Homework 1

Md Ali
CS 550: Advanced Operating Systems

September 19, 2020

Exercise 1. Answer the questions given in Lecture 3, slide 19. The questions are as follows:

a) What is the difference between operating system and (software) system?
b) What is the difference between Network OS and Distributed OS?
c) What is the difference between Distributed OS and Distributed (software) system?
d) What is middleware?
e) What is the difference between middleware and distributed (software) system?

Proof. Below are the corresponding answer to each question in exercise 1.

a) The difference between operating system and software system is that operating systems is a set of programs maintains all the processes and activities with computer hardware device while software system is an interface with the user, application software, and the computer's hardware.

b) The main difference between network operating system and distributed operating system is that in a network operating system each node and/or system will have its own operating system where as in a distributed operating system each node and/or system will have the same operating system. Other things to note is that in an network operating system the main objective is to provide the local services to remote clients and communication will take place on the basis of files while in distributed operating system the main object is to manage hardware resources and communication takes place in messages and shared memory.

c) Distributed operating systems will generally have the same operating system or similar ones at each node or system, it also manages more the communication relay between hardware while in a distributed software system there is more of an overlay between applications where the main objective is to provide more of a local service to remote clients that have various operating systems at each node or system basis.

d) Middleware is a piece of software that lies between an operating system and the applications running on it. This is working essentially as a layer that enables communication and data management for distributed applications.

e) Middleware is a piece of software that lies between the operating system and the application software. In contrast, software is encoded computer instructions for computer hardware while the middleware is a piece of software that that relays messages depending on if it was an application and operating system, databases, or client server.

□

Exercise 2. In this problem you are to compare reading a file using a single-threaded 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 man requests/sec can the server handle if it is a single threaded?
b) If it is multithreaded?

Proof. Below are the corresponding answer to each question in exercise 2.

First lets create a general formula where we will assume that we are dealing with a preemptive scheduler. We know that each request takes an average of 15 $msec$ of CPU time. This leaves us with the one-third of 75 for the I/O which means that it will take $(1/3) \cdot 75$ or 25 $msec$ I/O time. Let's make n the variable to represent the number of threads.

So the probability of n threads sleeping will be:

$$Probability\ n\ Threads\ Sleeping = \left(\frac{25}{40}\right)^n = \left(\frac{5}{8}\right)^n \tag{1}$$

We can subtract equation 1 from a unit value to get the utilization of the CPU as shown in equation 2.

$$CPU\ Utilization = 1 - \left(\frac{5}{8}\right)^n \tag{2}$$

From here we want to know the number of requests per second a server can handle if the server was single threaded and multithreaded. During one second the equation below will give the number of requests that the CPU will handle, hence giving our final equation to solve the problem.

$$Requests\ per\ Second = \left(1 - \left(\frac{5}{8}\right)^n\right) \cdot \left(\frac{1000}{15}\right) \tag{3}$$

a) Utilizing equation 3 with $n = 1$, we will get that the server can handle 25 req/s

b) Here we will put in $n = 2, 3, 4, 5$, and 6 as our values and see the gradual number of request the server can handle increase.
$n = 2,\ 40.625\ req/s$
$n = 3,\ 50.391\ req/s$
$n = 4,\ 56.494\ req/s$

$n = 5,\ 60.309\ req/s$
$n = 6,\ 62.693\ req/s$

□

Exercise 3. Would it make sense to limit the number of threads in a server process?

Proof. Yes it would make sense to limit the number of threads in a server process. This is due to the fact that threads require memory for setting up their own private stack. This will consume a lot of memory for the server to work. In addition, having many threads can lead to an issue with performance which will result in page thrashing. □

Exercise 4. 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?

Proof. I would suggest a separate process, for example S, that would solely handle the requests for F. The same interface would be utilized between P and F but in form of a proxy through S, making S basically a file server. □

Exercise 5. List all the components of a program state that are shared across threads in a multithreaded process.

Proof. It depends, but generally speaking that heap memory, global variables, and the page table are shared across a multithreded process. Threads do have private register values and private stacks segments and these are typically not shared amongst multithreaded processes. □

Exercise 6. Explain the tradeoffs between using multiple processes and using multiple threads.

Proof. The tradeoffs between using multiple processes and using multiple threads is that in multiple processes is that the operating system must schedule a new process before it can service the request, in other terms context switch. While, in multiple threads the main drawback is that the operating system doesn' know about all these threads and will not give more CPU time for a process. □

Exercise 7. Does a multi-threading solution always improve performance? Please explain your answer and give reasons.

Proof. Multi-threading solution doesn't always improve performance. This is due to the fact that the operating system doesn't have visibility over the threads, ultimately leading to inaccurate scheduling decisions. In addition, it could possibly was time context switching to a process where all the threads are block and it will not provide good CPU time to all the threads of a certain process because the operating system doesn't even know it exists. □

Exercise 8. Explain the tradeoffs between preemptive scheduling and non-preemptive scheduling.

Proof. The trade off of preemptive scheduling is that it will interrupt a thread that has not blocked or yielded after a certain amount of time has passed, while the trade off for non-preemptive scheduling is that the operating system will wait until the thread blocks or yields which could also cost time as it won't interrupt the thread until something happens to it. □

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

Proof. User-level threads are threads that the operating system is not aware of while the operating system is aware of kernel-level threads. Another difference is that user-level threads are much faster to switch between since there is no context switch while kernel-level threads are scheduled by the operating system meaning that each thread can be granted its on time by some sort of scheduling algorithm.

In terms of which is preferred, I would recommend if you have CPU-bound tasks that will switch among threads often, then user-level threads would be best to use. Now, if you have a task that has multiple threads that are I/) bound or has many threads, then it would be best to utilize kernel-level threads. □

Exercise 10. What is the difference between a process and a thread. Which one consumes more resources?

Proof. A process is an execution of a program while a thread is a segment of a process. Processes consumes more resources as it is the whole process while a thread is just a segment of a process. □

Exercise 11. The X protocol suffers from scalability problems. How can these problems be tackled?

Proof. It depends on the what type of scalability problem that X protocol has encountered. There is essentially categorical problems, numerical and geographically scalability. For a numerical scalability issue, the problem would lie in that there is too much bandwidth to handle. To remedy this issue, one can utilize compression techniques to reduce the bandwidth. The second problem that could occur for scalability issues is geographically scalability, which can be remedy by utilizing cache techniques to synchronize traffic as much as possible. □

Exercise 12. Read textbook Chapter 1, 2, and 3.

Proof. Below is my GitHub repo for my notes taken during each chapter I have read. If there are any questions or concern, please let me know. - Very Respectfully, Md Ali

https://github.com/xXxSpicyBoiiixXx/CS-550/tree/master/Textbook-Notes □