Timothé Belcour
TP1

Synchronisation de Threads avec les Sémaphores POSIX

Exercice 1 :

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#define N 1000000
int compteur = 0;
void * incr(void * a)
{
    int i, tmp;
    for(i = 0; i < N; i++)
    {
    tmp = compteur; tmp = tmp+1; compteur = tmp;
    }
    pthread_exit(NULL);
}
int main(int argc, char * argv[]){

    pthread_t tid1, tid2;
    if(pthread_create(&tid1, NULL, incr, NULL))
    {
        printf("\n ERREUR création thread 1");
        exit(1);
    }
    if(pthread_create(&tid2, NULL, incr, NULL))
    {
        printf("\n ERROR création thread 2");
        exit(1);
    }
    if(pthread_join(tid1, NULL)) {
        printf("\n ERREUR thread 1 ");
        exit(1);
    }
    if(pthread_join(tid2, NULL)) {
        printf("\n ERREUR thread 2");
        exit(1);
    }
    if ( compteur < 2 * N)
        printf("\n BOOM! compteur = [%d], devrait être %d\n", compteur, 2*N);
    else
        printf("\n OK! compteur = [%d]\n", compteur);
        pthread_exit(NULL);
    return 0 ;
}
```

Timothé Belcour
TP1

```
timothebelcour@mac semposix % gcc compt.c -o compt -lpthread
./compt; ./compt; ./compt
gcc compt_mutex.c -o compt_mutex -lpthread
./compt_mutex

  BOOM! compteur = [1002889], devrait être 2000000

  BOOM! compteur = [1002503], devrait être 2000000

  BOOM! compteur = [1000693], devrait être 2000000
```

Le programme lance deux threads qui essaient d'augmenter
la même variable compteur sans se coordonner.
Chacun fait : lire la valeur, ajouter 1, réécrire.
Comme ces trois étapes ne sont pas faites d'un seul bloc, l'autre thread peut s'intercaler entre-temps.
Exemple : les deux lisent 100, chacun calcule 101 ; le premier écrit 101, puis le second réécrit à nouveau 101.
On a fait deux incréments, mais la valeur n'a augmenté qu'une fois donc des incréments sont perdus.
C'est pour cela que le résultat change à chaque exécution et reste en dessous de 2×N (messages "BOOM!").

Timothé Belcour
TP1

Exercice 2 :

```c
#include <pthread.h>
#include <semaphore.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define N 1000000

int compteur = 0;
sem_t *mutex = NULL;
void* incr(void* a) {
    for (int i = 0; i < N; i++) {
        sem_wait(mutex);
        int tmp = compteur;
        tmp = tmp + 1;
        compteur = tmp;
        sem_post(mutex);
    }
    return NULL;
}
int main(void) {
    pthread_t tid1, tid2;
    char name[64];
    snprintf(name, sizeof(name), "/compteur_mutex_%d", getpid());
    sem_unlink(name);
    mutex = sem_open(name, O_CREAT | O_EXCL, 0644, 1);
    if (mutex == SEM_FAILED) {
        perror("sem_open");
        return 1;
    }
    if (pthread_create(&tid1, NULL, incr, NULL) != 0) { perror("pthread_create tid1"); return 1; }
    if (pthread_create(&tid2, NULL, incr, NULL) != 0) { perror("pthread_create tid2"); return 1; }
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    if (compteur < 2 * N)
        printf("\nBOOM! compteur = [%d], devrait être %d\n", compteur, 2 * N);
    else
        printf("\nOK! compteur = [%d]\n", compteur);
    sem_close(mutex);
    sem_unlink(name);
    return 0;
}
```

```
timothebelcour@mac semposix % ./compt_mutex

OK! compteur = [2000000]
timothebelcour@mac semposix %
```

On place la section critique (read–modify–write sur compteur)
entre sem_wait et sem_post d'un sémaphore POSIX, ce qui impose l'exclusion mutuelle : un seul thread à la fois modifie compteur. Dès lors, aucun incrément n'est perdu et la sortie devient déterministe, toujours égale à 2×N ("OK! compteur = 2000000").

Pour conclure
./compt : résultats variables et < 2×N (BOOM) = attendu (course critique).
./compt_mutex : toujours 2×N (OK) = attendu (section critique protégée).