King Saud University
Department of Computer Science
CSC227: Operating Systems
Tutorial – Chapter 4: Threads

Q 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.

- 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.

Q 2) What resources are used when a thread is created? How do they differ from those used when a process is created?

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.

Q 3) Describe the actions taken by a kernel to switch context

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

Q 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:

Similarities:

- Like processes threads share CPU and only one thread active (running) at a time.
- Like processes, threads within a process execute sequentially.
- Like processes, thread can create children.
- And like process, if one thread is blocked, another thread can run.

Differences:

- 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.

Q 5) Why threads are used in operating system? What is their biggest drawback?

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 inter-process communication.
- Because of the very nature, threads can take advantage of multiprocessors.
- Threads are cheap in the sense that:
    - They only need a stack and storage for registers therefore, threads are cheap to create.
    - 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.
    - Context switching is fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

The biggest drawback is that there is no protection between threads.

Q 6) What are the differences between user-level threads and kernel-supported threads? Under what circumstances is one type "better" than the other?

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.

Q 7) Give programming examples in which multithreading provides better performance than a single-threaded solution.

1. A web server that services each request in a separate thread.
2. A web browser with separate threads playing sound, downloading a file, collecting user input, etc.
3. A word processor with a thread to save, at regular intervals, the file that is currently being executed, and another thread as a spell-checker, etc.
4. An application such as matrix multiplication where each row of the matrix product is evaluated, in parallel, by a different thread.

Q 8) What is the difference between a process and a thread? Which one consumes more resources?

Some of the difference are:

- A process defines the address space, text, resources, etc. A thread defines a single sequential execution stream within a process.
- Threads are bound to a single process. Each process may have multiple threads of control within it.
- It is much easier to communicate between threads than between processes.
- It is easy for threads to accidentally disrupt each other since they share the entire address space.

Process consumes more resources.

Q 9) Assume that the OS implements Many-to-Many multithreading model. What is the minimum number of kernel threads required to achieve better concurrency than in the Many-to-One model? Why?

At least two kernel threads are required to achieve better concurrency in the many-to many model than in the Many-to-One model. With more than two kernel threads, CPU can still schedule other threads for execution when one threads is performing a blocking system calls.

Q 10) Consider two scenarios: a) two user threads are mapped into one kernel thread, and b) each of the two user threads is mapped into a unique kernel thread. Which achieves better concurrency in execution? Why?

The b) achieves better concurrency since two threads can execute concurrently. In a), when one thread is blocked or in waiting state, the other thread cannot be scheduled to run.

Q 11) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?

A multithreaded system consisting of multiple user-level threads mapped to one kernel thread cannot make use of the different processors in a multiprocessor system. Consequently, there is no performance benefit associated with this solution. The multithreaded solution could be faster if the multiple user-level threads are mapped to different kernel threads.

Q 12) Write a C program that calculates the sum of the numbers from 1 to 100,000,000. Split the numbers between 4 threads equally where each thread calculates the sum of one fourth of the numbers. For example, the 1st thread will calculate the sum of the numbers from 1 to 25,000,000 whereas the 2nd thread will calculate the sum of the numbers from 25,000,001 to 50,000,000 and so forth. The main thread will have to print out the sum after gathering the results of the 4 created threads.

```
#include <pthread.h>
#include <stdio.h>

unsigned long sum[4]; /* this data is shared by the thread(s) */

void *sum_thread(void *param); /* the thread */

int main(int argc, char *argv[]) {
      pthread_t workers[4];
      pthread_attr_t attr;

      /* get the default attributes */
      pthread_attr_init(&attr);

      /* create the thread */
      int i;
      for(i=0; i<4; i++)
            pthread_create(&workers[i], &attr, sum_thread,(void *) i);

      /* now wait for the thread to exit */
      for(i=0; i<4; i++)
            pthread_join(workers[i],NULL);

      printf("sum = %lu\n",sum[0]+sum[1]+sum[2]+sum[3]);
}

/**
 * The thread will begin control in this function
 */
void *sum_thread(void *param)
{
      long id = (long) param;
      int start = id*25000000;
      int i=0;
      while (i<25000000) {
            sum[id] += (i+start);
            i++;
      }
       pthread_exit(0);
}
```