**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 mechanism 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **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 and PrintIt always appear in numeric order? This is asking if 0: main always come before 1: main …, and if 0: It always comes before 1: It …, even if the main’s and the Its are intermixed.

iv. 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. The inclusion of the statement you found in part i causes the final line of output that is generated to always be the same. What is that line of output and why it is now guaranteed to the be the last line?

🔑 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 the modified program below.

6. Fork the Repl for ThreadJoin again and modify it so that it:

* creates a two threads of type PrintIt and starts each in an independent execution context. Thus, three threads (main plus 2 PrintIts) will be running simultaneously.
* ensures that:
  + All of the numbered lines from main print before any lines from either PrintIt.
  + The line “Main Done!” is always the final line printed.

Note: You do not need to add another class to the program, just create another instance of PrintIt and call start() on that one too.

**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. With *parallelization*, each thread will execute the same code but will process different parts the data. With *specialization*, each thread will execute different code that processes the data in different ways. The following questions will give you experience with each of these approaches.

7. Consider the following ThreadParallelization example:

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

a. What two static fields are in the Main class?

b. The field values is initialized with the return value from the getArray method on line 9. Describe what the getArray method returns.

c. Lines 11 and 12 create and start a new SumIt thread.

i. What are the parameters to the constructor of the SumIt thread?

ii. Describe what the SumIt thread computes when its run method is executed.

iii. Run the program. What value is computed as the final result?

iv. If you run the program multiple times does the final result change?

Note: This might be surprising since the program uses the random() method to fill the array. But because the Random class is created with the statement new Random(183) instead of what you might be more used to seeing, new Random(). When a parameter is passed to the constructor for the Random class, this initializes the random number generator with a *seed*. Doing so causes it to generate the same sequence of numbers. While not useful in practice for generating random numbers, it is a very useful feature for testing. It is not required reading, but if you are curious you can learn more here:

* + <https://mathbits.com/JavaBitsNotebook/LibraryMethods/RandomGeneration.html>

d. This example does not yet do parallelization since one thread processes all of the data.

Fork the ThreadParallelization example and modify the main method so that it uses four threads. Each thread should sum up ¼ of the values. For example, the first thread could sum the first ¼ of the values. The main thread should wait until all four threads are complete and then add up their results to get the total of all of the values. Paste the code from your main method below.

e. What value does your program report as the sum of all of the values? Does it match your result from part c.iii above? It should. If it does not revisit your solution in part d.

8. ThreadSpecialization program linked below provides starter code for this exercise.

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

a. Fork the ThreadSpecialization program and modify it so that it uses thread specialization to perform several different computations on an array of data concurrently: Your modified program should find and print the maximum and minimum of the values array. In doing so your program should:

* + use two specialized threads, one to compute the max and one to compute the min.
  + ensure that these specialized threads run simultaneously. That is neither one of them should be forced to wait for the other.
  + ensure that the main thread waits for both of the specialized threads to complete before displaying the results.

Paste the code for your main method and each of your specialization threads below. Note: Because a seed was used for the Random class, your answer should always be the same. You should find that:

max: 9999998

min: 5

**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.

🏆 9. 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.

🏆 10. 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.