TARBE Théophile
QUIDET Victor
ING4 SI (inter) Gr01

LAB3 : Processes vs Threads

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

I - Processes

Users > theophiletarbe > Desktop > cours > ING4 > SystemeExpl > lab3 > C lab3_processes.c > main(void)

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <sys/times.h>

clock_t times(struct tms *buf);

int main(void)
{
    struct tms start, end;
    struct rusage rstart, rend;

    // (a+b) - [(c*d) / (e-f)] + (g+h)
    double a = 2, b = 8, c = 5, d = 5, e = 10, f = 5, g = 4, h = 16;
    int shmid;
    double *res;
    int i;
    //struct tms time_buf;
    times(&start);
    getrusage(RUSAGE_SELF, &rstart);

    // create the varaibles to share results; using 4 slots to make things clear but can do with just 3
    shmid = shmget(IPC_PRIVATE, 4 * sizeof(double), IPC_CREAT | 0660);
    res = shmat(shmid, NULL, 0);

    ///1000 times in order to get a value large enough to determined it with times() & getrusage()
    for (i = 0; i < 1000; i++)
    {
        if (fork() == 0)
        {
            // child 1s
            res[1] = a + b;
            exit(0);
        }
        // parent
        if (fork() == 0)
        {
            // child 2
            res[2] = (c * d);
            exit(0);
        }
        if (fork() == 0)
        {
            // child 3
            res[3] = (e - f);
            exit(0);
        }
        // parent again...
        res[0] = g + h;

        // wait for children
        wait(NULL);
        wait(NULL);
        res[4] = res[1] - (res[2]/res[3]) + res[0];
    }

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

    printf("My sum = %lf\n", res[4]);
    printf("function times : %lf usec\n", (end.tms_utime + end.tms_stime - start.tms_utime - start.tms_stime) * 1000000.0 / sysconf(_SC_CLK_TCK));

```

4 processes

Calculation

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

    printf("My sum = %lf\n", res[3]);
    printf("function times : %lf usec\n", (end.tms_utime + end.tms_stime - start.tms_utime - start.tms_stime) * 1000000.0 / sysconf(_SC_CLK_TCK));

    //function getrusage
    //1 = time
    printf("getrusage() --> Time : %ld usec\n", (rend.ru_utime.tv_sec-rstart.ru_utime.tv_sec)*1000000 +(rend.ru_utime.tv_usec-rstart.ru_utime.tv_usec)+(r
    //2 = input
    printf("getrusage() --> Input : %ld input operations\n", (rend.ru_inblock-rstart.ru_inblock));
    //3 = output
    printf("getrusage() --> Input : %ld output operations\n", (rend.ru_oublock-rstart.ru_oublock));
    //4 = switches (voluntary + unvolontary)
    printf("getrusage() --> Switches : %ld\n", (rend.ru_nvcsw-rstart.ru_nvcsw)+(rend.ru_nivcsw-rstart.ru_nivcsw));
}
```

```
[➜  lab3 gcc -g -o exe lab3_processes.c
[➜  lab3 ./exe
My sum = 25.000000
function times : 480000.000000 usec
getrusage() --> Time : 488300 usec
getrusage() --> Input : 0 input operations
getrusage() --> Output : 0 output operations
getrusage() --> Switches : 2574
➜  lab3
```

II - Threads

Users > theophiletarbe > Desktop > cours > ING4 > SystemeExpl > lab3 > C lab3_threads.c > main()

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

clock_t times (struct tms * buf);

// 4 threads function in order to compute simple operations
void * sum1(void *arg) {
    int n1 = 2, n2 = 8;
    int *res1 = malloc(sizeof(int));
    *res1 = n1 + n2;
    pthread_exit(res1);
}

void * sum2(void *arg) {
    int n3 = 5, n4 = 5;
    int *res2 = malloc(sizeof(int));
    *res2 = n3 * n4;
    pthread_exit(res2);
}

void * sum3(void *arg) {
    int n5 = 10, n6 = 5;
    int *res3 = malloc(sizeof(int));
    *res3 = n5 - n6;
    pthread_exit(res3);
}

void * sum4(void *arg) {
    int n7 = 4, n8 = 16;
    int *res4 = malloc(sizeof(int));
    *res4 = n7 + n8;
    pthread_exit(res4);
}

int main() {

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

    int i;
    int *result1, *result2, *result3, *result4, *final_result;

    pthread_t thread1, thread2, thread3, thread4;
    int  iret1, iret2, iret3, iret4;

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

4 threads functions

```c
///1000 times in order to get a value large enough to determined it with times() & getrusage()
for (i = 0; i < 1000; i++) {

    iret1 = pthread_create( &thread1, NULL, sum1, NULL);
    if(iret1) {
        fprintf(stderr,"Error - pthread_create() return code: %d\n",iret1);
        exit(EXIT_FAILURE);
    }

    iret2 = pthread_create( &thread2, NULL, sum2, NULL);
    if(iret2) {
        fprintf(stderr,"Error - pthread_create() return code: %d\n",iret2);
        exit(EXIT_FAILURE);
    }

    iret3 = pthread_create( &thread3, NULL, sum3, NULL);
    if(iret3) {
        fprintf(stderr,"Error - pthread_create() return code: %d\n",iret3);
        exit(EXIT_FAILURE);
    }

    iret4 = pthread_create( &thread4, NULL, sum4, NULL);
    if(iret4) {
        fprintf(stderr,"Error - pthread_create() return code: %d\n",iret4);
        exit(EXIT_FAILURE);
    }

    /* Wait till threads are complete before main continues.*/
    pthread_join(thread1, (void **) &result1);
    pthread_join(thread2, (void **) &result2);
    pthread_join(thread3, (void **) &result3);
    pthread_join(thread4, (void **) &result4);

    //We compute the  final result with the values returned by eatch thread
    *final_result = *result1 - (*result2/(*result3)) + *result4;
}

//Display the result
printf ( "Final result = %d\n" , *final_result);

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

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

//function getrusage
//1 = time
printf("getrusage() --> Time : %ld usec\n", (rend.ru_utime.tv_sec-rstart.ru_utime.tv_sec)*1000000 +(rend.ru_utime.tv_usec-rstart.ru_utime.t
//2 = input
printf("getrusage() --> Input : %ld input operations\n", (rend.ru_inblock-rstart.ru_inblock));
//3 = output
printf("getrusage() --> Output : %ld output operations\n", (rend.ru_oublock-rstart.ru_oublock));
//4 = switches (voluntary + unvolontary)
printf("getrusage() --> Switches : %ld\n", (rend.ru_nvcsw-rstart.ru_nvcsw)+(rend.ru_nivcsw-rstart.ru_nivcsw));
```

Creation of the 4 threads

Computing the final result

```
➜  lab3 gcc -g -o exe lab3_threads.c
➜  lab3 ./exe
Final result = 25
times() function : 110000.000000 usec
getrusage() --> Time : 120119 usec
getrusage() --> Input : 0 input operations
getrusage() --> Output : 0 output operations
getrusage() --> Switches : 4954
➜  lab3
```

In your report, you should explain the following :

1. **How you implemented the parallelized calculation**

Concerning the second part, the parallelized calculation it is implemented through 4 different threads, each one related to a function which makes a little calculation and returns the value of the operation stocked in a pointer thanks to: `pthread_exit(ptr)`

`pthread_join()` waits for termination of another thread, in this case for the 4 threads to be achieved. Once it's done, we can easily compute the `final_value` stocked in the pointer `final_value` by using the values calculated and returned by the threads.

2. **List the differences between the process and the thread versions and which one is more adapted in this case ?**

Threads are not independent of one other like processes. As a result threads share with other threads their code section, data section and OS resources. In our case threads share their functions `sum()` whereas processes execute the different `fork()` independently.

Moreover multithreading (as our case in part2) allows several functions (here our `sum()`) to be executed in parallel, whereas the processes (`fork()`) must wait for an execution order determined by the scheduler.

As a result, thread version is more adapted in our case because threads operate faster than processes, and gives us the solution of the calculation solutions faster. In our case this is very simple so the difference is minimal but in more complicated calculations the difference would be remarkable due to these reasons:

1) Thread creation is much faster
2) Context switching between threads is much faster
3) Threads can be determined easely
4) Communication between threads is faster

3. **Explain how performance measurement has been conducted and state which version of the solution is better based on your results.**

In order to compare performance between the two versions, we used 2 different functions :

- **`times()`** stores the current process times in the *`struct tms`* that *`buf`* points to. In order to use it, we declare a variable of type *`struct tms`* `(start & end)`. We pass the address of the variable `start` in the function `time()` at the beginning and we pass `end` when we want the function to stop running. Then we can display the result in a `printf()` by using `clock_t tms_utime` (user time) and `clock_t tms_stime` (system time)
- **`getrusage()`** returns resource usage measures for `RUSAGE_SELF` in our case. It returns resource usage statistics for the calling process, which is the sum of resources used by all threads in the process. Then we can also compute the time of execution by using variable of type `struct rusage (rstart & rend)` passed in argument.

Now if we look at the results of both functions, we can note that the `getrusage()` function is more precise than `time()` but the results are slightly similar. The main difference is that the time of execution is almost 5 times bigger for processes than for threads :

- Processes : ~ 480000 *usec*
- Threads : ~ 110000 *usec*

→ In our case we can conclude that the thread solution which has better performances (based on our results) is better.

4. **Give the results concerning I/O and context switches. Is there a difference between the two versions and why ?**

If we compare the results concerning I/O and context switches :

- **I/O** are both at 0 in the 2 versions. It computes the number of times the filesystem had to perform input or output but in our case it is null.
- **context switch** is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point. Here, they are 2 times more important for threads compare to processes (~ 2500 & ~ 5000) . It means that there is much more "communication" with the central processing unit (CPU) when using threads.

Sources :

http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

https://pedago-ece.campusonline.me/pluginfile.php/309353/mod_resource/content/0/test.c

https://pedago-ece.campusonline.me/pluginfile.php/309354/mod_resource/content/0/test_times.c

https://pedago-ece.campusonline.me/pluginfile.php/309351/mod_resource/content/0/lab1-process.c

https://man7.org/linux/man-pages/man2/getrusage.2.html

https://man7.org/linux/man-pages/man2/times.2.html

https://www.geeksforgeeks.org/multithreading-c-2/

https://stackoverflow.com/questions/30142868/calculate-the-sum-of-two-numbers-using-thread

https://ostechnix.com/how-to-find-the-execution-time-of-a-command-or-process-in-linux/