CS 3113 Intro to Operating Systems

Name and ID __Daniel Yowell____113558774

Homework #3

Due date: 03/21/2022 at 11:59 pm

Instructions:

1) To complete assignment one, you need to read Chapters 1, 3 and 4 of the textbook.

2) HW must be submitted on typed pdf or word document.

You must do your work on your own.

Q1. Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores. (15 points)

a.

$$speedup\ gain \leq \frac{1}{S + \frac{(1-S)}{N}}$$

$$S = \text{serial component} = 100\% - 60\% = 40\% = 0.4$$

$$N = \text{number of cores} = 2$$

$$speedup\ gain \leq \frac{1}{0.4 + \frac{(1-0.4)}{2}} = \frac{1}{0.4 + \frac{0.6}{2}} = \frac{1}{0.4 + 0.3} = \frac{1}{0.7} = \frac{10}{7}$$

Thus our speedup gain can be up to 10/7 times faster, or roughly 1.43 times faster.

b. This is the same as part a, except now $N = 4$.

$$speedup\ gain \leq \frac{1}{0.4 + \frac{(1-0.4)}{4}} = \frac{1}{0.4 + \frac{0.6}{4}} = \frac{1}{0.4 + 0.15} = \frac{1}{0.55} = \frac{100}{55} = \frac{20}{11}$$

Thus our speedup gain can be up to 20/11 times faster, or roughly 1.82 times faster.

Q2. A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread). (20 points)

a. How many threads will you create to perform the input and output? Explain.

We only have one file to perform the input and output functions on, and these functions are most likely serial (i.e. it would not help to divide them into parallel threads). Therefore, we only need one thread for the I/O functions.

b. How many threads will you create for the CPU-intensive portion of the application? Explain.

The CPU-intensive portion, on the other hand, can be optimized with multithreading. Since we have four processors, we can assign a thread to each processor, resulting in four total threads.

Q3. Consider the following code segment: (15 points)

```
pid t pid;

pid = fork();

if (pid == 0) { /* child process */

fork();

thread create( . . .); /* for the purpose of this problem, you can ignore the lack

of arguments to the function */

}

fork();
```

a. How many unique processes are created?

8 unique processes are created.

- At the start of the program, a single process is forked into 2.
- The child process is then forked into 2 processes, for a total of 3.
- The 2 processes from the previous step each split into 2 threads. We now have a total of 3 unique processes and 5 unique threads (1 process with 1 thread and 2 processes with 2 threads: 1 + 2(2) = 1 + 4 = 5).
- Finally, every thread calls fork(). For the 3 unique processes, this is carried out as normal, resulting in 6 processes. However, when fork() is called in a thread, it generates a new process identical to the parent, except with the only running thread being the one which called fork(). This means our 2 additional threads will result in 2 processes and 2 extra threads (4 threads in total). This brings our total number of unique processes to 6 + 2 = 8.

b. How many unique threads are created?

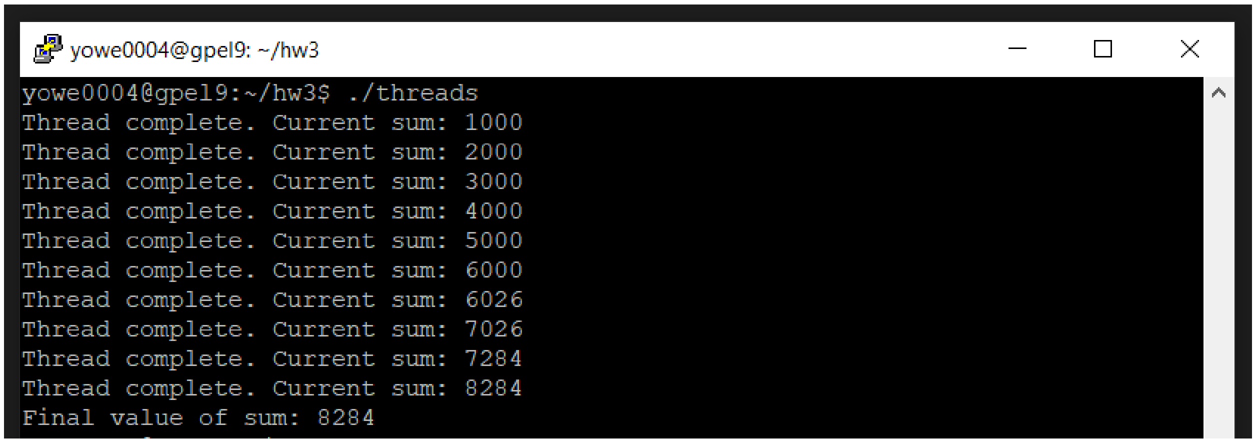
- Adding the 6 threads from each unique process as well as the 4 threads from our previous calculation gives us 10 total threads.

Q4. Pthread programming: writing a program to join on ten threads for calculating 0-9999. Each thread calculates the sum of 1000 numbers. Please attach screenshots of your execution results below. You also need to submit your code (along with a readme file) separately. (50 points)

All files (MUST INCLUDE: source codes, a readme file, and homework 2) should be zipped together.

https://github.com/danielyowell/CS3113-Homework3

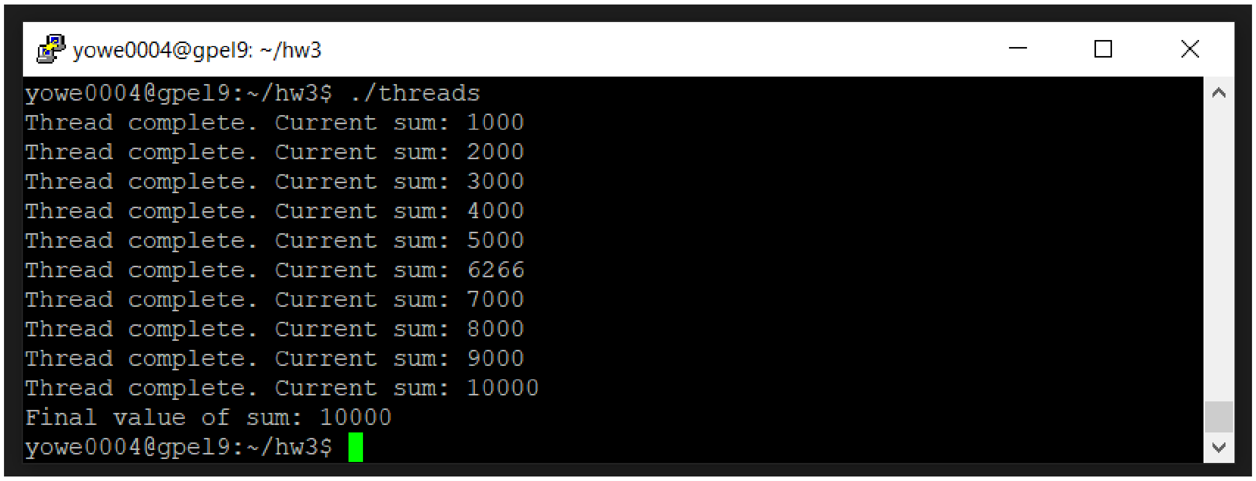
When “mutual_exclusion” is set to false:



yowe0004@gpel9: ~/hw3

```
yowe0004@gpel9:~/hw3$ ./threads
Thread complete. Current sum: 1000
Thread complete. Current sum: 2000
Thread complete. Current sum: 3000
Thread complete. Current sum: 4000
Thread complete. Current sum: 5000
Thread complete. Current sum: 6000
Thread complete. Current sum: 6026
Thread complete. Current sum: 7026
Thread complete. Current sum: 7284
Thread complete. Current sum: 8284
Final value of sum: 8284
```

When “mutual_exclusion” is set to true:



yowe0004@gpel9: ~/hw3

```
yowe0004@gpel9:~/hw3$ ./threads
Thread complete. Current sum: 1000
Thread complete. Current sum: 2000
Thread complete. Current sum: 3000
Thread complete. Current sum: 4000
Thread complete. Current sum: 5000
Thread complete. Current sum: 6266
Thread complete. Current sum: 7000
Thread complete. Current sum: 8000
Thread complete. Current sum: 9000
Thread complete. Current sum: 10000
Final value of sum: 10000
yowe0004@gpel9:~/hw3$
```