CS 6/73201 # Homework #2 **Advanced OS**

Due in class on Monday 5 March 2001

1. **(Exercise 3.25 from *Distributed Operating Systems*) In a client-server IPC, a client sends a request message to a server, and the server processes the request and returns the result of the request processing to the client. Is it useful for a process to behave as both a client and server? If no, explain why. If yes, give an example in support of your answer.**

   Yes.

   One answer: this may be necessary in situations where the service provided by the server must be distributed across multiple machines, for performance or geographic reasons. For example, consider a distributed database — a query is made to a specific database server at the home office of a company, but that server must contact several other database servers at regional offices at widely scattered geographic locations to collect all the information requested by the original client. In this case, the first database server is acting as a server to the requesting client, and as a client to the other database servers.

   Another answer: this may be necessary in a situation where a server needs the services of another server of a different type. For example, any arbitrary server may need the services of a file server or name server. In this case, the server is acting as a client when it contacts one of those other servers.

2. **(Exercise 4.7 from *Distributed Operating Systems*) What is a "stub? How are stubs generated? Explain how the use of stubs helps in making an RPC mechanism transparent.**

   A stub is a piece of software that is generated automatically by the RPC system to handle the details of marshalling and unmarshalling messages. Stubs are often generated automatically, meaning the user describes the remote functions using an interface definition language, and the system uses that description to generate the stub code. Using stubs, the user does not have to worry about specific message formats, errors, locating the server, etc., which makes the RPC look very similar to a normal procedure call.

3. **(Exercise 5.4 from *Distributed Operating Systems*) What is false sharing? When is it likely to occur? Can this problem lead to any other problem in a DSM system? Give reasons for your answer.**

   False sharing happens when two processes are accessing two different variables in the same page, making it appear to the system that data in that page is being shared between the two processes when in fact it is not. False sharing is more likely to occur as the page size is increased.

   False sharing can lead to thrashing in a system that support non-replicated migrating pages — the movement of that page back and forth between the two processors accessing the information in it.

4. **(Exercise 6.3 from *Distributed Operating Systems*) Explain why one-time synchronization of the clocks of all the nodes of a distributed system is not sufficient**

and periodic resynchronization is necessary. How will you determine the interval for periodic resynchronization?

Resynchronization is necessary because physical clocks drift over time; for crystal clocks, about one second every 11.6 days.

If the drift rate of a clock is ρ, then after t seconds, two clocks could differ by as much as $2 \rho t$ seconds (one drifts in one direction, and the other drifts in the opposite direction). If the maximum amount that two clocks will be allowed to differ is δ, then $2 \rho t < \delta$, which gives $t < \delta/(2\rho)$. Thus after $\delta/(2\rho)$ seconds, the clocks must be resynchronized.

5. (Exercise 8.19 from *Distributed Operating Systems*) What are the main advantages and disadvantages of using threads instead of multiple processes? Give an example of an application that would benefit from the use of threads and another application that would not benefit from the use of threads.

Advantages: data can be shared between threads, threads are quick and easy to create, etc. *(many advantages — see class notes)*

Disadvantages: must use kernel-level threads for the process to get CPU time proportional to number of threads in the process *(other disadvantages — see class notes)*

Would benefits: application with tasks that can be parallelized or pipelined, applications like servers in particular.

Would not benefit: application that consists of sequential tasks that can not be pipelined or parallelized, such as an application that requires a lot of sequential user I/O.