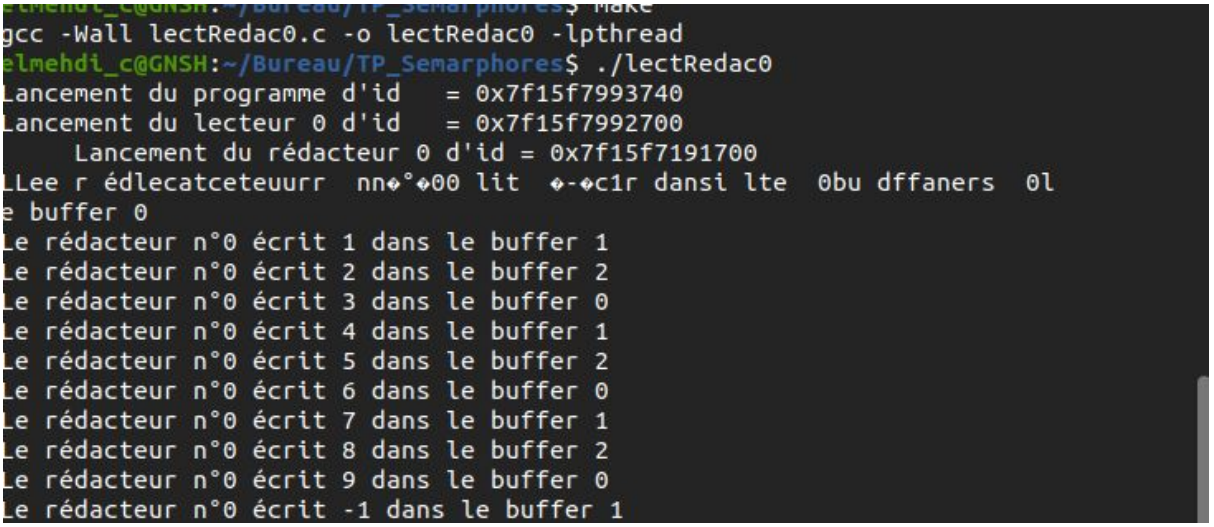
```
elmehdi_c@GNSH:~/Bureau/TP_Semaphores$ ./MutexThread2
Le processus (pid = 20051, tid = 0x7f350b296740) va créer 2 threads
Attente de la terminaison des 2 threads

QUE J'AIME A FAIRE APPRENDRE UN NOMBRE UTILE AUX SAGESq
Fin du thread n°1 avec le code 3.1415926535
ue j'aime a faire apprendre un nombre utile aux sages
Fin du thread n°2 avec le code 3.1415926535
Fin du processus 20051 (tid = 0x7f350b296740)
elmehdi_c@GNSH:~/Bureau/TP_Semaphores$
```

Chapitre 3 : Exercice de synthèse

Question 5 :

En compilant et en exécutant *lectRedac0,* on obtient :



On remarque qu'un message du lecteur est affiché mais celui-ci n'est pas correctement généré, alors que les messages des rédacteurs sont tous lisibles.

Question 6 :

1 - Création du mutex (global) mutexEcran :

pthread_mutex_t mutex

2 - Initiation dans la fonction main() :

CHECK_T(pthread_mutex_init(&mutexEcran, NULL), "pthread_mutex_init");

3 - Utilisation de mutexEcran dans putLine() :

```
void putline(const char *s)
{
    CHECK_T(pthread_mutex_lock(&mutexEcran),        //Verrouillage de mutexEcran pour l'utilisation
    //contenu de la fonction putLine()//

    CHECK_T(pthread_mutex_unlock(&mutexEcran),      //Déverrouillage de mutexEcran
    "pthread_mutex_unlock()") ;
}
```

4 - Finalement, on supprime celui-ci après terminaison (pthread_join()) dans la fonction main():

CHECK_T(pthread_mutex_destroy(&mutexEcran),"pthread_mutex_destroy()") ;

Le programme *lectRedac1-1.c* ainsi que les autres fichiers sources peuvent être trouvés dans l'archive joint.

Question 7 :

En exécutant le programme, on remarque que le lecteur 0 lit une valeur avant qu'aucun rédacteur ait écrit. Ce lecteur 0 n'effectue donc plus aucune action à cause de la valeur -1 lue :

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ make
gcc -Wall lectRedac1-1.c -o lectRedac1-1 -lpthread
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-1
Lancement du programme d'id = 0x7f0b7d208740
Lancement du lecteur 0 d'id   = 0x7f0b7d207700
     Lancement du rédacteur 0 d'id = 0x7f0b7ca06700
Le lecteur n°0 lit -1 dans le buffer 0
Le rédacteur n°0 écrit 0 dans le buffer 0
Le rédacteur n°0 écrit 1 dans le buffer 1
Le rédacteur n°0 écrit 2 dans le buffer 2
Le rédacteur n°0 écrit 3 dans le buffer 0
Le rédacteur n°0 écrit 4 dans le buffer 1
Le rédacteur n°0 écrit 5 dans le buffer 2
Le rédacteur n°0 écrit 6 dans le buffer 0
Le rédacteur n°0 écrit 7 dans le buffer 1
Le rédacteur n°0 écrit 8 dans le buffer 2
Le rédacteur n°0 écrit 9 dans le buffer 0
Le rédacteur n°0 écrit -1 dans le buffer 1
```

Question 8 :

a - *-DNOREADER* permet de définir une constante symbolique qui interdit la créature des threads lecteurs.

b - On exécute cette fois le programme *gcc* avec l'option *-DNOREADER* à la compilation. On remarque que les rédacteurs écrivent toujours dans des variables non vides.

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ make
gcc -DNOREADER -Wall lectRedac1-1.c -o lectRedac1-1_DNO -lpthread
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-1_DNO
Lancement du programme d'id = 0x7f01c3c27740
Lancement du rédacteur 0 d'id = 0x7f01c3c26700
Le rédacteur n°0 écrit 0 dans le buffer 0
Le rédacteur n°0 écrit 1 dans le buffer 1
Le rédacteur n°0 écrit 2 dans le buffer 2
Le rédacteur n°0 écrit 3 dans le buffer 0
Le rédacteur n°0 écrit 4 dans le buffer 1
Le rédacteur n°0 écrit 5 dans le buffer 2
Le rédacteur n°0 écrit 6 dans le buffer 0
Le rédacteur n°0 écrit 7 dans le buffer 1
Le rédacteur n°0 écrit 8 dans le buffer 2
Le rédacteur n°0 écrit 9 dans le buffer 0
Le rédacteur n°0 écrit -1 dans le buffer 1
```

Question 9 :

a - *-NOWRITER* permet de définir une constante symbolique qui interdit la créature des threads rédacteurs.

b - On exécute cette fois le programme avec l'option *-NOWRITER* à la compilation *gcc.*
On a le résultat suivant :

```
lectRedac1-1_DNO        lectRedac1-1_DNOWrite
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-1_DNOWrite
Lancement du programme d'id = 0x7fc69e941740
Lancement du lecteur 0 d'id   = 0x7fc69e940700
     Le lecteur n°0 lit -1 dans le buffer 0
```

Question 10 :

Une fonction d'initialisation init() est :

```
void init(circBuffer_t * b) {  //Pour un pointeur vers un buffer circulaire b, on initie deux sémaphores, un vide et un plein
    int i ;
    for (i = 0 ; i < NBMAX_BUFFER ; i++)
        b->buffer[i] = OVER ;
    CHECK(sem_init(&buf->semBufferVide ,
    0,NBMAX_BUFFER),"sem_init(semVide)") ;
    CHECK(sem_init(&buf->semBufferPlein , 0, 0 ), "sem_init(semPlein)") ; }
```

Question 11 :

On crée : buf et on l'initie(buf);
Après l'usage des deux sémaphores, on les supprimes après la terminaison des threads à la fin de la fonction main() :

```
CHECK( sem_destroy (&buf->semBufferPlein ), " sem_destroy ( semPlein )") ;
CHECK( sem_destroy (&buf->semBufferVide ), " sem_destroy ( semVide )") ;
```

Question 12 :

Au niveau du rédacteur, on attend que le buffer soit vide avant de l'utiliser on utilise ensuite *sem_post()* pour indiquer que le buffer est plein et suspendre la lecture.

```
CHECK (sem_wait (&buf. semBufferVide ), " sem_wait ( semVide )");
//////lecture par le rédacteur n ////
CHECK (sem_post (&buf. semBufferPlein ), " sem_post ( semPlein )") ;
```

La nouvelle fonction redacteur devient :

```
void  redacteur(long no)
{
    int i = 0, n = 0;
    char line[MAXLEN + 1];

    printf("Lancement du rédacteur %ld d'id = 0x%08lx\n",
           no, (long)pthread_self());

    while  ( n != OVER ) {
        CHECK(sem_wait (&buf. semBufferVide), " sem_wait ( semVide )");
        if (n == MAX_VALUE)
            buf.buffer[i] = n = OVER;
        else
            buf.buffer[i] = n++;
        sprintf(line,"Le rédacteur n°%ld écrit %d dans le buffer %d\n",
                no, buf.buffer[i] , i);
        putLine(line);
        CHECK (sem_post (&buf. semBufferPlein ), " sem_post ( semPlein )") ;
        i++;
        if (i == NBMAX_BUFFER) i = 0;
    }
}
```

Question 13 :

Pour chaque rédacteur, on prend en compte les modifications au niveau du rédacteur avant de pouvoir le lire un compartiment. Une fois lu, on utilise un *sem_post()* pour indiquer qu'il est vide pour qu'on puisse y écrire.

La nouvelle fonction lecteur devient :

```
void  lecteur(long no)
{
    int i = 0, n = 0;
    char line[MAXLEN + 1];

    printf("Lancement du lecteur %ld d'id   = 0x%08lx\n",
           no, (long)pthread_self());
    while ( n != OVER ) {
        CHECK(sem_wait (&buf.semBufferPlein), "sem_wait(semPlein)") ;
        n = buf.buffer[i];
        sprintf(line,"%*sLe lecteur n°%ld lit %d dans le buffer %d\n",
                5 * (int)(no + 1), "    ", no,   n, i);
        putLine(line);
        CHECK(sem_post(&buf.semBufferVide), "sem_post(semVide)") ;
        i++;
        if (i == NBMAX_BUFFER) i = 0;
    }
}
```

Question 14 :

Le programme précédent, appelé *lectRedac1-2.c*. On le compile.

A l'exécution, on obtient :

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-2
Lancement du programme d'id = 0x7f7e60f0e740
Lancement du lecteur 0 d'id   = 0x7f7e60f0d700
Lancement du rédacteur 0 d'id = 0x7f7e6070c700
Le rédacteur n°0 écrit 0 dans le buffer 0
Le rédacteur n°0 écrit 1 dans le buffer 1
     Le lecteur n°0 lit 0 dans le buffer 0
     Le lecteur n°0 lit 1 dans le buffer 1
Le rédacteur n°0 écrit 2 dans le buffer 2
Le rédacteur n°0 écrit 3 dans le buffer 0
Le rédacteur n°0 écrit 4 dans le buffer 1
     Le lecteur n°0 lit 2 dans le buffer 2
     Le lecteur n°0 lit 3 dans le buffer 0
     Le lecteur n°0 lit 4 dans le buffer 1
Le rédacteur n°0 écrit 5 dans le buffer 2
Le rédacteur n°0 écrit 6 dans le buffer 0
Le rédacteur n°0 écrit 7 dans le buffer 1
     Le lecteur n°0 lit 5 dans le buffer 2
     Le lecteur n°0 lit 6 dans le buffer 0
     Le lecteur n°0 lit 7 dans le buffer 1
Le rédacteur n°0 écrit 8 dans le buffer 2
Le rédacteur n°0 écrit 9 dans le buffer 0
Le rédacteur n°0 écrit -1 dans le buffer 1
     Le lecteur n°0 lit 8 dans le buffer 2
     Le lecteur n°0 lit 9 dans le buffer 0
     Le lecteur n°0 lit -1 dans le buffer 1
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-2
```

On remarque que le rédacteur peut écrire plusieurs fois quand le compartiment est vide. 10 valeurs ont été écrites puis lues par le lecteur.

Question 15 :

On exécute cette fois le programme gcc avec l'option *-DNOREADER* à la compilation *gcc.*
On remarque que les rédacteurs écrivent toujours dans des variables non vides.

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-2
lectRedac1-2           lectRedac1-2_DNORead   lectRedac1-2_DNOWrite
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-2_DNORead
Lancement du programme d'id = 0x7f0628c04740
Lancement du rédacteur 0 d'id = 0x7f0628c03700
Le rédacteur n°0 écrit 0 dans le buffer 0
Le rédacteur n°0 écrit 1 dans le buffer 1
Le rédacteur n°0 écrit 2 dans le buffer 2
^C
```

Le lecteur n'ayant pas été créé dans cette version, le rédacteur n'écrit pas tant que le compartiment est plein. On quitte avec un ctr+C.

Question 16 :

On exécute cette fois le programme gcc avec l'option *-DNOWRITER* à la compilation *gcc.*
On remarque que les rédacteurs écrivent toujours dans des variables non vides.

```
elmehdi_c@GNSH:~/Bureau/TP_Semarphores$ ./lectRedac1-2_DNOWrite
Lancement du programme d'id = 0x7f4eb5b19740
Lancement du lecteur 0 d'id   = 0x7f4eb5b18700
^C
```

De la même manière, on interrompt le programme. Celui-ci est figé car le lecteur ne lit aucune valeur puisque le thread rédacteur n'a pas été créé à la compilation.

Question 17 :

Voir le fichier source lectRedac2.C disponible dans l'archive.