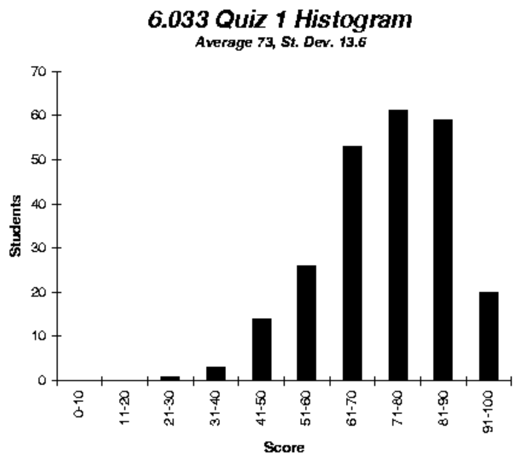
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering Handout 18—March 11, 1997

Quiz 1 Solutions



For Official Use Only

Problem	Your Score	Maximum Score
1		20
2 page 4		6
2 page 5		24
2 page 7		18
2 page 8		32
Total		100

Name: *Official Solutions*

1. (*20 points*) Reading comprehension (multiple choice: select the one best answer to each question, and circle the number to its left).

a. "PC-losering" is *(see Gabriel paper)*

1) AI-lab slang for lowering oneself to the use of a UNIX system.
2) describing losing ideas using politically correct terminology.
3) backing up the saved program counter on a supervisor call.
4) a recurring OS problem in which the appropriate value to be restored into the PC is lost.
5) a term that has not appeared in 6.033 readings assigned so far this term.

b. Hierarchic systems *(see Simon paper)*

1) include artificial but not naturally-occuring systems.
2) exploit structures that incorporate little or no redundancy.
3) are exemplified by the structure of the World Wide Web.
4) are systems whose components are not themselves hierarchic.
5) exhibit a property called "near decomposability".

c. Therac-25 failures were difficult to reproduce in laboratory tests because they depended on *(Therac-25 paper)*

1) the technician's drunken stupor.
2) the absence of a potentiometer which was readily available in the laboratory.
3) the subject's actual cancer symptoms.
4) fast typing by the operator.
5) cosmic rays striking the field-light collimator.

d. The design of X Windows separates policy from mechanism by *(X Windows paper)*

1) the use of icons rather than names.
2) a policy of free distribution of source code.
3) support for big-endian as well as little-endian machines.
4) symmetric support for all platforms.
5) the demotion of the window manager to ordinary-program status.

Name: *Official Solutions*

e. Congestion is said to occur in a store-and-forward network when

1) communication stalls due to cycles in the flow-control dependencies.

2) the throughput demanded of a network link exceeds its capacity.

3) the volume of email received by each user exceeds the rate at which users can read email.

4) the buffering demanded of a network node exceeds its capacity.

5) the amount of space required to store routing tables at each node becomes burdensome.

(Halstead's notes on networking)

2. (*80 points*) Quarria is a new country formed on a 1 Km rock island in the middle of the Pacific Ocean. The founders, recent 6.033 dropouts, have organized the Quarria Stock Market in order to get the economy rolling. The stock market is very simple, since there is only one stock to trade (that of the Quarria Rock Company). Moreover, due to local religious convictions, the price of the stock is always precisely the wind velocity at the highest point on the island.Rocky, Quarria's president, proposes that the stock market be entirely network based. He suggests running the stock market from a server machine, and requiring each investor to have a separate client machine which makes occasional requests to the server using a simple RPC protocol. The two requests Rocky proposes supporting are

   Balance() - requests that the server return the cash balance of a client's account. This service is very fast, requiring a simple read of a memory location.

   Trade(nshares) - requests that *nshares* be bought (assuming *nshares* is positive) or -nshares be sold (if *nshares* is negative) at the current market price. This service is potentially slow, since it potentially involves network traffic in order to locate a willing trade partner.

   Quarria implements a simple RPC protocol in which a client sends the server a request packet of the format

```
struct Request {
   int Client;        /* Unique code for the client */
   int Opcode;        /* Code for operation requested */
   int Argument;      /* integer argument, if any */
   int Result;        /* integer return value, if any*/
};
```

   and the server replies by sending back the same packet, with the Result field changed. We assume that link- and network-layer error checking detect and discard garbled packets, and that Quarria investors are scrupulously honest; thus any received packet was actually sent by some client (although sent packets might get lost).

   a. Is this RPC design appropriate for a connectionless network model, or is a connection-based model assumed?

   Yes, it is appropriate for a connectionless network model; each request is completely self-contained and doesn't depend on a "connection" for its context.

The client RPC stub blocks the client thread until a reply is received, but includes a timeout provision allowing any client RPC operation to return with a TIMEOUT error code if no response is heard from the server after Q seconds.

b. In a single sentence, give a reason for preferring returning a timeout error code over simply having the RPC operation block forever.

The latter approach would cause the client to hang forever even on transient errors (lost packets); the timeout code allows the client to dictate a recovery procedure.

c. In a single sentence, give a reason for preferring returning a timeout error code over having the RPC stub transparently retransmit the packet.

The latter approach would cause the client to hang forever on a permanent failure (eg, server crash); the timeout code allows the client to dictate a recovery procedure.

d. Suppose you can bound the time taken for a request, including network and server time, at 3 seconds. What advantage is there to setting the timeout, Q, to 4 seconds instead of 2 seconds?

Setting Q above the time bound for a request/response guarantees that no retransmission will occur unless the original request, or its response, was actually lost.

(Note that this doesn't eliminate the need for at-most-once server semantics, since the server may still see duplicate requests...)

Unfortunately, no such bound exists for Quarria's network.

e. Q. What complication does client packet retransmission introduce into the RPC semantics, in the absence of a time bound?

In general, retransmission makes packet duplication possible. In the absence of a time bound, duplicate responses may be seen by the client.

Name: ***Official Solutions***

Rocky's initial implementation of the server is as follows:

```
#define N 1000              /* Total number of clients*/
int Cash[N],                /* Cash balance of each client */
    Shares[N];              /* Stock owned by each client*/

Server()
{
    struct Request *req;        /* Pointer to request packet*/
    while (1)  {                /* loop forever...*/
        req = GetNextRequest();/* take next incoming request,*/
        if (req.Opcode==1)      /* ... and dispatch on opcode.*/
            Balance(req);       /* Request 1: return balance*/
        if (req.Opcode==2)      /* Request 2: buy/sell stock*/
            Trade(req);
    }
}

/* Process a Balance request.. */
Balance(struct Request *req)
{
    int client=req->Client;         /* Get client number from request */
    req->Result = Cash[client];     /* Return his cash balance*/
    Reply(req);                     /* and send the reply.*/
}

/* Perform a trade: buy/sell Argument/-Argument shares of
* stock, and return the total number of shares owned after
* trade. */
Trade(struct Request *req)
{
    int client = req->client;   /* The client who is requesting*/
    int p=StockPrice();         /* Price, using network requests*/
    int nshares=req->Argument;  /* Number of shares to buy/sell*/
    int actual;
    /* See how many shares we can trade */
    actual = ConfirmTrade(req, p,nshares);

    Cash[client] = Cash[client]+p*actual;/* Update our records*/
    Shares[client] = Shares[client]+actual;
    req->Result = actual;
    Reply(req);
}
```

Note that ConfirmTrade uses network communication to check on available shares, executes the trade, and returns the number of shares that have actually been bought or sold.

Name: ***Official Solutions***

Rocky tests this implementation on a single server machine by having clients scattered around the island sending Balance requests as fast as they can. He discovers that after some point adding more clients doesn't increase the throughput----the server throughput tops out at 1000 requests/second.

f. Rocky is concerned about performance, and hires you to recommend steps for improvement. Which, if any, of the following steps might significantly improve Rocky's measured 1000 **Balance** requests/sec?

Y___ N_X_: Use faster client machines.

Y___ N_X_: Use multiple client threads (each making Balance requests) on each client.

Y_X_ N___: Use a faster server machine.

Y_X_ N___: Use faster network technology.

(Since server is already flooded with client requests, bottleneck is server and/or network)

Stone Galore, a local systems guru, has another suggestion to improve the performance generally. He proposes multithreading the server, replacing calls to service procedures like

```
   Balance(req);            /* Run Balance, to service request */
with
   NewThread(Balance,req);/* Fork thread to run Balance(req)*/
```

The NewThread primitive creates a new thread, runs the supplied procedure (in this case Balance) in that thread, and deactivates the thread on completion. Stone's thread implementation is preemptive.

Stone changes the appropriate three lines of the original code according to the above model, and re-tries the experiment with Balance() requests. He now measures a maximum server throughput of 500 requests/second.

g. Explain, in a single sentence, the performance degradation.

The degradation is due to thread overhead (eg, costs associated with thread creation, deactivation, and context switching).

h. Is there an advantage to the use of threads in other requests? Explain, in a single sentence.

Yes; it allows Trade requests, which may involve long (blocking) steps in their execution, to be interleaved. Server time otherwise wasted on I/O waits can be profitably used on other requests, improving throughput.

Name: ***Official Solutions***

i. Select the best advice for Rocky regarding server threads:

A. Don't use threads; stick with your original design.

B. Don't use threads for Balance requests, but use them for other requests.

C. Continue using them for all requests; the benefits outweigh the costs.

(pretty much implied by last two answers...!)

Independently of your advice, Stone is determined to stick with the multithreaded implementation.

j. Should the code to Trade be changed to reflect the fact that it now operates in a multithreaded server environment? Explain, suggesting explicit changes as necessary.

Because the client is single-threaded, no server changes are necessary since at most one client request can be processed at any time and server variables are not shared among requests from different clients. (One ought to check ConfirmTrade for potential locks, but we did not ask for it.)

k. What if the client is multithreaded and can have multiple request outstanding? Should the code to Trade be changed? Explain, suggesting explicit changes as necessary.

Under these circumstances, the body of Trade should be made atomic, e.g. via lock/unlock pair surrounding the critical 3 lines. Preferably, each client would use a separate lock to minimize contention.

Rocky decides that multithreaded clients are complicated and abandons that idea.

Rocky left MIT just before the lecture on RPC, and isn't sure whether his server requires **at-most-once** RPC semantics.

l. Which of the requests require at-most-once RPC semantics? Explain, in a single sentence.

Trade requires at-most-once, since it causes a change in server state. Since Balance doesn't change state, at-most-once is not needed for it.

m. Suggest, in a sentence or two, how one might modify Rocky's implementation to guarantee at-most-once semantics. You can ignore the possibility of crashes, but should consider lost packets and retransmissions.

Add a sequence number, supplied by the client, to the request. The client increments sequence number on new requests but not on retransmission, and the server keeps a table of past responses which it uses for responses to subsequent requests with identical sequence numbers.

THE END

Name: ***Official Solutions***