Introduction

1. Pros and cons of distributed vs. parallel systems?
   1. Distributed systems
      1. Reliable – other nodes serve as backups for every other node
      2. Available – If one node is occupied, will be directed to unoccupied node
      3. Require synchronization systems (MPI)
      4. Communicate and interact using RPC
   2. Parallel systems
      1. Shared memory
      2. Global clock that all processes share – enables SIMD and MIMD
      3. Processes can communicate on the order of TBs
      4. High performance
2. Discuss the difference between the workstation-server and the processor-pool model from the availability viewpoint
   1. If one of the mini-computers in workstation-server goes down, the process that computer handled will become offline
3. Discuss the difference between the processor-pool and the cluster model from the performance viewpoint.
   1. Cluster model focuses more on performance. Can utilize multiple nodes
   2. Cluster model shares storage. Cluster model appears as one computer. All requests must pass through master node who redirects (hierarchical)
   3. Processor pool wastes less resources. Each user is only allocated the necessary number of processors
4. Discuss about pros and cons of Microkernel
   1. Pros – even if the kernel fails, it won’t take everything down with it
   2. Cons – very hefty. Can’t pick and choose

Networking

1. Why do we need layered network protocols?
2. When implementing TCP with datagram, what do we have to take care of?
3. Compare UDP and TCP for the implementation of each of the following application-level or presentation-level protocols (textbook p142, Q3.7):
   1. Telnet
   2. FTP
   3. Rwho, finger
   4. HTTP
   5. RPC or RMI

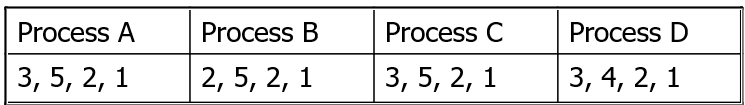
IPC

1. Consider the pros and cons of polling and interrupt in non-blocking communication.

|  |  |
| --- | --- |
| Polling | Pros:   * + - Non-blocking     - No high level responsiveness     - No additional function   Cons:   * + - CPU is used to constantly check if server/client is available     - Have to constantly swap processing power between checking if new inputs were received and computing the inputs themselves |
| Interrupt | Pros:   * + - Non-blocking     - Can split computation and checking into two separate threads. Don’t have to split   Cons:   * + - High level responsiveness (when computing, doesn’t automatically check if new inputs are received     - Has to manually check interrupt between)     - Function to jump in C++     - Multithreading     - A context switch is needed to handle the interrupt.       * The application the user decides will write the comp() and handling() functions. The transition between each function is handled by the OS       * The OS will know whether a socket is available. If so, will start the program again from the handling function       * User -> kernel -> user |

1. Consider an example inducing an accidental system hang-up (named a deadlock) in no-buffering communication.
   1. Left hand right hand example. Both client and server are unable to receive data until their data has been sent and received. Use multiple threads. One to receive one to send.
2. Which of the following operations are idempotent?
   1. cin >> data;
   2. ifstream infile(“input.txt”); infile.seek( );
   3. cout << data;
   4. int a = 1, b = 2, c; c = a + b;
   5. int c = 1; c++;

MPI

1. Explain, in atomic multicast on page 6, why reversing the order of operations “all receivers forward the message to the same group and thereafter deliver it to themselves” makes the multicast no longer atomic.
2. Assume that four processes communicate with one another in causal ordering. Their current vectors are show below. If Process A sends a message, which processes can receive it immediately?
3. Show that, if the basic multicast that we use in the algorithm of P9 is also FIFO-ordered, then the resultant totally-ordered multicast is also causally ordered.
4. Consider pros and cons of PVM’s daemon-based and MPI’s library linking-based message passing.
5. Why can MPI maintain FIFO ordering?

RMI

1. The caller process of an RPC/RMI must wait for a reply from the callee process after making a call. Explain how this can actually be done.
2. Which types of server did you implement for the programming assignments 3 and 4, a stateful or a stateless server? Then, why did you implement such a type of server?
3. Discuss the similarities and differences among the following parameter passing:
   1. Call-by-object-reference
   2. Call-by-move
   3. Call-by-visit
4. Discuss about the pros and the cons of dynamic binding in RPC.
5. Textbook p227, Q5.12: Discuss the invocation semantics that can be achieved when the request-reply protocol is implemented over a TCP connection, which guarantees that data is delivered in the order sent, without loss or duplication. Take into account all of the conditions causing a connection to be broken.
6. Textbook p227, Q5.14: 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?
7. Textbook p227, Q5.15: 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 T. In what way does the latter assumption simplify the implementation?
8. Textbook p228, Q5.22: A client makes remote procedure calls to a server. The client takes 5 milliseconds to compute the arguments for each request, and the server takes 10 milliseconds to process each request. The local operating system processing time for each send or receive operation is 0.5 milliseconds, and the network time to transmit each request or reply message is 3 milliseconds. Marshalling or unmarshalling takes 0.5 milliseconds per message.

Calculate the time take by the client to generate and return from two requests:

* 1. (i) if it is single-threaded, and
  2. (ii) if it has two threads that can make requests concurrently on a single processor.
  3. You can ignore context-switching times.

OS

1. Textbook p333, Q7.7: Explain the advantage of copy-on-write region copying for Unix, where a call to fork is typically followed by a call to exec. What should happen if a region that has been copied using copy-on-write is itself copied?
2. Why can threads perform their context switch faster than processes?
3. What are the thread groups? What are the daemon threads? How can those contribute to the multithreaded server?
4. If you comment out the following statement from the C++ code on Slide 11, what change will you observer in the execution?
   1. pthread\_join( child, NULL );
5. If you comment out the following statement from the Java code on Slide 12, what change will you observe in the execution?
   1. child.join( );
6. Textbook p333, Q7.8: A file server uses caching, and achieves a hit rate of 80%. File operations in the server costs 5ms of CPU time when the server finds the requested block in the cache, and take an additional 15ms of disk I/O time otherwise. Explaining any assumptions you made, estimate the server’s throughput capacity (average requests/sec) if it is:
   1. Signle-threaded;
   2. Two-threaded, running on a single processor;
   3. Two-threaded, running on a two-processor computer.
7. Textbook p333 Q7.16: Network transmission time accounts for 20% of a null RPC and 80% of an RPC that transmits 1024 user bytes (less than the size of a network packet). By what percentage will the times for these two operations improve if the network is upgraded from 10Mbps to 100Mbps.
8. The following server code assumes that each client program asks three user inputs:
   1. the id of a file to operate on : 0 or 1, each corresponding “file0” and “file1”
   2. a file operation type: ‘r’ as a file read or ‘w’ as a file write
   3. if the file operation is ‘w’, a 100-byte message to be read from a keyboard.

The client sends those inputs to the server through a socket. If the file operation type is ‘r’, it receives a 100-byte content of the corresponding file from the server and prints it out.

The server spawns two child threads: child\_thread[0] and childe\_thread[1], each associated with socket descriptor sd[0] and sd[1] respectively. Every time the server accepts a new socket request from a client, it receives a file id from the client, and passes this socket request to the corresponding thread. The thread opens its file, checks a file operation type, and reads/writes the file according to the type.

1. Which server model was used in this tcp.cpp program, worker pool, per-request threads, per-connection threads, per-object threads, or pipeline model? Choose one of these models and justify your choice.
2. The server in the above program is not so scalable. In other words, it can’t handle many client requests concurrently. Why? Explain the reason.
3. Which server model(s) are scalable? If you change this program into such server model(s), what side effect will occur? Describe the major problem you have conceived.

int main( int argc, char \*argv[] ) {

int sd = NULL\_FD; // socket descriptor

char id = 0; // a file (thread) id

if ( argc == 1) { // I'm a server

pthread\_t child[2];

for ( int i = 0; i < 2; i++ ) { // create two child threads

sd[i] = NULL\_FD;

pthread\_create( &child[0], NULL, thread\_func, (void \*)&i );

}

while ( true ) { // keep receiving a client socket request

if ( ( sd = sock.getServerSocket( ) ) == NULL\_FD )

return -1;

read( sd, &id, 1 ); // receive a file (thread) id

while ( sd[id] != NULL\_FD )

; // wait for cihld\_thread[id] to be ready

sd[id] = sd; // pass this client socket to the thread

}

}

if ( argc == 2 ) { // I'm a client

if ( ( sd = Sock.getClientSocket( argv[1] ) ) == NULL\_FD )

return -1;

cout << "Operate on file0 or file1? (type 0 or 1): "; cin >> id;

write( sd, id, 1 ); // send a file (thread) id to server

cout << "Operation type? (type r or w): "; cin >> op;

write( sd, &op, 1 ); // send a file operation type to server

if ( op == 'w' ) { // if it's a write operation

cout << "message: "; cin >> message; // read a user message

write( sd, message, SIZE ); // send it to the server

}

if ( op == 'r' ) { // if it's a read operation

read( sd, message, SIZE ); // receive a message from server

cout << message << endl; // display it

}

}

}

#include "Socket.h"

#include "pthread.h"

#define PORT 10000

#define SIZE 100 // message size

int sd[2]; // socket descriptor

void\* thread\_func( void \*arg ) { // thread to read and write a given file

int id = \*(int \*)arg; // my thread id

char fileName = (id == 0) ? "file0" : "file1"; // if I'm 0, operate on file0

while ( true ) {

if ( sd[id] == NULL\_FD ) // check if a new socket request came to me.

continue;

int fd = open( fileName, O\_RDWR );// open my file

char op; // a file operation type: 'r' or 'w'

read( sd[id], &op, 1 ); // read an operation type from the socket

if ( op[id] == 'r' ) { // a read operation

read( fd, message, SIZE ); // read 100 bytes from my file

write( sd[myId], message, SIZE ); // send them back to the client

} else if ( op[id] == 'w' ) { // a write operation

read( sd[myId], message, SIZE ); // receive 100 bytes from the client

write( fd, message, SIZE ); // write them down to my file

}

close( fd ); // close my file

close( sd[Id] ); // close this socket conneciton

sd[Id] = NULL\_FD; // inform the server that I'm ready

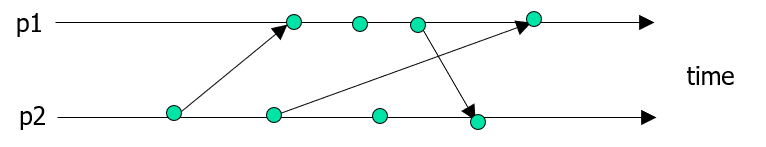
}

}

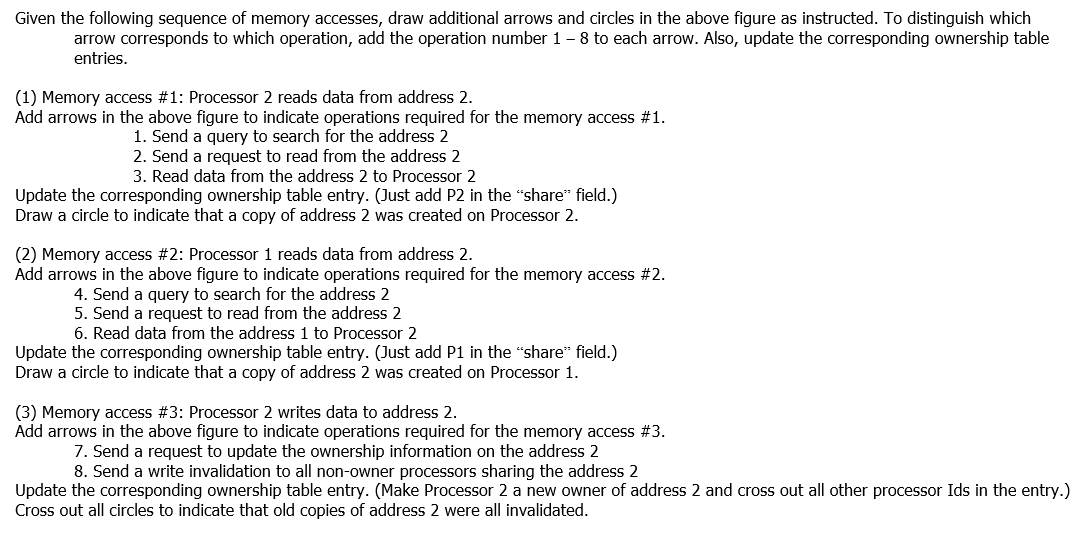
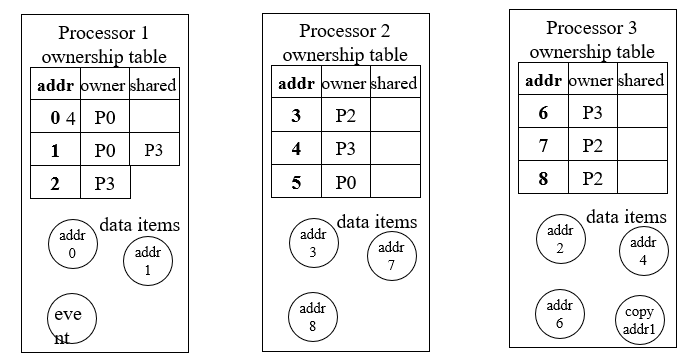
Migration

1. What are items to be taken into consider when migration decision is made?
2. Why is address space transfer so difficult when pointers are involved? What is necessary to transfer data structures on memory to the destination?
3. Why do mobile agents have more security concerns than process migration.
4. What made it possible to implement java-based mobile agents?
5. Why do java-based mobile agents need to take weak migration

Synchronization

1. Textbook p627, Q14.7: An NTP server B receives server A’s message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B’s timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.
2. Textbook p628, Q14.14: Two processes P and Q are connected in a ring using two channels, and they constantly rotate a message m. At any one time, there is only one copy of m in the system. Each process’s state consists of the number of times its has received m, and P sends m first. At a certain point, P has the message and its state is 101. Immediately after sending m, P initiates the snapshot algorithm. Explain the operation of the algorithm in this case, given the possible global state(s) reported by it.
3. Textbook p429, Q14.15: The figure below shows events occurring for each of two processes, p1 and p2. Arrows between processes denote message transmission. Draw and label the lattice of consistent states (p1 state, p2 state), beginning with the initial state (0,0).

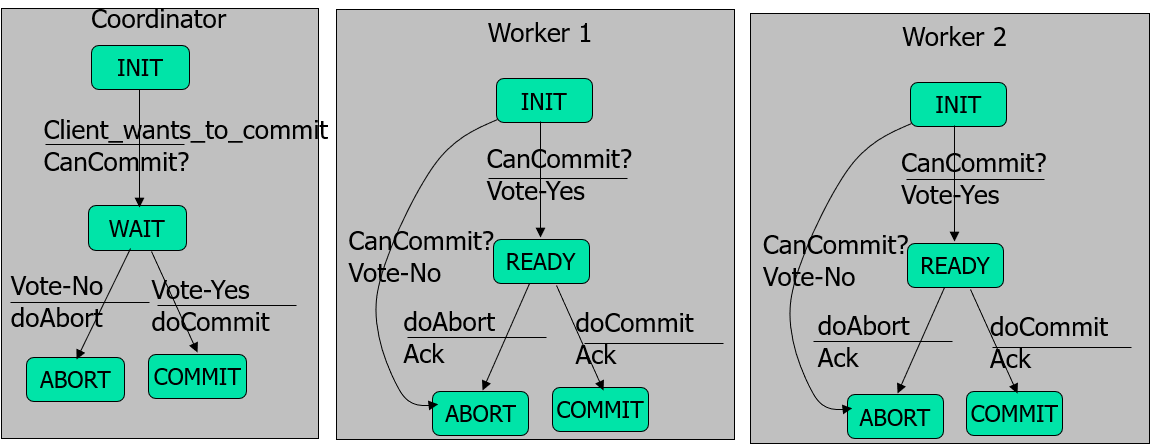
DSM

1. Is the memory underlying the following execution of two processes sequentially consistent (assuming that, initially, all variables are set to zero)? P1: R(x)1; R(x)2; W(y)1
   1. P2: W(x)1; R(y)1; W(x)2
2. Show that the following history is not causally consistent.
   1. P1: W(a)0; W(a)1
   2. P2: R(a)1; W(b)2
   3. P3: R(b)2; R(a)0
3. Explain the relationship between false sharing and data granularity in DSM.
4. There is a DSM system that is based on the write-invalidation protocol, uses a fixed distributed-server algorithm for locating a given data item, and consists of three processors such as 1, 2, and 3. Each processor has the following data items and an ownership/sharing-processor table.

DFS

1. In transaction-like semantics a.k.a. concurrency control, compare the pros and cons of backward and forward transactions. In particular, consider the case where each transaction includes more read than write operations.
   1. Backward transaction
   2. Pros:
   3. Cons:
   4. Forward transaction
   5. Pros:
   6. Cons:
2. Answer the following five questions about file-caching. When you are asked to show which systems use a given caching scheme, choose all applicable systems from NFS, AFS, xFS and Plan9.
3. Why can file-caching contribute to performance improvement? Answer two reasons.
   1. Reason 1:
   2. Reason 2:
4. Q2-2. State one merit for using server-side caching. Which system uses server-side-caching?
   1. Merit:
   2. System: Plan9
5. Client-side caching allows multiple clients to cache the same file. There are two schemes to validate the contents of a locally-cached file (or invalidate the contents of the same file cached at remote clients.) Those are client-initiated and server-initiated validations. Does the client-initiated validation require a file server to be stateful? Justify your answer. Also show which systems use the client-initiated validation.
   1. Stateless or stateful?
   2. Reason:
   3. Systems: NFS, Plan9
6. Does the server-initiated validation require a file server to be stateful? Justify your answer. Also show which system uses the server-initiated validation.
   1. Stateless or stateful?
   2. Reason:
   3. System: AFS, xFS

Replication

1. The following state transition diagram describes the two-phase commitment protocol. Let’s assume that worker1 crashed when a coordinate sent a commit message. Trace this diagram. To be specific, make appropriate dashed arrows “thick and solid arrows” with your pen or pencil.
2. Textbook p762, Q17.1: In a decentralized variant of the two-phase commit protocol, the participants communicate directly with one another instead of indirectly via the coordinator. In phase 1, the coordinator sends its vote to all the participants. In phase 2, if the coordinator’s vote is No, the participants just abort the transaction; if it is Yes, each participant sends its vote to the coordinator and the other participants, each of which decides on the outcome according to the vote and carries it out. Calculate the number of messages and the number of rounds it takes. What are its advantages or disadvantages in comparison with the centralized variant?
3. Textbook p816, Q18.10: Explain why allowing backups to process read operations directly, (i.e., without contacting a primary), leads to sequentially consistent rather than linearizable executions in a primary-copy replication.
4. Textbook p816, Q18.11: Could the gossip architecture be used for a distributed computer game as describe below?
5. The players move figures around a common scene. The state of the game is replicated at the players’ workstations and at a server, which contains services controlling the game overall, such as collision detection. Updates are multicast to all replicas.
6. The quorum-based replication protocol can address network partition problems. Why didn’t Coda use this protocol? Explain the reason.
7. What if a message is lost in ISIS group communication? Describe a solution.