King Saud University
Department of Computer Science
CSC227: Operating Systems
Tutorial No. 4

Ex 1. What two advantages do threads have over multiple processes? What major disadvantage do they have? Suggest one application that would benefit from the use of threads, and one that would not.
Answer:
- Threads are very inexpensive to create and destroy, and they use very little resources while they exist.
- They do use CPU time for instance, but they don't have totally separate memory spaces.
- Threads must "trust" each other to not damage shared data. For instance, one thread could destroy data that all the other threads rely on, while the same could not happen between processes unless they used a system feature to allow them to share data.
- Any program that may do more than one task at once could benefit from multitasking. For instance, a program that reads input, processes it, and outputs it could have three threads, one for each task.
- "Single-minded" processes would not benefit from multiple threads; for instance, a program that displays the time of day.

Ex 2. What resources are used when a thread is created? How do they differ from those used when a process is created?
Answer: A context must be created, including a register set storage location for storage during context switching, and a local stack to record the procedure call arguments, return values, and return addresses, and thread-local storage. A process creation results in memory being allocated for program instructions and data, as well as thread-like storage. Code may also be loaded into the allocated memory.

Ex 3. Describe the actions taken by a kernel to switch context
a) Among threads.
b) Among processes.

Answer:
a) The thread context must be saved (registers and accounting if appropriate), and another thread's context must be loaded.
b) The same as (a), plus the memory context must be stored and that of the next process must be loaded.

Ex 4. Describe similarities and differences between thread and process.
In many respect threads operate in the same way as that of processes. Some of the similarities and differences are:
<u>**Similarities**</u>
- Like processes threads share CPU and only one thread active (running) at a time.
- Like processes, threads within a processes execute sequentially.
- Like processes, thread can create children.

- And like process, if one thread is blocked, another thread can run.

<u>Differences</u>

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task.
- Unlike processes, threads are design to assist one other. Note that processes might or might not assist one another because processes may originate from different users.

Ex 5. Why threads are used in operating system?
Following are some reasons why we use threads in designing operating systems.

- A process with multiple threads make a great server for example printer server.
- Because threads can share common data, they do not need to use interprocess communication.
- Because of the very nature, threads can take advantage of multiprocessors.
- Threads are cheap in the sense that

a) They only need a stack and storage for registers therefore, threads are cheap to create.
b) Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources.
c) Context switching is fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.
d) The biggest drawback is that there is no protection between threads.

Ex 6. What are the differences between user-level threads and kernel-supported threads? Under what circumstances is one type "better" than the other?
Answer: User-level threads have no kernel support, so they are very inexpensive to create, destroy, and switch among. However, if one blocks, the whole process blocks. Kernel-supported threads are more expensive because system calls are needed to create and destroy them and the kernel must schedule them. They are more powerful because they are independently scheduled and block individually.