Adrien BOURGET – OS / LAB3

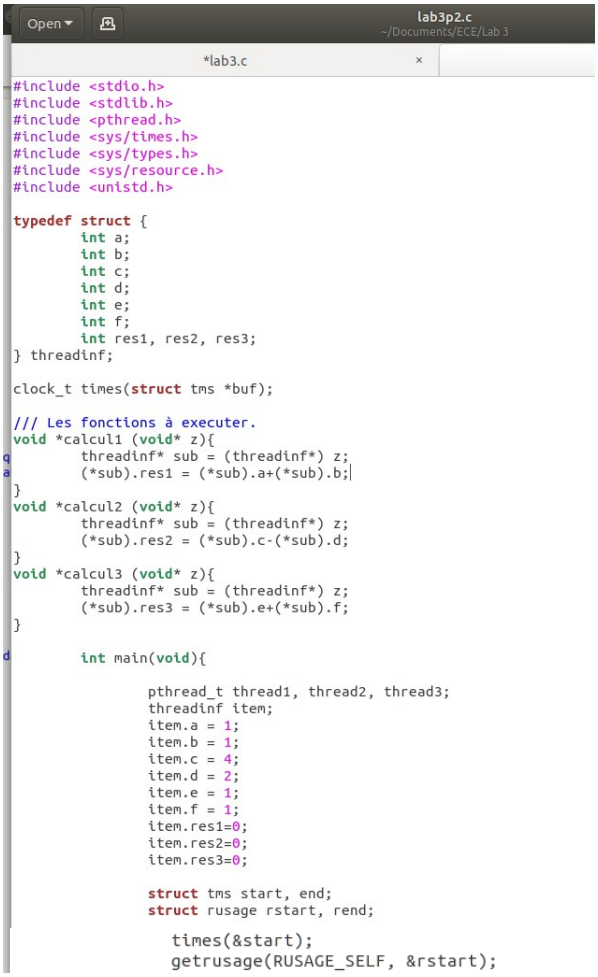
Lab 3 – Threads

Processes v. Threads.

1. With processes

```c
clock_t times(struct tms *buf);

int main()
{
        int id1, id2, *ptr1, *ptr2;
        int a=1, b=1, c=4, d=2, e=1, f=1;
        int i;
        struct tms start, end;
        struct rusage rstart, rend;
        // Initialisation du temps
        times(&start);
        getrusage(RUSAGE_SELF, &rstart);

        id1 = shmget(KEY1, sizeof(int), IPC_CREAT | PERMS);
        id2 = shmget(KEY2, sizeof(int), IPC_CREAT | PERMS);
        ptr1 = (int *) shmat(id1, NULL, 0);
        ptr2 = (int *) shmat(id2, NULL, 0);

        for(i=0; i<500; i++)
        {
                //1er fils
                if(fork()==0)
                {
                        *ptr1 = (a+b);
                        exit(0);
                }

                //parent
                else
                {
                        //second fils
                        if(fork()==0){
                                *ptr2 = (c-d);
                                exit(0);
                        }
                        //parent
                        else
                        {
                                int var = (e+f);
                                wait(NULL);
                                int res = var + ((*ptr1)*(*ptr2));
                                printf("(a+b)*(c-d)+(e+f) = %d\n", res);
                        }
                }
        }
        }
        times(&end); // Temps à la fin du process
        getrusage(RUSAGE_SELF, &rend);

        //Affichage du temps
        printf("%lf usec\n", (end.tms_utime+end.tms_stime-start.tms_utime-start.tms_stime)*1000000.0/
sysconf(_SC_CLK_TCK));

        // Affichage du contexte switch
        printf(" Context switches: %ld\n", rend.ru_nvcsw-rstart.ru_nvcsw+rend.ru_nivcsw-
rstart.ru_nivcsw);

        // Affichage des input / ouput
        printf("Imput= %ld ; Output = %ld\n", rend.ru_inblock, rend.ru_oublock);
        shmctl(id1, IPC_RMID, NULL);
        shmctl(id2, IPC_RMID, NULL);

        return 0;
}
```

2. With threads




```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/times.h>
#include <sys/types.h>
#include <sys/resource.h>
#include <unistd.h>

typedef struct {
        int a;
        int b;
        int c;
        int d;
        int e;
        int f;
        int res1, res2, res3;
} threadinf;

clock_t times(struct tms *buf);

/// Les fonctions à executer.
void *calcul1 (void* z){
        threadinf* sub = (threadinf*) z;
        (*sub).res1 = (*sub).a+(*sub).b;
}
void *calcul2 (void* z){
        threadinf* sub = (threadinf*) z;
        (*sub).res2 = (*sub).c-(*sub).d;
}
void *calcul3 (void* z){
        threadinf* sub = (threadinf*) z;
        (*sub).res3 = (*sub).e+(*sub).f;
}

        int main(void){

                pthread_t thread1, thread2, thread3;
                threadinf item;
                item.a = 1;
                item.b = 1;
                item.c = 4;
                item.d = 2;
                item.e = 1;
                item.f = 1;
                item.res1=0;
                item.res2=0;
                item.res3=0;

                struct tms start, end;
                struct rusage rstart, rend;

                  times(&start);
                  getrusage(RUSAGE_SELF, &rstart);

                  for(int i=0;i<500;i++){
                          pthread_create(&thread3, NULL, calcul3, (void*) &item);
                          pthread_create(&thread2, NULL, calcul2, (void*) &item);
                          pthread_create(&thread1, NULL, calcul1, (void*) &item);

                          pthread_join(thread3, NULL);
                          pthread_join(thread2, NULL);
                          pthread_join(thread1, NULL);

                          printf("%d", (item.res1*item.res2)+item.res3);
                  }

                  times(&end);
                  getrusage(RUSAGE_SELF, &rend);

        //Affichage du temps
        printf("\n %lf usec\n", (end.tms_utime+end.tms_stime-start.tms_utime-
start.tms_stime)*1000000.0/sysconf(_SC_CLK_TCK));

        // Affichage du contexte switch
        printf(" Context switches: %ld\n", rend.ru_nvcsw-rstart.ru_nvcsw+rend.ru_nivcsw-
rstart.ru_nivcsw);

        // Affichage des input / ouput
        printf("Imput= %ld ; Output = %ld\n", rend.ru_inblock, rend.ru_oublock);
}
```

Part 1. Time

```
struct tms {
    clock_t tms_utime;  /* user time */
    clock_t tms_stime;  /* system time */
    clock_t tms_cutime; /* user time of children */
    clock_t tms_cstime; /* system time of children */
};
```

On mesure la performance de nos deux solutions grâce à la fonction times() (man 2 times) donnée dans l'énnoncé.

L'utilisation de cette fonction nécéssite l'import de la biliothèque <sys/time.h>

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

'user time' est le temps que le CPU prend pour les instructions de l'utilisateur du processus appelant.

'system time' est le temps que le CPU a pris pour l'exécution du programme, la difference initial/final pour ces deux valeurs donne le temps total d'utilisation du CPU.

<!> Cette fonction différencie les durées parent et fils

Part 2. Getrusage

```
struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    long   ru_maxrss;        /* maximum resident set size */
    long   ru_ixrss;         /* integral shared memory size */
    long   ru_idrss;         /* integral unshared data size */
    long   ru_isrss;         /* integral unshared stack size */
    long   ru_minflt;        /* page reclaims (soft page faults) */
    long   ru_majflt;        /* page faults (hard page faults) */
    long   ru_nswap;         /* swaps */
    long   ru_inblock;       /* block input operations */
    long   ru_oublock;       /* block output operations */
    long   ru_msgsnd;        /* IPC messages sent */
    long   ru_msgrcv;        /* IPC messages received */
    long   ru_nsignals;      /* signals received */
    long   ru_nvcsw;         /* voluntary context switches */
    long   ru_nivcsw;        /* involuntary context switches */
};
```

La fonction Getrusage utilise la structure rusage qui contient les infos sur les pricessus appelant.

Les premières variables nous permettent de réaliser les mesures éffectuées dans la partie 1. Mais nous devons différencier parents et enfant.

ru_inblock et ru_outlbock nous informent sur le nombre d'entrées et sorties du programme.
ru_nvcsw & ru_nivcsw informent sur les content switches.

Getrusage() permet de recevoir ces informations et les envoie dans une instance de rusage. Elle revoit également en paramètres les options RUSAGE_SELF or RUSAGE_CHILDREN selon les informations que l'on souhaite obtenir (processus appelant ou processus appellés (fils)).

```
#include <sys/time.h>
#include <sys/resource.h>

int getrusage(int who, struct rusage *usage);
```

1. La difference entre Threads et Processes.

Un seul et même processus peut contenir plusieurs threads. Ces derniers vont s'executer en parallèle. Comme avec la fonction fork(), ils vont s'executer de manière indépendante (le plus petit en premier).

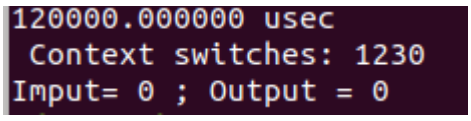
Les threads ont l'avantage de permettre le partage d'informations entre tous mes threads d'un même processus. Ils partagent le même espace mémoire.

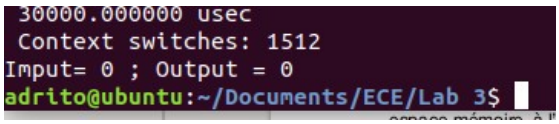
Un processus fonctionne grâce à des copies de données présentes dans le processus appelant tandis que les threads peuvent les modifier directement via le partage des informations.

La comparaison effectuée dans la partie 1 nous montre clairement que les threads sont plus simple à utiliser.

Pour conclure, on peut dire que les threads sont optimisés pour les opérations nécéssitant une communication de données alors que les processus sont eux mieux isolés.

2. Performances.





Ici on compare la fonction fork() avec l'utilisation des threads. On constate une augmentation de switch de ~300. Les threads sont de ce point de vue bien plus optimisés.
Aussi, l'exécution des 500 calculs mets pres de 4 fois moins de temps avec l'utilisation des threads.

3- I/0 et switch.

Mon code permettant l'affichages des input/outputs ne fonctionne pas.

Les switchs quand à eux sont bels et bien affichés. On constate une diminution de moitié en changent de méthode.