CS 5204 Sample Midterm

This is an open-book, open-note, closed-internet, closed-cellphone and closed-computer exam.

1. RPC.

An RPC architecture, such as OMG's CORBA, performs a number of services or tasks that allow a client A to invoke a procedure provided by a server object B. A key goal is to provide transparency, i.e., that client A should be unaware of where server object B is located.

a) Give two examples of tasks or services that an RPC system does *not* have to perform if A and B happen to be located on the same machine, but in different processes.
b) Give two examples of tasks or services that an RPC system does *not* have to perform if A and B happen to be located within the same process.

2. Eraser.

Consider the following code fragment. Recall the naïve version of lockset refinement (i.e., without considering read-sharing and read-write sharing separately.) Consider the sets locks_held(t) for a thread t and the candidate lockset C(x) for a shared variable x.

a) Complete the following table, adding entries whenever locks_held(t) or C(x) changes. Write down the value of the respective set *after* the execution of the corresponding statement in the left-most column.

	locks_held(t)	candidate set C(x)

mutex mu1, mu2;
lock(mu2);
if (x == 0) {
  lock(mu1);
  unlock(mu2);
  if (x == 0) {
  x = 5;
  }
  unlock(mu1);
} else {
  unlock(mu2);
}
	∅	{mu1, mu2}

b) Will Eraser flag a race condition? Say why or why not.

3. Logical Clocks.

Consider a system of two processes that are exchanging messages. Ignore internal events for the purposes of the problem: all events of interests are either sending or receiving of messages.

a) Construct a scenario (timeline) in which for two events a and b the following is true. (Show your work.)
   - *a* < *b* according to the total order < provided by Lamport timestamps with process-id tiebreaker.
   - *a* and *b* are concurrent, that is, neither $a \rightarrow b$ nor $b \rightarrow a$ is true.

b) Now assume the processes used vector clocks instead. Give the vector timestamps for *a* and *b* and explain how they show *a* || *b*.

4. Proportional Share Scheduling (VTRR).

Nieh's paper defines the service time error of a client A during an interval (t_1, t_2)

$$E_A(t_1,t_2) = W_A(t_1,t_2) - (t_2 - t_1)\frac{S_A}{\sum_i S_i}$$

Assume a system with only two clients, A and B, which are both runnable during (t_1, t_2). Express B's service time error $E_B(t_1,t_2)$ as a function of $E_A(t_1,t_2)$ (Show your work.)

Solution

1. RPC.

c) Examples of tasks an RPC system typically does, but does not have to do when both parties are on the same machine, include:

   a. No conversion between different data representations (big-endian, little-endian) has to be done.
   b. No need to setup a transport layer connection between client and server.
   c. No need to retransmit requests to make sure server received the request (depending on semantics of RPC.)

d) Examples of tasks or services that an RPC system does *not* have to perform if A and B happen to be located within the same process.

   - No need for a client-stub or server-stub.
   - No need to create or dispatch a server thread.
   - No need to marshal or unmarshal the arguments.

2. Eraser.

a)

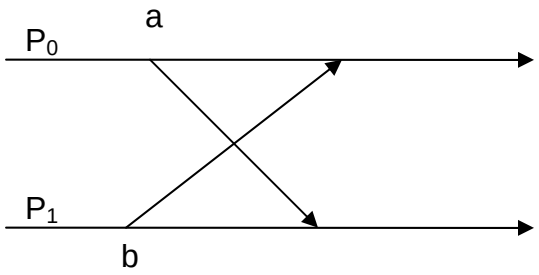
		locks_held(t)	candidate set C(x)

1	mutex mu1, mu2;	∅	{mu1, mu2}
2	lock(mu2);	{mu2}	
3	if (x == 0) {		{mu2}
4	lock(mu1);	{mu1, mu2}	
5	unlock(mu2);	{mu1}	
6	if (x == 0) {		∅
7	x = 5;		∅
8	}		
9	unlock(mu1);	∅	
10	} else {		
11	unlock(mu2);	∅	
12	}		

b) Eraser will flag a race condition because C(x) is empty in line 6 (and 7).

3. Logical Clocks.

a) Because Lamport+tiebreaker will always order events *a* and *b*, *any* timeline that shows two concurrent events will work. In particular, picking event *a* as the very first message sent from P_0 to P_1 and event *b* as the very first message sent from P_1 to P_0 will result in two concurrent events.



Both *a* and *b* have Lamport timestamps 0 since they are the first event in their process, so in Lamport+tiebreaker event *a* is expressed as tuple (0, P_0) whereas *b* is (0, P_1). We have $a < b$ since (0 == 0 && 0 < 1). However, *a* and *b* are concurrent.

b) The vector timestamps of both *a* and *b* is [0 0] because they are the first event in their process, and that process has not received any message from the other process. The vector timestamps show that *a* and *b* are concurrent because we have $\neg([0\ 0]_a < [0\ 0]_b \vee [0\ 0]_b < [0\ 0]_a)$, that is, neither of them is in at least one component strictly larger, and in the other components larger or equal than the other. (In fact, they're equal in this case.)

4. Proportional Share Scheduling (VTRR).

Nieh's paper defines the service time error of a client A during an interval (t_1, t_2)

$$E_A(t_1,t_2) = W_A(t_1,t_2) - (t_2 - t_1)\frac{S_A}{\sum_i S_i}$$

Assume a system with only two clients, A and B, which are both runnable during (t_1, t_2). Express B's service time error $E_B(t_1,t_2)$ as a function of $E_A(t_1,t_2)$

"Only two clients, both are runnable" means that $t_2 - t_1 = W_A(t_1,t_2) + W_B(t_1,t_2)$
Therefore,

$$\begin{aligned}
E_B(t_1,t_2) &= W_B(t_1,t_2) - (t_2 - t_1)\frac{S_B}{\sum_i S_i} \\
&= (t_2 - t_1) - W_A(t_1,t_2) - (t_2 - t_1)\frac{S_B}{\sum_i S_i} \\
&= -W_A(t_1,t_2) + (t_2 - t_1)(1 - \frac{S_B}{\sum_i S_i}) \\
&= -W_A(t_1,t_2) + (t_2 - t_1)\frac{\sum_i S_i - S_B}{\sum_i S_i} \\
&= -\left(W_A(t_1,t_2) - (t_2 - t_1)\frac{S_A}{\sum_i S_i} \right) \\
&= -E_A(t_1,t_2)
\end{aligned}$$

This derivation shows that this definition of service time error preserves the intuitively desirable invariant that the service time error of B is the negative of the service time error of A: whatever A gets too much or too little, B gets too little or too much.