Homework 1

CS550 (V01)

Amit Nikam (A20470263)
anikam@hawk.iit.edu
Fall 2020

---

1. **(0 points)** Answer the questions given in Lecture 3, slide 19.

   •What is the difference between operating system and (software) system?
   Operating system is a software that is responsible for controlling communication with hardware, managing resources, users, permissions and keeping processes separated and scheduled. On the other hand, (software) system is a collection of software that work together for which may comprise of OS or use OS to start, run and maintain the user-space.

   •What is the difference between Network OS and Distributed OS?
   In Network OS each system has its own OS, where as in Distributed OS it is a common OS across each machine.

   •What is the difference between Distributed OS and Distributed (software) system?
   In distributed OS, single common OS is used on each system. Where as Distributed (software) system is a collection of machines running software and communicating to achieve a goal. The software might be running of different OS in Distributed System but not in Distributed OS.

   •What is middleware?
   Middleware is a software above OS level which provides services that are not provided by the OS. These services enable distributed systems to communicate and manage data. Middleware provides these services as an interface for the application in higher levels.

   •What is the difference between middleware and distributed (software) system?
   Middleware is a software that enables better communication and data management for the Distributed system. Distributed systems are built upon multiple systems using middleware software to achieve transparency.

2. **(10 points)** In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in a cache in main memory. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps.

   a) How many requests/sec can the server handle if it is single threaded?

   In a single threaded server, only one thread exists at a time. So given that 2/3 of the threads (t1 & t2) are fulfilled by cache in main memory, they get completed in 15 msec each. And collectively (t1 + t2) is 30 msec. The remaining t3 - thread require a disk operation and take a total of 90 msec to complete.

t1 + t2	t3
30 msec	90 msec

In total t1+t2+t3 take 120 msec to complete and on average a thread takes 40 msec to complete in a single threaded server.

Therefore, in a second i.e. 1000 msec, total number of requests that can be fulfilled are 1000/40 = 25 requests/sec.

b) If it is multithreaded?

In multithreaded, there can be multiple threads existing simultaneously. Thus taking the same case as above, t1 and t2 threads will still take 15 msec each. The main difference is observed at t3 thread, after 15 msec t3 would need a disk operation during which t3 thread will sleep and another thread would be executed by overlapping t3 thread.

Since every request will take 15 msec, this server can fulfill 1000/15 = 66.6667 requests/sec.

3. **(10 points)** Would it make sense to limit the number of threads in a server process?

   Yes, there are a few reasons ranging from security to resource management to do so.

   First, it might be possible for the server to get exploited and fork bombed, wherein the thread continuously replicates itself, which can lead to a slow down, denial-of-service or even server crashing due to resource starvation.

   Second, threads within a process share resources like file descriptors, visual memory, signal handlers to name a few. A large number of threads trying to use these resources might incur bottleneck problem leading to performance degradation of the process. Also dividing a fixed amount of work among many threads may give each thread little work compared to the overhead of starting and terminating threads.

   Lastly, threads require memory for setting up thread context. So having many threads may consume too much memory for the server to work properly.

4. **(10 points)** Consider a process P that requires access to file F which is locally available on the machine where P is currently running. When P moves to another machine, it still requires access to F. If the file-to-machine binding is fixed, how could the system-wide reference to F be implemented?

   In this situation, previous machine's existing remote file service could be used or a new one could be create if it doesn't exist. The file service from the previous machine would act like a file server and let the process P at new machine use the file F just as before, that is, P would be offered the same interface as before.

5. **(10 points)** List all the components of a program state that are shared across threads in a multithreaded process.

   Text Segments - instructions
   Data and BSS segment
   open file descriptors
   signal handlers
   current working directory heap memory
   User IDs and group IDs

6. **(10 points)** Explain the trade-offs between using multiple processes and using multiple threads.

   In multiple processes, every time a new process is created on a request, a new process context is created which includes process control block, text segments, heap memory and so on. On the other hand, a thread when created generates a

thread context which is comparatively less in size, for most components are shared with the process it is forked from. So using multiple processes can be memory consuming compared to multiple threads.

Additionally, processes need to be scheduled by the operating system for service, which also means there are more context switch. Context switch can be expensive due to the bottleneck at memory level. Where as in multiple threads the process can decide which thread to execute without any context switch.

Multiple threads, due to their nature are not secure. The sharing of resources between all the threads makes it vulnerable to violations. Processes on the other hand are isolated at operating system level and are more secured. Multiple processes can thus be used in applications where such security is a concern.

7. **(10 points)** Does a multi-threading solution always improve performance? Please explain your answer and give reasons.

   No, not every time. As we discussed above in question 3, there are multiple issues that we may face with multi-threading which can degrade performance. One of this issue is excessive usage of memory space by high number of threads, which can lead to performance degradation. And other issue is multiple threads using shared process resources leading to bottleneck issues.

   Additionally, in normal cases too, the OS is not context aware of the threads within a process. In such a case, an OS may context switch to a process which has threads that are blocked. So an OS may not provide CPU time to the threads of a process.

8. **(10 points)** Explain the trade-offs between preemptive scheduling and non-preemptive scheduling.

   In preemptive scheduling, if a high priority task arrives, the current task can be interrupted to let the high priority task execute. While this is a good way to obtain a stable kernel and OS, a series of high priority tasks can delay the execution of secondary tasks leading to what is known as starvation. On the other hand, in a non-preemptive scheduling, a task once started doesn't get interrupted until it goes to waiting state or is finished. In this, tasks with long burst times can lead to delay in execution of priority and secondary tasks.

   In priority scheduling overhead is involved and context switch can also be time consuming under some circumstances. While in non-preemptive no overhead is involved.

9. **(10 points)** What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

   First, the user-level threads are scheduled by thread library used and can be managed by the process while in kernel-level threading kernel is responsible for scheduling.

   Second, user-threads belong to a process and are unknown to the kernel while kernel threads are known by the kernel and may or may not belong to a process.

10. **(10 points)** What is the difference between a process and a thread. Which one consumes more resources?

    Process is a program under execution, while thread are lightweight process. Processes do not share memory, while threads do share memory with their peers. Processes have more context compared to threads and thus require

more time for context switching than threads. Processes consume more resources compared to threads.

11. **(10 points)** The X protocol suffers from scalability problems. How can these problems be tackled?

    X protocol is an application-level communication protocol provided by X Windows System which happens to be one of the oldest and widely used networked user interfaces. One of the scalability problems in X protocol is the high bandwidth need. To tackle this compression techniques can be used to reduce the size of X message.

    Second problem with scalability is synchronization, as an application and the display generally need to be synchronized too much. To tackle this a local state can be stored at application side as cache memory which the application will check to find out the state of the display. Also the application can only request differences in state, this way further bandwidth can be saved and synchronization can be achieved.

12. **(0 points)** Read textbook Chapter 1, 2, and 3.

    Done