Discussion Section 2

Distributed Systems
Spring 2014

Problems Chapter 3

Single and Multithreaded Server

Q: ***It takes 15 ms to get a request for work, dispatch it and do the rest of the processing, assuming the data is cached in main memory. If a disk operation is needed (happens ⅓ of the cases) additional 75 ms are needed (the thread sleeps meanwhile). How many requests/s can the server handle if is single-threaded? If it is multithreaded?***

A: Single-threaded server:
Typical access time is: ⅔ * 15 ms + ⅓ * (15 ms+75 ms) = 40 ms
Requests/s: 1000 ms/s * 1/(40 ms/req) = **25 req/s**

Multithreaded server:
Typical access time is: 15 ms
Requests/s: 1000 ms/s * 1/(15 ms/req) ≈ **66.6 req/s**

Problems Chapter 3

Threads and Lightweight Processes

Q: ***Statistically associating only a single thread with a lightweight process is not such a good idea. Why not?***

A: The overhead of creating a lightweight process to handle a single thread dominates the process execution which nullifies the benefits of using threads.

Problems Chapter 3

Processes and Lightweight Processes

Q: ***Having only a single lightweight process per process is also not such a good idea. Why not?***

A: If we do this, we effectively have only user-level threads, which means that any blocking system call will block the entire process.

Problems Chapter 3

Lightweight Processes and Threads

Q: ***Describe a simple scheme in which there are as many lightweight processes (LWP) as there are runnable threads.***

A: Start with a single LWP and let it select a thread. When a thread is found, the LWP creates another LWP to look for another thread to execute, if no thread is found, the LWP destroys itself.

Problems Chapter 3

Object (Process) Migration

Q: ***Strong mobility in UNIX systems could be supported by allowing a process to fork a child on a remote machine. Explain how this would work.***

A: fork() on a local machine

- OS allocates resources to the child process
- Parent process creates a duplicate process.

fork() on a remote machine

- Target OS reserves resources
- Target OS creates an appropriate process and a memory map for the child process
- Parent's image is copied from the source OS into the target's memory and the child is activated.

Problems Chapter 4

Synchronous vs. Asynchronous Communication

Q: Suppose that you could make use of only transient **asynchronous** communication primitives, including only an asynchronous receive primitive. How would you implement primitives for transient **synchronous** communication?

A: To implement a synchronous send primitive, for example, we could make the client send a request and then make it continuously poll for a reply from the server.

Problems Chapter 4

Synchronous vs. Asynchronous Communication

Q: Suppose that you could make use of only transient **synchronous** communication primitives. How would you implement primitives for transient **asynchronous** communication?

A: An asynchronous send could be implemented by making the sender append its message into a buffer that is shared by the process that transfers the message. Each time a client appends a message to the buffer, it wakes the send process which sends the message with a blocking call. The receiver can be implemented by offering a buffer with incoming messages that can be checked by an application.