Worksheet 2

Processes and Communication

Group discussion questions

1. Compare the worker pool multi-threaded architecture with the thread-per-request architecture.

2. The usual restriction on *ports* is that it has exactly one receiver but can have many senders. Allowing multiple receivers is complicated, since the processes would need access to the common queue of messages at that port, and means for distributing those messages. Can you identify one possible communication pattern in which this may be too restrictive?

Problem solving questions

3. A file server uses caching, and achieves a hit rate of 80%. File operations in the server cost 5 ms of CPU time when the server finds the requested block in the cache, and take an additional 15 ms of disk I/O time otherwise. Explaining any assumptions you make, estimate the server's throughput capacity (average requests/sec) if it is:

i) single-threaded;

ii) two-threaded, running on a single processor.

4. A request-reply protocol is implemented over a communication service with omission failures to provide *at-least-once* RMI invocation semantics. In the first case the implementor assumes an asynchronous distributed system. In the second case the implementor assumes that the maximum time for the communication and the execution of a remote method is 10ms. In what way does the latter assumption simplify the implementation?

5. A web-based local election service is being implemented in a small town. The *Election* interface provides two remote methods:

vote: with two parameters through which the client supplies the name of a candidate (a string) and the 'voter's number' (an integer used to ensure each user votes once only). The voter's numbers are allocated sparsely from the range of integers to make them hard to guess.

result: with two parameters through which the server supplies the client with the name of a candidate and the number of votes for that candidate.

Which of the parameters of these two procedures are *input* and which are *output* parameters?

6. The *Election* service must ensure that a vote is recorded whenever any user thinks they have cast a vote. Discuss the effect of maybe call semantics on the *Election* service. Would at-least-once call semantics be acceptable for the *Election* service or would you recommend at-most-once call semantics?