**Score: \_\_\_\_\_**

**OSA6 – Threads**

**Activities**

COMP256 – Computing Abstractions

Dickinson College

Spring 2023

Prof. Grant Braught

**Name:**

**Introduction:**

Today’s class generalized the notion of a process context to that of an execution context. Operating systems can use multiple execution contexts within a single process to provide a Thread abstraction. Threads make it possible to execute different parts of the same program simultaneously. All of the threads in a process share a logical memory space, including the code, data and heap segments. But every thread in a process has its own context (PC, register values, stack). By having an independent context, each thread may be executing a different part of the program and/or processing a different part of the program’s data. The following activities will give you experience working with Java threads and will use that experience to reinforce your understanding of the thread concept.

It is not required viewing, but if you are interested in another introduction to threads check out the video *Thread vs. process - Java Tutorial* from the course Parallel and Concurrent Programming with Java 1. They introduce a cooking metaphor and do a nice job of illustrating the multiple processes, each with multiple threads.

* <https://www.linkedin.com/learning/parallel-and-concurrent-programming-with-java-1/thread-vs-process?autoplay=true&trk=learning-course_table-of-contents_video&upsellOrderOrigin=default_guest_learning> (4:36)

**Your Metaphor:**

🔑 1. This question asks you to use your metaphor to illustrate threads using your metaphor.

a. In a sentence, reintroduce your metaphor. You can copy this from a previous assignment. I ask for it to help me remember what it was as I read the rest of your answer.

b. Identify the elements of your metaphor that you will be using to play the roles of the hardware (i.e. device(s)), the processes, and the operating system.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **System Element** | **Metaphor Element** |  |
|  | Memory |  |  |
|  | Processes |  |  |
|  | Operating System |  |  |
|  |  |  |  |

c. Use the elements of your metaphor from part b to explain the idea of a process having multiple points of execution (i.e. threads). Be sure to clearly identify what the threads are within the process and how each thread has its own independent execution context.

**Java Threads:**

Most modern high-level languages provide the ability to write programs with multiple threads (a.k.a. multithreaded programs). We’ll look at Java as an example, but C/C++, Go, JavaScript, Rust, Python, etc. all provide similar capabilities, though of course the particular syntax and mechanisms that they use differ slightly.

🔑 2. Consider the first example of a multithreaded program presented during today’s class:

* <https://repl.it/@braughtg/TwoThreads>

a. In Java, the code that is to be run in a separate thread is defined in a class (e.g. PrintIt).

i. What is the super-class of the class PrintIt?

ii. The super-class that you identified in part i defines an abstract method that all sub-classes must implement. This method contains the code for the thread. What is the signature of the abstract method that will contain the code for the thread?

b. Run the TwoThreads program multiple times and observe the output. Notice that the output is not the same each time. In the table below copy and paste the output of four runs that each have different outputs. It may take a number of tries to get four different results - keep trying.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **Output 1** | **Output 2** | **Output 3** | **Output 4** |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

c. Let’s now make some observations about the outputs that you found in part b.

i. Does the output “Hi I’m Main” always appear first? Briefly explain why.

ii. Do all of the lines of output from PrintIt (or from main) always appear together or are they intermixed sometimes? Briefly explain why.

iii. Do the numbered lines output by main always appear in numeric order? This is asking if 0: main always come before 1: main, and if that always comes before 2: main, even if the main’s and the Its are intermixed.

iv. Do the numbered lines output by It always appear in numeric order, even if the Its and the mains are intermixed?

v. Explain why the mains and Its always appear in numeric order, even if they are intermixed.

vi. Does the same line always appear last?

🏆 3. **Optional:** There are multiple ways to create and execute threads in Java. The example above did so by creating a sub-class of Thread. An alternative that has advantages in some circumstances is to implement the Runnable interface. Read the following page from the Java Tutorial and then reimplement the TwoThreads program by using the Runnable interface instead of by extending Thread:

* <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>

🔑 4. In class we saw the following ThreadJoin example:

* <https://repl.it/@braughtg/ThreadJoin>

a. Which statement in this program causes the main thread to pause and wait for the thread t1 to complete?

b. Briefly explain why “main Done!” is now guaranteed to the be the last line every time?

5. Fork the Repl for ThreadJoin and modify it so that it guarantees that all of the numbered lines printed by PrintIt (e.g. 0: It) appear before any of the numbered lines that are printed by main (e.g. 0: main). Be sure to run your program a number of times to ensure that it works. Paste your modified program below.

6. Fork the Repl for the TwoThreads example (<https://repl.it/@braughtg/TwoThreads>)

a. Modify your fork of TwoThreads so that it:

* creates one thread of type PrintIt
* contains a new Thread subclass named PrintIt2. This thread should output five numbered lines like “1: It2”, “2: It2”
* starts one PrintIt thread and one PrintIt2 thread. So that there are three threads (main, PrintIt, and PrintIt2) running simultaneously.

Paste a copy of the code for your modified program here. Be sure to include main, PrintIt and PrintIt2.

b. Fill in the column below with three different outputs generated by running your program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Output 1** | **Output 2** | **Output 3** |  |
|  |  |  |  |  |
|  |  |  |  |  |

c. Briefly explain how you can tell from part b that your program now has 3 different threads of execution and that they are all running independently.

7. Now modify the main method in your program from #6 so that the line “Main Done!” is always the final line printed. Hint: Add two join statements! Be sure to run your program many times and confirm that “Main Done!” always appears last.

Paste a copy of the code for your modified main method here. Do not include the PrintIt or PrintIt2 classes.

🏆 8. Now modify the main method in your program from #7 so that:

* The first 5 lines from main are output before any lines from PrintIt2. Though, the lines from main may be intermixed with the lines from PrintIt1. Hint1: Split the for loop in main into two loops. Hint2: Think about when to start PrintIt2.
* All of the lines of output from PrintIt1 are guaranteed to appear before the second 5 lines output from main. Though they can be intermixed with lines from PrintIt2.
* All of the line of output from PrintIt2 are guaranteed to appear before the “Main Done!” line.

Paste a copy of the code for your modified main method here. Do not include the PrintIt or PrintIt2 classes.

**Thread Applications:**

Threads can be used in applications in many ways. Two of the most common use patterns are parallelization and specialization. In both of these patterns a problem will be broken into multiple parts where each part is solved by a different thread.

9. With *parallelization*, each thread will execute the same code but will process different parts the data. Give a few sentences describing an example of parallelization using your metaphor from question #1.

10. With *specialization*, each thread will execute different code that processes the data in different ways. Give a few sentences describing an example of specialization using your metaphor from question #1.

**Context Switching:**

Performing context switches between processes or between threads take some time. These time costs can be broken into two categories, *direct costs* and *indirect costs*. Direct costs include things like saving and restoring registers and updating the page table so that the processor can run the new process. Indirect costs have to do with how cache hit rates are affected by the context switches.

11. Let’s consider the direct costs first:

a. What elements of the execution context are specific to each thread?

b. What elements of the execution context are shared by all threads within a process?

c. When the OS context switches between threads within the same process will the system incur more or less direct cost than switching between threads in different processes? Briefly justify your answer.

🏆 12. Indirect costs are more subtle than direct costs but can be understood based on the cache principles that we studied earlier.

a. What are the two locality principles that determine how effective caching will be?

b. Will two threads that exist within the same process be more or less likely to obey these principles than threads that are from two different processes? Briefly justify your answer.

c. Would switching between two threads that are in the same process be more or less likely to produce cache misses as compared to switching between two threads that are in different processes. Briefly justify your answer.

d. Will context switching between two threads within the same process incur more or less indirect cost than switching between two threads that are in different processes? Briefly justify your answer.

Optional: To help me improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.