CS 6/73201 # Homework #1 **Advanced OS**

Due in class at 5pm on Monday 26 September 2005
typed answers preferred

1. **MIMD systems can be classified as either shared-memory multiprocessors or distributed-memory multicomputers. How would these two types of machines be used or programmed differently? What type of applications is each most appropriate for?**

   A shared-memory multiprocessor has quick communication due to the shared memory, so is ideal for communication-intensive applications, though is also limited to a small number of processors by the need to access that shared memory. Programmers code their applications to communicate through the shared memory, although message-passing may also be supported. Example applications include…

   A distributed-memory multicomputer does not have such tight coupling, but is more supportive of heterogeneous systems, or systems that are more geographically distributed. Programmers code their applications to communicate via message-passing.

   (Note that this question did not ask you to define the two types of systems or to explicitly compare their architectures. Your answer should focus on answering the question asked — "how is each used?", "how is each programmed?", and "what type of application is ideal for each?" — rather than just describing the architecture or (worse) listing every fact you know about these systems in random order.)

2. **One distributed system model is the "processor pool" model. What are the features or distinguishing characteristics of this model? What are the challenges involved in designing a system to use this model?**

   The system maintains a "pool" of processors which are dynamically assigned to users and jobs as necessary, without the user needing to choose a particular processor. Processor pool systems are designed to be easily scalable (add new processors to increase overall system performance) and tolerant of failures (e.g., the system works almost as well if a small number of processors fail).

   The challenges include determining which processor is overloaded or underutilized and balancing the job load throughout the system.

3. **When data is broken up into packets to send over a network, either all packets could be sent along the same route, or each could be routed independently. What are the advantages of each of these two methods?**

   Sending all data along the same route has the advantage that the route has to be determined only once, and that packets will generally arrive in the order sent. Further, if the route can be "reserved" for a certain level of bandwidth then there is more assurance that a particular quality of service can be met.

Sending data along different routes has the advantage that it allows later packets to follow a better route if service degrades for some reason along a particular route.

4. **If a server's reply is lost or delayed, it is acceptable to retransmit the request only if the request is idempotent. Give an example of an idempotent request, and one which is not idempotent. Can a non-idempotent request be converted into an idempotent request?**

   Examples of idempotent request: set the value of "total" to 1000, or retrieve the value of "total".

   Example of non-idempotent request: add 100 to the value of "total", send the value "output" to network port 200

   In some cases, a non-idempotent request can be directly converted into an idempotent request (e.g., setting the value of "total" instead of adding to it) but in other cases it can not be converted easily, particularly if the outside world is involved (e.g., the network example above). However, assigning sequence numbers to requests or otherwise maintaining state information about requests previously received and processed can allow many requests to be converted, and embedding the request in a transaction can also help, though transactions have not yet been discussed in this course.

5. **Suppose a set of processors externally synchronize to an authoritative time source. Does that mean the set is now internally synchronized? Explain.**

   Yes and no.

   The argument for yes: If all processors in the set synchronize to the same external source at almost the same time, then for most practical purposes they have the same time and thus are internally synchronized.

   The argument for no: The processors in the set may synchronize at different times or may encounter variations in network delays, in which case they will be roughly internally synchronized, but will not be internally synchronized to the level that an internal synchronization algorithm might achieve.

   (Note that whether the clocks of the processors in the set drift after being synchronized is a separate issue.)