CS 3220: Operating Systems

Assignment 2

Date Assigned: Mar 22nd, 2018 Date Due: Mar 25th, 2018

(**Total Score**: 17 points)

(**Questions**: 7)

Q 1. Under what circumstances is a single-threaded system better than a multi-threaded system? **(2 points)**

Q 2. Name two real-world applications in which multi-threading does not provide better performance than a single-threaded solution. **(2 points)**

Q 3. In a real-world application of a Bank ATM server that processes all customers' requests placed upon their respective ATM machines, provide names and quantities of all possible threads that might be needed in the server for a process, where a process is associated with a single user using a single ATM machine. **(5 points)**

Q 4. In Figure 2, the register set is listed as a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers. **(2 points)**

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Figure 2. Public versus Private Items of Threads

Q 5. Why would a thread ever voluntarily give up the CPU by calling $thread\_yield$? After all, since there is no periodic clock interrupt, it may never get the CPU back. **(2 points)**

Q 6. If a multithreaded process forks, a problem occurs if the child gets copies of all the parent's threads. Suppose that one of the original threads was waiting for keyboard input. Now two threads are waiting for keyboard input, one in each process. Does this problem ever occur in single-threaded processes? Give reason. **(2 points)**

Q 7. Which one is a better implementation of a World Wide Web server; using User-level threads or Kernel-level threads? Give reason. **(2 points)**

A 1. The need of threads take place when some single-threaded processes are I/O dependent, so some part of the CPU time they are allocated is used in waiting for data, during which the entire processes remain blocked. During this time, some useful work can be done by other processes or miniprocesses (threads). However, if no such processes (I/O dependent) exist in a system, then there is no need of threads, and a single-threaded system would be better than a multi-threaded one.

A 2.

i. Game of Chess
ii. Clock

A 3. The ATM machine would be implementing a single-threaded process for a user, since both, all work carried out by a user through ATM machine, and service provided to each user separately by the same ATM machine, are sequentially processed. However, the ATM server would be implementing a multi-threaded system, where each unique type of thread would be assigned responsibility of a unique type of function, details of which are provided in Table 1.

S#	Thread	Quantity
1	User Login	High
2	Balance Check (Balance Check, Cash Withdrawal)	High
3	Bill Payment	Medium
4	Cash Deposit	Low
5	Money Transfer	Low

Table 1. Threads in the Server

A 4. Since registers store current working variables of a thread, they are considered a private entity for a thread. Information in the registers is stored each time CPU switches from one thread to another. Hence the register set is listed as a per-thread rather than a per-process item.

A 5. Threads cooperate, rather than compete (like processes), with each other in order to successfully and efficiently run a process. Voluntarily giving up the CPU is just one way of such cooperation among threads. $thread\_yield$ is usually used by programmers for applications, where it is known beforehand that one thread would take more time than usual, while the other thread would take less time than usual, and after finishing its work the latter thread would let the former thread (which activated $thread\_yield$) continue with its work. Hence, the possibility of the former thread not receiving the CPU back from the latter thread is quite low.

A 6. In single-threaded processes, if the (only) thread of a process is waiting for keyboard input, the whole process gets blocked, and until a process is blocked, it can not compute further (one of the computation tasks is forking; creating a child process). On the other hand, a multi-threaded process can fork even if one of its threads is blocked, since the whole process does not get blocked due to any of its threads getting blocked.

A 7. Kernel-level threads is a better implementation of a World Wide Web server, compared to User-level threads, since in the latter implementation the whole process gets blocked due to any of its

threads getting blocked. In a World Wide Web server, we have two threads; Dispatcher and Worker. In a User-level threads implementation, if Worker gets blocked while it is fetching data from storage (since it might take some time), the whole user process (containing Dispatcher and Worker threads) will get blocked, and even Dispatcher (which takes less time to process its task and is therefore available to perform more tasks) would not be able to do its job.