HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY, VNU HCM
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEM

Lab 5: Thread

Pham Minh Tuan - MSSV: 1752595

1 Exercises

1.1 Questions:

Question 1: What resources are used when a thread is created ? How do they differ from those used when a process is created?

- When thread is created they are independent executing the program like processes, except the global memory, the initialized data, uninitialized data, and heap segments are shared between threads.
- When process is created, for example `fork()`, the child created doesn't share the memory with parent's process, instead it copies all the attributes and resources existed in and executes independently with its parent.

Question 2:

(a) How is concurrency different from parallelism? Provide an example to clarify your answer. In developing your answer, be sure to address the definition of concurrency and parallelism.

- Concurrency is when two or more tasks can start, run and terminate in overlapping time periods. That is, it doesn't necessarily to run at same instant at the time. For example the *multitasking* on the single-core machine
- Parallelism is when two or more tasks **literally** run at the same time periods. For instance, the *multi-core* processor.

(b) Then explain how it is possible to have concurrency but not parallelism.

- The concurrency is possible to happen when we have a shared resources between two or more tasks, it requires the way to perform each jobs in task not really exactly at the same time.
- And the way concurrency performs can be interrupted at specific time (synchronization in resources)

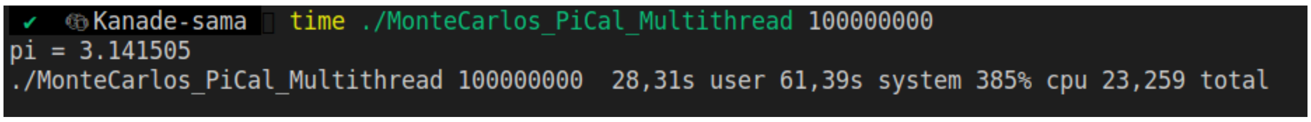
2 Programming Exercises:

Problem 1:

- Explain how your program solve the problem concurrently (e.g., how did you distribute the counting task to each thread?).
  - The shared variable is number of point in circle and number of point per thread.
  - I divide points equally in 5 threads and each thread will concurrently find and count the point that is inside the circle.
  - I initialize the mutex, and use mutex for locking in thread function to update the number of thread inside the circle, also call `pthread_exit(NULL)` at the end of thread function.
  - After all threads finishes its work, I call `pthread_join()`, to join the result of each thread.
  - When the calculation completes call `pthread_exit(NULL)` to exit the main thread.

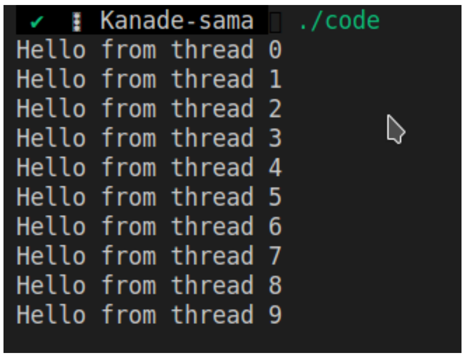
- How does your method augment the performance? (i.e., determine the speed-up, the difference in running time between multithreaded and single-threaded program. You should calculate the time reduced in terms of number of threads)
  - Since I divide the counting point task into 5 task, so the time for finish the counting is reduced to 5 times compared with the single-threaded program.
  - However the time or speed-up of the program depends on the cores of the CPU. If the CPU has only 1 core, the time spent for a single-threaded program or the multithreaded program is exactly the same after all because **only** one thread can be executed at the time only. The difference only occurs when you have multiple CPUs since more than 1 threads can run ar the time.
- Calculate the execution time needed to handle 100 million points. It should be within 5 seconds.

  The time need to handle 100 million points is 89.7s in total.



```
✔  Kanade-sama  time ./MonteCarlos_PiCal_Multithread 100000000
pi = 3.141505
./MonteCarlos_PiCal_Multithread 100000000  28,31s user 61,39s system 385% cpu 23,259 total
```

Problem 2:



```
✔  Kanade-sama  ./code
Hello from thread 0
Hello from thread 1
Hello from thread 2
Hello from thread 3
Hello from thread 4
Hello from thread 5
Hello from thread 6
Hello from thread 7
Hello from thread 8
Hello from thread 9
```