15-440/15-640: Homework 1

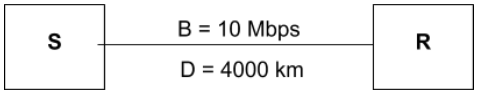
Due: September 23, 2018 11:59pm

Name: _____

Andrew ID: _____

1. Networking (20 Points)

Part A (10 points)



Consider the situation depicted in the figure above. The sender S want to send a file of size $S = 1Gbit$ to the receiver R over a direct link of bandwidth $B = 10Mbps$ and of length $D = 4000km$, where packets travel at $\frac{2}{3}$ the speed of light ($c = 3 \times 10^8 m/s$). The file is subdivided into packets of size $P = 5Kbit$ and a stop and wait protocol is used to ensure correct delivery.

(a) Suppose the client sends one packet at a time, then waits for an ACK before delivering another packet. How long will it take to transfer the whole file? What is the effective bandwidth? **[5pts]**

(b) To ensure optimal bandwidth usage, we have to make sure that the sender is never idle i.e. there are always packets being written. The sender is now allowed to transmit up to N unacked packets at a time, and then has to wait for an ACK before transmitting the next packet. Note that ACKs are for individual packets; they are not cumulative. What is the minimal value of N that ensures optimal bandwidth usage? **[5pts]**

Solution:

(a) 1 Packet takes: $2\frac{3D}{2c} + \frac{P}{B} = 40.5ms$ where $2\frac{3D}{2c}$ is the RTT and $\frac{P}{B}$ is the transmission delay.
The total time is $\frac{S}{P} * (2\frac{3D}{2c} + \frac{P}{B}) = 2.25h$ The effective bandwidth is $5kb/44.03ms = 123.45kbps$

(b) We need to ensure that the link always in usage, which occurs when we transfer `BW * Latency` $= B * 2\frac{3D}{2c}$. So the optimal value is $N = \frac{B*2\frac{3D}{2c}}{P} = 80$.

Part B (10 Points)

(a) For each of the following scenarios, state whether you would use TCP or UDP. Briefly explain your reasoning. **[1pt each]**

i. A high performance video streaming application.

ii. A banking application

iii. An online video game.

iv. Server-residing cache using broadcast invalidations (server broadcasts message to multiple clients that their cache is invalid for a particular key).

(b) State and explain one advantage and one disadvantage of the "IP waist" in the hourglass model. **[3pts]**

(c) We saw in lecture that web page retrieval involves several steps, which could potentially blow up the latency. What solution is employed to deal with this issue? Briefly explain. **[3pts]**

Solution: Accept answer as long as explanation makes sense.

(a) i. UDP: Performance is very important. Lost frames are mostly fine (lag/stutter).

ii. TCP: Data integrity is crucial in the application.

iii. UDP: Performance is critical and loss of packets, or out of order packets, is tolerable. Game will lag/stutter but continue onward.
TCP: "Financial" portions of the games (microtransactions, ...) cannot tolerate loss of data.

iv. UDP: Allows multicasting; can build another protocol on top of UDP for reliability.
TCP: Don't want clients to read stale data so we need guaranteed delivery.

(b) Advantage: It provides a clear layer of abstraction that upper-level and lower-level protocol can use to guarantee portability and reusability. New transport-layer and data-link-layer protocols can be created and swapped in/out.
Disadvantage: It makes it hard to introduce new network layer protocols like IPv6.

(c) Caching is the main mechanism that allows us to reduce latency. Instead of making several remote requests that could take hundreds of milliseconds, cached data is accessed locally, which speeds up the retrieval process.

2. Concurrency (15 Points)

Part A (6 Points)

```
 1.   C.Init():
 2.     C.mutex = NewMutex()
 3.     C.readCvar = NewCond(C.mutex)
 4.     C.writeCvar = NewCond(C.mutex)
 5.     C.value = undefined
 6.     C.full = false
 7.     return
 8.
 9.   C.send(value):
10.     C.mutex.lock()
11.     If C.full {
12.       C.writeCvar.wait()
13.     }
14.     C.full = true;
15.     C.value = value;
16.     C.readCvar.signal()
17.     mutex.unlock()
18.
19.   C.receive(value):
20.     C.mutex.lock()
21.     If (not C.full) {
22.       C.readCvar.wait()
23.     }
24.     C.full = false
25.     value = C.value
26.     C.writeCvar.signal()
27.     mutex.unlock()
```

Identify the bugs in this code. For each of them clearly state the line number, the reason why it is a bug, and how you would fix it (*Hint: some values are being overwritten, while others are being read multiple times*)

Solution: L11 and L21 should be while loops. For L11, another sender can fill up the channel after the signal is received but before the lock is acquired. The current sender will think that the channel is empty, which leads to an overwrite. For L21, a similar situation can occur. The current reader can think that the channel is empty and re-read the value stored in C.

Part B (9 Points)

(a) Explain the main benefit of using condition variables instead of spin locks. **[4pts]**

(b) What is the main difference between a livelock and a deadlock. **[2pts]**

(c) Consider a Producer/Consumer pattern with is working on a fixed number of data items. Do you anticipate that a correct implementation uses semaphores or mutexes? **[3pts]**

Solution:

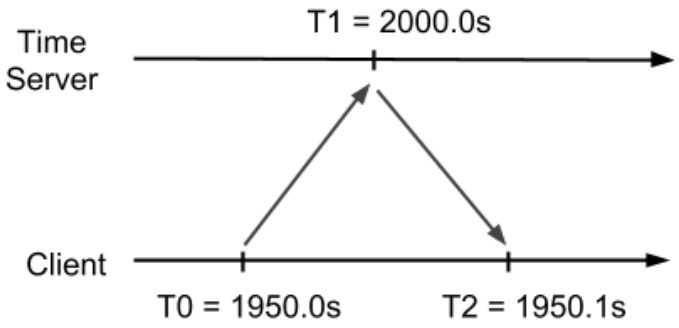
(a) Spin locks repeatedly acquire a lock, check for a condition and, if that condition is not met, release the lock. They use up not only consume CPU time without doing any useful work, but also slow down other threads by continually acquiring the lock. By using condition variables, the thread is put to sleep until the condition is met. During this time, it does not use the CPU and does not acquire the lock, which improve performance.

(b) In the deadlock, none of thread makes progress because they are all waiting for each other to release some resources. In a livelock, the threads do not make progress even though they are not blocked. They just do not do any useful work.

(c) A good implementation will use semaphores to control access to the queue of work items, because of the fixed number of items. In addition, a good implementation may use mutexes (see the synchronization lecture). So, either "semaphores" or "both" are accepted solutions. Just "mutexes" isn't enough and won't lead to an efficient Producer/Consumer Pattern.

3. Time & Synchronization (25 Points)

Part A (9 Points)



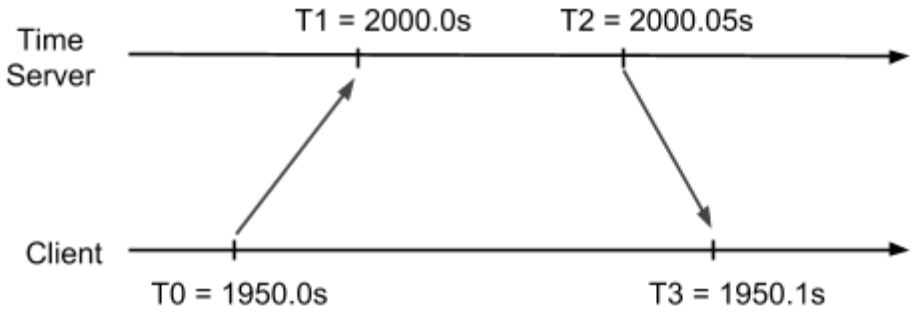
Consider the scenario depicted above. The system guarantees that the minimum one way delay between the client and the server is 25ms.

(a) How will the client update its local time according to Cristian's algorithm? **[1pt]**

(b) Keeping $T0$ and $T2$ constant, what is the best case error of Cristian's algorithm? When does this situation arise? **[1pt]**

(c) Keeping $T0$ and $T2$ constant, what is the worst case error of Cristian's algorithm? When does this situation arise? **[2pts]**

(d) In a more realistic setting, the time server would not be able to immediately respond to the client's request. We therefore have the setting below. Determine the estimated network RTT and time offset. **[3pts]**



(e) How would you implement the function that updates the local time on the client? Describe your reason in 2 sentences. **[2pts]**

Solution:

(a) The client set local $t = T1 + \frac{T2-T0}{2} = 2000.05s$

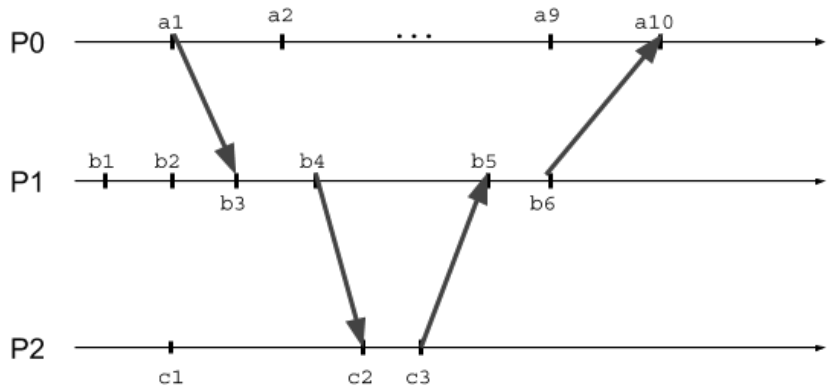
(b) Best case error = 0. This occurs when the communication is perfectly symmetrical; the two delays are the same.

(c) Worst case error $= \frac{T2-T0}{2} - minDelay = 50ms - 25ms = 25ms$. Occurs when one of the delays is equal to minDelay

(d) RTT $= (T3 - T0) - (T2 - T0) = 50ms$
Offset $= \frac{(T1-T0)+(T3-T2)}{2} = (50s + 49.95s)/2 = 49.975s$

(e) Setting t = t + offset might create inconsistencies when the offset is negative. Instead of using the naive update method, we can just change the rate of the clock. This way, time is always moving forward.

Part B (16 Points)



(a) Consider the situation depicted above. Events a2, ..., a9 are all local to the process P0. Fill in the following table using the various algorithms described in class. Assume that P0's id is 0, P1's is 1 and P2's id is 2. **[14pts]**

Event	Lamport's Algorithm	Totally Ordered Lamport Clock	Vector Clock Algorithm
a1			
a2			
a9			
a10			
b1			
b2			
b3			
b4			
b5			
b6			
c1			
c2			
c3			

(b) What are the advantages and disadvantages of using Lamport Clocks instead of Vector Clocks? State at least one advantage and one disadvantage. **[2pts]**

Solution:

(a)

Event	Lamport's Algorithm	Totally Ordered Lamport Clock	Vector Clock Algorithm
a1	1	3	(1, 0, 0)
a2	2	6	(2, 0, 0)
a9	9	27	(9, 0, 0)
a10	10	30	(10, 6, 3)
b1	1	4	(0, 1, 0)
b2	2	7	(0, 2, 0)
b3	3	10	(1, 3, 0)
b4	4	13	(1, 4, 0)
b5	7	22	(1, 5, 3)
b6	8	25	(1, 6, 3)
c1	1	5	(0, 0, 1)
c2	5	17	(1, 4, 2)
c3	6	20	(1, 4, 3)

(b) Pros: Simpler implementation, less data transfer.
Cons: Have to apply rules of transitivity to determine the happens before relationship since V(e) $<$ V(e') does not imply e $\rightarrow$ e'.

4. Remote Procedure Calls (15 Points)

Part A (10 Points)

Consider the following code snippet.

```
struct Node {
  int value;
  Node* next;
}

void processList(Node *head);
```

The processList function is very computation heavy, so clients delegate the work to a more powerful backend server using an RPC.

(a) When implemented as a local procedure, processList only needs to copy an 8 bytes pointer to the head of the list. How will this change when it comes to implementing the function as an remote procedure? What drawbacks can arise? **[5pts]**

(b) Provide a high level description of how you would marshall this list on the client, and of how you would unmarshall it on the server. **[5pts]**

Solution:

(a) Pointers are meaningless when it comes to RPC because different processes do not share an address space. Therefore, the whole data structure needs to marshalled on the client side and unmarshalled on the server side. This can create an issue when the data that needs to be transfered gets larger and larger.

(b) There are several possible solutions here. Here is a simple method.
To marshall the list: pack the values into an array. First send the size of the array, and then send the array itself.
To unmarshall the list: Read the first 4 (or 8) bytes to get the size of array N. You can then read $N * \texttt{sizeof(int)}$ bytes to recreate the array. Finally, create a new list containing the values and execute the function.

Part B (5 Points)

For each of the following scenarios, state whether you would use an at-most-once semantics or an at-least-once semantics. Explain in no more than 2 sentences. **[1pt for each]**

(a) Booking a flight from a travel agency

(b) Retrieving the value associated with a key

(c) Withdrawing cash from bank account

(d) Sending a message/post on social media

(e) Registering for a class at CMU

Solution:

(a) at-most-once: because we don't want to get charged twice for a single airline ticket.
at-least-once: very unlikely, but exceptionally good justifications could be accepted.

(b) at-least-once, because retrieving a value should be an idempotent process.

(c) at-most-once, because bank account balance could be deducted twice or more for a single withdrawal
at-least-once, very unlikely, but exceptionally good justifications could be accepted.

(d) at-least-once, because user expects their message to be delivered/posted, and they can always delete extraneous messages.

(e) at-least-once, because student expects to be registered for the class and registering twice has no negative effect.

5. Distributed File Systems (25 Points)

(a) Briefly state 2 advantages of using NFS over AFS. **[2pts]**

(b) Briefly state 2 advantages of using AFS over NFS. **[2pts]**

(c) For each of the following scenarios, choose whether a file system like Coda, a file system like LBFS, or neither would be a better fit. Briefly justify your decision. **[1pt for each]**

i. An application that allows multiple users to collaborate on a single document.

ii. Operating on a network with frequent disconnects

iii. Limited network resources (High latency)

iv. An application where, overall, the data being sent has a lot of redundancy.

(d) Briefly explain the CAP theorem (2 sentences, max). **[3pts]**

(e) You are put in charge of designing an interplanetary distributed file system meant to be used on Earth and Mars. The clients on Mars are rovers that execute software from the file system to decide where to go. The rovers will take pictures and gather data at regular intervals and upload these to the file system as well. The software on the file system will constantly be updated by developers on Earth and the rovers will apply these updates whenever they come. However, compiling and loading the new software takes several hours and the rover cannot move or gather any data during this time.

i. In case of a network partition, is 'consistency' or 'availability' a more desirable property of this file system? Briefly explain your decision. **[3pts]**

ii. Based on your answer from (i.) would optimistic replication or pessimistic replication be a better choice for this file system? **[2pts]**

iii. Would you implement client-side caching for this file system? Briefly justify your decision. **[4pts]**

iv. It turns out a lot of the images on Mars are very similar. How can this fact be used to optimize the file system? Briefly explain your answer. **[5pts]**

Solution:

(a) Simple implementation (serverless state)
Cost-effective/"thin" clients (no hard-drive necessary)

(b) Scalability (less server load thanks to client-side caching)
Performance (client-side caching on hard-drive)
Transparency (server keeps tracks of volumes; on NFS, clients would have to remount)
Can still perform well if network has high-latency

(c) i. Neither, because Coda/LBFS are both weakly/eventually consistent.

ii. Coda, because Coda allows disconnected operation due to its hoarding pre-fetch process.

iii. LBFS, because it uses 90% less bandwidth.

iv. LBFS, because it breaks up files into chunks and utilizes chunk redundancy across files to reduce network traffic.

(d) The CAP theorem states that, in the presence of network partition - a distributed file system cannot have both Consistency and Availability.

(e)
i. Availability, because the rovers should continually be moving and taking pictures, even if it is running outdated instructions.

ii. Optimistic replication would be a better choice, because availability is a priority over consistency.

iii. Servers can specify reasonable TTLs for cache items, so that developers can continually update their software and the rover will only check the file system for new updates periodically.

iv. Some form of chunking can be used, similar as LBFS, so redundant bytes don't have to be sent over the network for multiple similar images. Also, a strong compression algorithm can be used to reduce the overall file size of each image.