Tutorial

Chapter 4

Threads

What two advantages do threads have over multiple processes?

- Threads are 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.

What major disadvantage do they have?

- 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.

Suggest one application that would benefit from the use of threads, and one that would not.

- 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.

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.

Describe the actions taken by a kernel to switch context
a) among threads.
b) among processes.

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.

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 processes execute sequentially.
  - □ Like processes, thread can create children.
  - □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.

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.

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.

Which of the following components of program state are shared across threads in a multithreaded process

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

- The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

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

- A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

Is it possible to have concurrency but not parallelism? Explain.

- Yes. Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

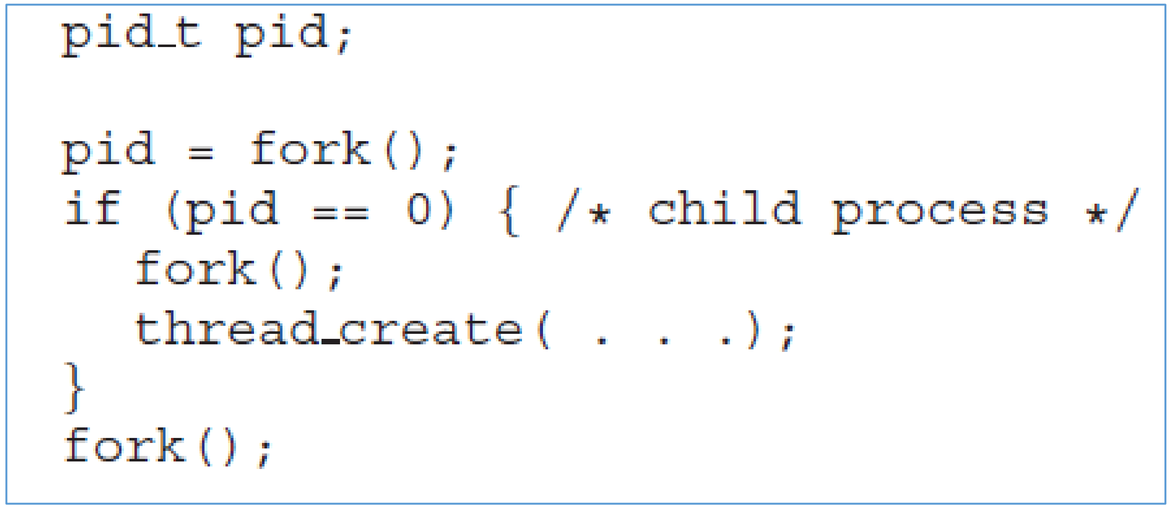
Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

- Two processing cores = 1.43 speedup; four processing cores = 1.82 speedup.

Consider the following code segment:

a. How many unique processes are created?

b. How many unique threads are created?



```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
   fork();
   thread_create( . . .);
}
fork();
```

- There are six processes and two threads.

Consider the following Java program

```
public class WhereAmI2 implements Runnable{
    int n;
    // constructor
    public WhereAmI2(int number) {
        n = number;
    }
    // override the run method
    public void run() {
        for (int i = 0; i < 100; i++)
            System.out.println("I'm in thread " + n);

    }
}
```

```
public class ThreadTester2 {
    public static void main(String[] args) {
        // create a runnable objects,
        // and the thread to run them.
        WhereAmI2 place1 = new WhereAmI2(1);
        Thread thread1 = new Thread(place1);
        WhereAmI2 place2 = new WhereAmI2(2);
        Thread thread2 = new Thread(place2);
        WhereAmI2 place3 = new WhereAmI2(3);
        Thread thread3 = new Thread(place3);
        // start the threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

- Modify the above program so that it prints as a last message

  “**I am in main**”

solution

- In main you need to wait for all threads to finish
- You can do that using

```
thread1.join();
thread2.join();
thread3.join();
```

- Then you print the required message using

```
System.out.println("I am in main" );
```