Lab3: Threads

Subject

- Running a file : `gcc -pthread -o thread thread.c`
- Then use `./execName` to display results.

Processes VS Threads

1- Evaluate the following expression using the same number of tasks : `(a+b)-[(c*d)/(e-f)] + (g+h)`

- with process
- with threads

1. Measure the performance of both solutions using the `times` function `(man 2 times)`

The first solution using threads has been benchmarked using the time function. It takes approximately 4 seconds to run on my computer. However i was almost at the maximum number of threads possible. Here is the function :

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct {
  int a, b;
  int res;
} thread_data;


void *add(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a + data->b;
  pthread_exit(NULL);
}

void *sub(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a - data->b;
  pthread_exit(NULL);
}

void *mul(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
```

```
  data->res = data->a * data->b;
  pthread_exit(NULL);
}

void *divi(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a / data->b;
  pthread_exit(NULL);
}

int main()
{
    pthread_t thread[8]; //Different threads
    thread_data data[8]; //data to calculate in threads
    int res; //results
    int j = 0;
    int iterator = 0;
    time_t seconds;

    for(j=0;j<8;j++)
    {
      data[j].a=0;
      data[j].b=0;
      data[j].res=0;

    }
    data[0].a = 1;
    data[0].b = 2;
    data[1].a = 3;
    data[1].b = 4;
    data[2].a = 6;
    data[2].b = 5;
    data[3].a = 7;
    data[3].b = 8;

    seconds = time(NULL);
    //Creation of independant thread
    for(iterator=0 ; iterator < 65000 ; iterator++)
    {
      pthread_create( &(thread[0]), NULL, add, &data[0]);
      pthread_create( &(thread[1]), NULL, mul, &data[1]);
      pthread_create( &(thread[2]), NULL, sub, &data[2]);
      pthread_create( &(thread[3]), NULL, add, &data[3]);
      data[4].a = data[1].res;
      data[4].b = data[2].res;
      pthread_create( &(thread[4]), NULL, divi, &data[4]);
      data[5].a = data[0].res;
      data[5].b = data[4].res;
      pthread_create( &(thread[5]), NULL, sub, &data[5]);

      res = data[5].res + data[3].res;

      //Joining threads
```

```
      pthread_join(thread[0], NULL);
      pthread_join(thread[1], NULL);
      pthread_join(thread[2], NULL);
      pthread_join(thread[3], NULL);
      pthread_join(thread[4], NULL);
      pthread_join(thread[5], NULL);
    }

    seconds -= time(NULL);

    printf("Temps %ld", seconds);
    return 0;
}
```

The first solution using process has been benchmarked using the time function. It takes approximately 56 seconds to run on my computer. Here is the program :

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <time.h>
#define KEY 4567
#define PERMS 0660

int main(int argc, char **argv)
{
    time_t seconds;
    int iterator = 0;
    int id, id2, id3, id4;
    int *ptr, *ptr2, *ptr3, *ptr4;
    int total;
    int a = 1, b = 2, c = 3, d = 4, e = 6, f = 5, g = 7, h = 8;
    seconds = time(NULL);

    id = shmget(KEY, sizeof(int), IPC_CREAT | PERMS); //Create an IPC of 4
bytes (=sizeof int) with shared memory (=allocate shared memory) (id)
    id2 = shmget(KEY+1, sizeof(int), IPC_CREAT | PERMS); //We change the
key to avoid having a conflict with ptr
    id3 = shmget(KEY+2, sizeof(int), IPC_CREAT | PERMS);
    id4 = shmget(KEY+3, sizeof(int), IPC_CREAT | PERMS);



    ptr = (int *) shmat(id, NULL, 0); //Attach an existing shared memory
(id) to an adress space (ptr)
    ptr2 = (int *) shmat(id2, NULL, 0);
    ptr3 = (int *) shmat(id3, NULL, 0);
```

```
    ptr4 = (int *) shmat(id4, NULL, 0);

    for(iterator = 0 ; iterator < 65000 ; iterator++)
    {
    if (fork() == 0) //Creating a child process
    {//Parent
        if (fork() == 0)
        {
            (*ptr) = a + b;
            exit(0);
        }
        else
        {
            wait(NULL);
            (*ptr2) = c * d;
        }

        (*ptr3) = e - f;
        exit(0); //Close the parent process
    }
    else
    { //Child
        wait(NULL); //Wait until the parent process finish
        if (fork() == 0)
        {
            (*ptr4) = g + h;
            exit(0);
        }
        else
        {
            wait(NULL);
            (*ptr2) = (*ptr2) / (*ptr3);
        }
        (*ptr) = (*ptr) - (*ptr2);

    }
    total = (*ptr) + (*ptr3);
    }
    seconds -= time(NULL);
    printf("%ld",seconds);
}
```

In both programs my implementation was : each thread/process perform one operation (like a + b) until the calculation is complete. In order to increase the difference in time between threads and processes I added a loop of 65 000 iterations of that calcul.

The difference between threads and processes are that threads run in a shared memory space while processes run in separate memory space. Sources There are a lot more differences that i found here, too much to be all displayed.

2. Display the number of I/O and context switches using the `getrusage` function `(man 2 getrusage)`

In order to evaluate the efficiency of both programs i'm now using the function `getrusage` as we can see in my program below, testing it on threads:

```
#define __USE_GNU
#include <sys/times.h>
#include <sys/resource.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int a, b;
  int res;
} thread_data;

clock_t times(struct tms *buf);

void *add(void *received_struct)
{
    thread_data *data = (thread_data*) received_struct;
    data->res = data->a + data->b;
    pthread_exit(NULL);
}

void *sub(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a - data->b;
  pthread_exit(NULL);
}

void *mul(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a * data->b;
  pthread_exit(NULL);
}

void *divi(void *received_struct)
{
  thread_data *data = (thread_data*) received_struct;
  data->res = data->a / data->b;
  pthread_exit(NULL);
}

int main()
{
    struct tms start, end;
  struct rusage rstart, rend;
  times(&start);
```

```
  getrusage(RUSAGE_SELF, &rstart);
    pthread_t thread[8]; //Different threads
    thread_data data[8]; //data to calculate in threads
    int res; //results
    int j = 0;
    int iterator = 0;

    for(j=0;j<8;j++)
    {
      data[j].a=0;
      data[j].b=0;
      data[j].res=0;

    }
    data[0].a = 1;
    data[0].b = 2;
    data[1].a = 3;
    data[1].b = 4;
    data[2].a = 6;
    data[2].b = 5;
    data[3].a = 7;
    data[3].b = 8;

    //Creation of independant thread
      pthread_create( &(thread[0]), NULL, add, &data[0]);
      pthread_create( &(thread[1]), NULL, mul, &data[1]);
      pthread_create( &(thread[2]), NULL, sub, &data[2]);
      pthread_create( &(thread[3]), NULL, add, &data[3]);
      data[4].a = data[1].res;
      data[4].b = data[2].res;
      pthread_create( &(thread[4]), NULL, divi, &data[4]);
      data[5].a = data[0].res;
      data[5].b = data[4].res;
      pthread_create( &(thread[5]), NULL, sub, &data[5]);


      //Joining threads
      pthread_join(thread[0], NULL);
      pthread_join(thread[1], NULL);
      pthread_join(thread[2], NULL);
      pthread_join(thread[3], NULL);
      pthread_join(thread[4], NULL);
      pthread_join(thread[5], NULL);
      res = data[5].res + data[3].res;
      times(&end);
  getrusage(RUSAGE_SELF, &rend);

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

  printf("%ld usec\n", (rend.ru_utime.tv_sec-
rstart.ru_utime.tv_sec)*1000000 +(rend.ru_utime.tv_usec-
rstart.ru_utime.tv_usec)+(rend.ru_stime.tv_sec-
rstart.ru_stime.tv_sec)*1000000 +(rend.ru_stime.tv_usec-
```

```
rstart.ru_stime.tv_usec));

    return 0;
}
```

The function returned 2253 usec. I'm now testing the same on the processes version :

```
//#define __USE_GNU
#include <sys/times.h>
#include <sys/resource.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#define KEY 4567
#define PERMS 0660

clock_t times(struct tms *buf);

int main(int argc, char **argv)
{
    struct tms start, end;
    struct rusage rstart, rend;
    int iterator = 0;
    int id, id2, id3, id4;
    int *ptr, *ptr2, *ptr3, *ptr4;
    int total;
    int a = 1, b = 2, c = 3, d = 4, e = 6, f = 5, g = 7, h = 8;

    times(&start);
    getrusage(RUSAGE_SELF, &rstart);
    id = shmget(KEY, sizeof(int), IPC_CREAT | PERMS); //Create an IPC of 4
bytes (=sizeof int) with shared memory (=allocate shared memory) (id)
    id2 = shmget(KEY+1, sizeof(int), IPC_CREAT | PERMS); //We change the
key to avoid having a conflict with ptr
    id3 = shmget(KEY+2, sizeof(int), IPC_CREAT | PERMS);
    id4 = shmget(KEY+3, sizeof(int), IPC_CREAT | PERMS);


    ptr = (int *) shmat(id, NULL, 0); //Attach an existing shared memory
(id) to an adress space (ptr)
    ptr2 = (int *) shmat(id2, NULL, 0);
    ptr3 = (int *) shmat(id3, NULL, 0);
    ptr4 = (int *) shmat(id4, NULL, 0);
```

```
    if (fork() == 0) //Creating a child process
    {//Parent

        if (fork() == 0)
        {
            (*ptr) = a + b;
            exit(0);
        }
        else
        {
            wait(NULL);
            (*ptr2) = c * d;
        }
        (*ptr3) = e - f;
        exit(0); //Close the parent process
    }
    else
    { //Child
        wait(NULL); //Wait until the parent process finish
        if (fork() == 0)
        {
            (*ptr4) = g + h;
            exit(0);
        }
        else
        {
            wait(NULL);
            (*ptr2) = (*ptr2) / (*ptr3);
        }
        (*ptr) = (*ptr) - (*ptr2);

    }
    total = (*ptr) + (*ptr3);
    times(&end);
    getrusage(RUSAGE_SELF, &rend);
    printf("%lf usec\n", (end.tms_utime+end.tms_stime-start.tms_utime-
start.tms_stime)*1000000.0/sysconf(_SC_CLK_TCK));

  printf("%ld usec\n", (rend.ru_utime.tv_sec-
rstart.ru_utime.tv_sec)*1000000 +(rend.ru_utime.tv_usec-
rstart.ru_utime.tv_usec)+(rend.ru_stime.tv_sec-
rstart.ru_stime.tv_sec)*1000000 +(rend.ru_stime.tv_usec-
rstart.ru_stime.tv_usec));

    return 0;
}
```

This time i obtain 456 usec. At first sight we could believe that processes are faster but i suspect this function to not count the other processes. In order to find the same results as the first test i tried `getrusage(RUSAGE_CHILDREN, &rend);` in the processes program but it kept giving me negative results(aproximately -1500). I also tried `getrusage(RUSAGE_THREAD, &rstart);` but it gave me some

high result (aproximately 4200) much higher than `getrusage(RUSAGE_SELF, &rstart);` so i did not understand how to have proper results on these tests.