Paper 3F6: Software Engineering and Design

Distributed Systems Design

Examples Paper 3

Straightforward questions are marked †
Tripos standard (but not necessarily Tripos length) questions are marked ∗

CORBA

1. A stocks trading broker wishes to provide an online service to their customers. The service must provide customers with the facility to

   - obtain the price of any given share identified by its share code
   - to trade (buy or sell) a number of shares
   - obtain the balance of their cash account with the brokers

   The server must check that the customer does not sell shares they do not own and that they do not buy shares they do not have money for. The service is to be implemented using CORBA.

   (a) Design an interface for the trading system, showing your design as an `idl` file.

   (b) The company now wishes to extend the system so that the server can give statistical information about shares. Customers must now be able to:

      - obtain the average price for a share over the previous n days (where n is given by the user).
      - obtain the price to earnings ratio for a share.

      Give a new `idl` file which shows how these facilities can be incorporated into the system without breaking programs that only use (and know about) the existing facilities.

   (c) Many customers will be using the broker's facilities concurrently.

      i. Identify the information that must be stored on the server for each customer.

      ii. Describe how the factory design pattern can be used to keep customers' transactions separate, drawing a UML diagram to show the modified design.

2. ∗ A virtual tourism project wishes to offer users the capability to obtain a 3D model of a tourist attraction from a server.

   As the user moves through the virtual world defined by the model, they are presented with hot-spots at certain locations. Clicking on these hot-spots causes media such as an image, movie or sound clip to be retrieved from the server and presented to the user.

   (a) Design an interface for this system, showing your answer as an `idl` file. Assume that there is a pre-defined `Model` datatype, which stores a 3D model, and `Image`, `Movie` and `Soundclip` types for the media.

   (b) It is found that the model takes a long time to download. In order to reduce the time that users have to wait before viewing, the model is to be broken down into zones which can be obtained from the server as they are needed. The 3D model for a zone now defines which zones adjoin the current zone, and when a new zone will need to be loaded. When the user approaches the boundary of the current zone, their client software can thus request the next zone from the server.

   Show what changes to the interface are required to support this new scheme.

   (c) The popularity of the system grows, and the main server becomes overloaded. To resolve this, the design needs changing to allow zones and their associated media to be hosted on different CORBA servers elsewhere in the world. Write a new `idl` file showing a design for this revised system.

   (d) Some zones become very popular and the servers that provide those zones become overloaded. What could be done to allow load-balancing, so that process of requesting a zone file from a server allows the server to delegate the responsibility for supplying that zone data to any one of multiple computers?

   (e) Currently, each user moves around in the virtual world in isolation from every other user. What would be required to make it possible to see other users who are visiting the same zone and to send messages to them?

Transaction Processing

3. (a) Explain what is meant by an optimistic and a pessimistic concurrency control scheme.

   (b) State whether an optimistic or a pessimistic policy is likely to produce a more efficient system in each of the following cases:

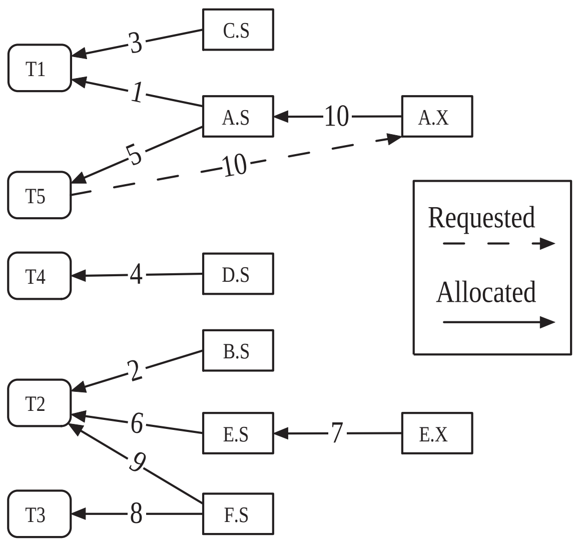
      i. A flight booking system;
      ii. A police criminal database;
      iii. A database maintaining patient records.

   In each case, give reasons for your answer.

4. ∗ The following list records a sequence of actions invoked by a set of 12 transactions, $T1 \dots T12$ operating on database accounts $A, B, \dots, H$. Each operation `Q.read` must acquire a shared lock $Q.S$ on account Q, and each write operation `Q.write` must acquire an exclusive lock $Q.X$ on account Q. Assume that all locks once acquired are held until the transaction either commits or aborts.

time	transaction	action	time	transaction	action
1	T1	`A.read`	19	T9	`G.write`
2	T2	`B.read`	20	T8	`E.read`
3	T1	`C.read`	21	T7	`Commit`
4	T4	`D.read`	22	T9	`H.read`
5	T5	`A.read`	23	T3	`G.read`
6	T2	`E.read`	24	T10	`A.read`
7	T2	`E.write`	25	T9	`H.write`
8	T3	`F.read`	26	T6	`Commit`
9	T2	`F.read`	27	T11	`C.read`
10	T5	`A.write`	28	T12	`D.read`
11	T1	`Commit`	29	T12	`C.read`
12	T6	`A.read`	30	T2	`F.write`
13	T5	`Abort`	31	T11	`C.write`
14	T6	`C.read`	32	T12	`A.read`
15	T6	`C.write`	33	T10	`A.write`
16	T7	`G.read`	34	T12	`D.write`
17	T8	`H.read`	35	T4	`G.read`
18	T9	`G.read`	36		

(a) Verify that the resource allocation graph shown below accurately represents the state of resource allocation after step 10 just before $T1$ commits.



Resource graph after step 10

(b) construct a resource allocation graph for each of the remaining 4 phases of processing i.e. upto steps 12, 20, 25 and 35.

(c) Identify any deadlocks that occur.

5. The following code illustrates an attempted solution to the mutual exclusion problem which does not rely on operating system primitives such as semaphores:

```
bool c1 = true;
bool c2 = true;

void thread1 {
   while(true) {
      while(!c2);
      c1 = false;
      critical_section
      c1 = true;
      non_critical_section
   }
}

void thread2 {
   while(true) {
      while(!c1);
      c2 = false;
      critical_section
      c2 = true;
      non_critical_section
   }
}
```

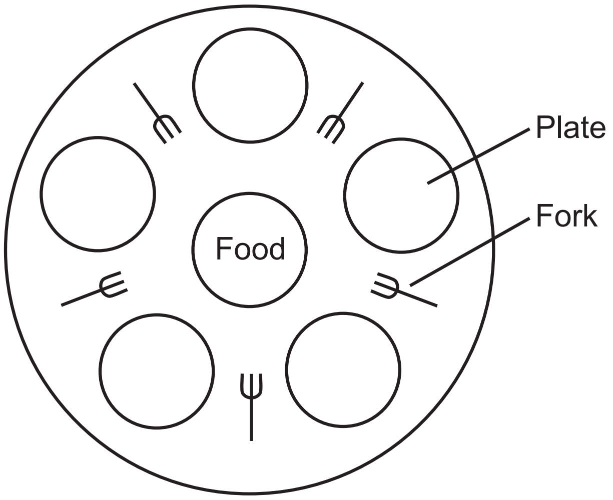
where the functions `thread1` and `thread2` are executed as parallel threads.

(a) In what way does this fail to provide a satisfactory solution?

(b) Does changing the body of both threads to the following form (by swapping the first two lines after each `while(true)` line as shown below) result in a satisfactory solution? Explain your reasoning.

```
while(true) {
   cN = false;
   while(!cM);
   critical_section
   cN = true;
   non_critical_section
}
```

6. ∗ An abstract problem used in the analysis of concurrent algorithms is that of the Dining Philosophers. In a room there is a round table with a never-ending supply of food in the centre and five plates and five forks arranged as shown below.



There are five philosophers whose life is a cycle described by

```
void philosopher(int philosopherID)
{
   while(true)
   {
      think();
      eat();
   }
}
```

Before he can eat, a philosopher must enter the room, sit at a free plate and pick up the two forks adjacent to it. Once finished, he replaces the forks and leaves the room.

Unlike the Boolean semaphores used in lectures, an integer semaphore `s` has an associated count N which is used to control access to N units of resource. Each function call `secure(s)` will reduce the count by one until it is zero, at which point the calling thread is suspended. Each function call `release(s)` will either cause a waiting thread to be resumed, or if there are none, the count is incremented by one.

Using the integer semaphores defined by

```
integer_semaphore sfork[5];   // i.e. one for each fork on the table
integer_semaphore sroom;
```

modify the `philosopher()` function so that the philosophers can eat; ensure mutual exclusion (i.e. no two philosophers are trying to use the same fork at the same time); avoid deadlock (all the philosophers are waiting for something to happen); and avoid starvation (one or more of the philosophers starve to death). Define the initial count to be given to each semaphore.

7. A simple Boolean semaphore object is used to control access to a critical section as in,

```
semaphore s;

s.enter();
critical_section
s.leave();
```

Due to poor system design and a complex nested call structure, it was found that in some parts of the system, a thread was attempting to reuse the same semaphore from within the critical section, i.e. it was attempting to execute:

```
semaphore s;

s.enter()
...
s.enter();
critical_section
s.leave();
...
s.leave();
```

and as a result, the system deadlocked on the second call to `s.enter()`. Since the nested calls to `s.enter() ... s.leave()` are logically harmless, it was considered simpler to define a new semaphore object which allowed this behaviour, rather than redesign the system. Design a new semaphore object which provides a wrapper around the existing semaphore object, and which makes the above behaviour harmless.

Tom Drummond
February 2010