CS252 Final Review

Please aswer this final review and return it during the final exam.

If you write down the solution in HTML and turn it in before 1:00pm the day before the final exam, you will get up to 40 extra points in the final exam. The submitted solutions will be posted.

Login to lore and type:

```
mkdir hw2
copy your files to hw2
turnin -c cs252 -p hw2 hw2
```

1. Complete the procedure runCommand*(command, outputBuffer, bufferSize)* that executes a command in a different process and stores its output in outputBuffer. *command* is the name of the program with no arguments. See how main uses *runCommand*(). *runCommand* will return 0 on success or -1 otherwise. Hint: Use a pipe to communicate the parent and the child process running runCommand(). Have the parent read from the pipe and write into the outputBuffer.

```
int
runCommand( char * command, char * outputBuffer, int maxBufferSize)
{
```

```
int defaultin = dup(0);
int defaultout = dup(1);
int pipefd[2];
int ret;

if (pipe(pipefd) == -1)
{
   perror("pipe");
   return -1;
}

dup2(pipefd[1], 1);
close(pipefd[1]);
ret = fork();
if (ret == 0)
{
   close(pipefd[0]);
   char * execCommands [2];
   execCommands[0] = command;
   execCommands[1] = NULL;
   execvp(execCommands[0], execCommands);
   perror("execvp");
   _exit(1);
}
```

```
else if (ret > 0)
{
   dup2(pipefd[0], 0);
   close(pipefd[0]);

   char c = 0;
   int i = 0;
   while (i < maxBufferSize && (c = read(0, &c, sizeof(c))) != -1)
   {
      outputBuffer[i] = c;
      i++;
   }
}

else
{
   perror("fork");
   dup2(defaultin, 0);
   close(defaultin);
   dup2(defaultout, 1);
   close(defaultout);
   return -1;
}

dup2(defaultin, 0);
close(defaultin);
dup2(defaultout, 1);
close(defaultout);

waitpid(ret, 0, 0);
return 0;
```

```
}

int
main()
{
    // The output of "ls" will be stored in buffer
    char buffer[ 1024 ];

     if ( runCommand( "ls", buffer, 1024 ) < 0 ) {
       perror("runCommand" );
       exit( -1 );
    }

    printf( "ls: %s\n", buffer );

    exit( 0 );
}
```

2. Add the necessary code to the insert() and removeFirst() functions to make them synchronized. removeFirst() will have to wait if the list is empty. insert() will have to wait if there are already 20 elements in the list. **Use semaphores**. Add also the variables you need.

```
struct List {
  int val;
  int next;
};

struct List * head = NULL;

// More variables
```

```
pthread_mutex_t mutexLock;
sema_t fullSem;
sema_t emptySem;
```

```
main()
{

 // DO any initializations here
```

```
pthread_mutex_init(&mutexLock, NULL);
sema_init(&fullSem, 20, USYNC_THREAD, NULL);
sema_init(&emptySem, 0, USYNC_THREAD, NULL);
```

```
}

void insert( int val )
{
```

```
sema_wait(&fullSem);
```

```
 List tmp = new List;

 tmp->val = val;
```

```
pthread_mutex_lock(&mutexLock);
```

```
 tmp->next = head;

 head = tmp;
```

```
pthread_mutex_unlock(&mutexLock);
sema_post(&emptySem);
```

```
}

Struct List * removeFirst()
{
```

```
sema_wait(&emptySem);
pthread_mutex_lock(&mutexLock);
```

```
  List tmp = head;

  head = tmp->next;
```

```
sema_post(&fullSem);
pthread_mutex_unlock(&mutexLock);
```

```
  return tmp;

}
```

3. From lab3, assuming you have a procedure *void dispatchHTTP(int slaveSocket)* that processes the request and closes *slaveSocket,* write the loop server code for a) iterative server, b) concurrent server using fork, c) concurrent server creating a thread after each request, and d) pool of threads, in the procedures indicated. Each procedure receives as argument the master socket already initialized and ready to be used inside accept.

```
void iterativeServer( int masterSocket) {
```

```
while (1)
{
   struct sockaddr_in clientIPAddr;
   int alen = sizeof(clientIPAddr);
   int slaveSocket = accept(masterSocket, (struct sockaddr
   *)&clientIPAddr, (socklen_t *)&alen);
   if (slaveSocket < 0)
   {
      perror("accept");
      exit(1);
   }

   dispatchHTTP(slaveSocket);
}
```

}

void forkServer(int masterSocket) {

```
while (1)
{
   struct sockaddr_in clientIPAddr;
   int alen = sizeof(clientIPAddr);
   int slaveSocket = accept(masterSocket, (struct sockaddr
   *)&clientIPAddr, (socklen_t *)&alen);
   if (slaveSocket < 0)
   {
      perror("accept");
      exit(1);
   }

   int ret = fork();
   if (ret == 0)
   {
      dispatchHTTP(slaveSocket);
      exit(0);
   } else if (ret < 0)
   {
      perror("fork");
      exit(1);
   }
   close(slaveSocket);
}
```

}

void poolOfThreads(int masterSocket) {

```
int i = 0;
pthread_t threads[4];
for (; i < 4; i++)
{
   pthread_create(&threads[i], NULL, loopthread, (void *)masterSocket);
}
loopthread((void *)masterSocket);
```

}

void createThreadForEachRequest(int masterSocket) {

```
while (1)
{
    struct sockaddr_in clientIPAddr;
    int alen = sizeof(clientIPAddr);
    int slaveSocket = accept(masterSocket, (struct sockaddr
    *)&clientIPAddr, (socklen_t *)&alen);
    if (slaveSocket < 0)
    {
       perror("accept");
       exit(1);
    }

    pthread_attr_t attributes;
    pthread_attr_init(&attributes);
    pthread_attr_setdetachstate(&attributes, PTHREAD_CREATE_DETACHED);
    pthread_create(NULL, &attributes, dispatchHTTP, slaveSocket);
 }
```

```
}
```

```
// Other procedures
```

```
void * loopthread(void * socket)
{
   int masterSocket = (int)socket;
   while (1)
   {
      struct sockaddr_in clientIPAddr;
      int alen = sizeof(clientIPAddr);
      int slaveSocket = accept(masterSocket, (struct sockaddr
      *)&clientIPAddr, (socklen_t *)&alen);
      if (slaveSocket < 0)
      {
         perror("accept");
         exit(1);
      }

      dispatchHTTP(slaveSocket);
   }
}
```

4. Implement a R/W lock class.

RWLock.h

```
class RWLock
{
    int _nreaders;
    sema_t _semAccess; mutex_t _mutex;
public:
    RWLock();
    void readLock();
    void writeLock();
    void readUnlock();
    void writeUnlock();
};
```

RWLock.cpp

```
RWLock::RWLock()
{
   _nreaders = 0;
   pthread_mutex_init(&_mutex);
   sema_init(&_semAccess);
}

void RWLock::readLock()
{
   pthread_mutex_lock(&_mutex);
   _nreaders++;
   if (_nreaders == 1)
   {
      sema_wait(&_semAccess);
   }
   pthread_mutex_unlock(&_mutex);
}

void RWLock::writeLock()
{
   sema_wait(&_semAccess);
}

void RWLock::readUnlock()
{
   pthread_mutex_lock(&_mutex);
   _nreaders--;
   if (_nreaders == 0)
   {
      sema_post(&_semAccess);
   }
   pthread_mutex_unlock(&_mutex);
}

void RWLock::writeUnlock()
{
   sema_post(&_semAccess);
}
```

5. What are the four parameters that a computers needs to be able to get connected to the internet and what are they used for?

```
IP address - Current IP address of machine
Subnet mask - Used to send packets to hosts in the same LAN
Default router - Used to send packets to hosts outside LAN
DNS Server - Used to resolve host names into IP addresses

_
```

6. How does a computer know when it can deliver a packet directly and when it has to pass a packet to a router?

```
By referring to a routing table and doing a bitwise-and operation  between
the destination IP to send packet to and the subnet mask of each route
entry. If it matches a target network, it will send to the router
specified in the entry. If the target network is local, packet is sent
directly. Otherwise, it is sent to the default router.

_
```

7. What does ARP mean and how does it work?

```
Address Resolution Protocol.
It converts an IP address into a hardware address when a router or host
delivers a packet directly. It accepts an IP address of a computer in
locally connected network and outputs the Ethernet address of the
computer. The bindings (IP address, hardware address) are kept in an ARP
table or an ARP cache.

_
```

8. What does DNS mean and what it is used for?

```
Domain Name Server.
It is a service that resolves host names into IP addresses. It uses a
distributed lookup algorithm and contacts as many servers as necessary.

_
```

9. What does DHCP mean and how does it work?

```
Dynamic Host Configuration Protocol.
It is a server that assigns each computer the IP address, subnet mask, DNS
server, and default router automatically when it is connected to a
network. Therefore, an administrator is not needed to configure all these
```

```
parameters manually.
_
```

10. What does UDP mean?

```
User Datagram Protocol.
It is an unreliable protocol and message-oriented. It does not require a
connection to send messages. Each message is encapsulated in an IP
datagram and its size is limited to the network's MTU. It is mainly used
for broadcasting and for real-time data applications.
_
```

11. What does TCP mean? What are the 6 features of TCP?

```
Transmission Control Protocol.
1. Adaptive retransmission - retransmission time is set to RTT + 4*RTTVar
so that it works for all networks
2. Cumulative acknowledgements - acknowledgment is sent for total bytes
received instead of for each packet
3. Fast retransmission - when duplicate acknowledgment is received,
retransmissions are sent even before timer expires
4. Flow control - sender slows down when receiving buffer is full. Window
of buffer size is sent in acknowledgment
5. Congestion control - during congestion, sender will use "Slow Start",
then "Congestion Avoidance" where window of retransmission data is reduced
in size
6. Reliable connection and shutdown - uses Three way handshake. Three
packets are enough to make sure that lost packets or host crashes will not
interfere future connections
_
```

12. When should you use TCP and when should you use UDP?P?

```
TCP is used when a reliable connection is needed. UDP is needed
when a computer needs to broadcast messages to multiple computers
or when it needs to use real-time data applications where data needs
to be sent without delay. Retransmissions will add to the delay so
TCP is slow for those applications.
_
```

13. What does NAT stand for? Assume that a packet <A, 4563, X, 80> is sent from a host behind a NAT box to a webserver X. Describe the steps for the translation (6 steps) since it goes from the host A, through the NAT box, to X and then back from X to the NAT box to A.A.

```
Network Address Translation.
1. Host A wants to establish connection with webserver X in the Internet.
2. NAT box is default router so A sends TCP packet to it.
3. NAT box chooses a random unused port (1234) and substitutes the source
port with it. It also changes the source IP into its own IP address (B).
Packet (B, 1234, X, 80) then sent to X.
4. The old source IP (A) and port (4563) are added into the NAT table for
use when packets come back.
5. When packet is back from X, NAT box looks up the destination port
(1234) in NAT table and substitute the destination IP and port with
original values (A, 4563).
6. Packet is forwarded to A. Everything will be transparent to both ends.
The mapping will be used until connection is closed or timed out.
_
```

14. Explain why NAT boxes can be used as firewalls to prevent unwanted connections. Also explain why it is not normally possible to run web servers behind a firewall and how this problem can be solved.d.

```
NAT box only passes packets if connection is initiated inside the network.
If packet arrives to NAT box and does not have existing entry
in NAT table, packet will be discarded. This will protect the private
network from hacker attacks. It is often not possible to run
webservers behind a firewall because connections will start from outside
the network and there is no mapping in NAT table yet.
Static entries can be added manually to the NAT table so that NAT box
knows where to forward the incoming packets.
_
```

15. Write a simple client program "echo-client host port string" that sends a string "string" followed by "\r\n"to "host : port" and then it reads the server's response and prints it to stdout.t.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
```

```
#include <assert.h>

int main(int argc, char ** argv)
{
   if (argc != 4)
   {
      printf("Usage: echo-client [host] [port] [string]");
      exit(1);
   }

   char * host = argv[1];
   int port = atoi(argv[2]);
   assert(port >= 0);
   char * echostring = argv[3];
   struct sockaddr_in socketAddress;

   memset((char *)&socketAddress, 0, sizeof(socketAddress));
   socketAddress.sin_family = AF_INET;
   socketAddress.sin_port = htons((u_short)port);

   struct hostent * ptrh = gethostbyname(host);
   assert(ptrh != NULL);
   memcpy(&socketAddress.sin_addr, ptrh->h_addr, ptrh->h_length);

   struct protoent * ptrp = getprotobyname("tcp");
   assert(ptrp != NULL);
   int socket = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
   assert(socket >= 0);

   if (connect(socket, (struct sockaddr *)&socketAddress,
   sizeof(socketAddress)) < 0)
   {
      perror("connect");
      exit(1);
   }

   write(socket, echostring, strlen(echostring));
   write(socket, "\r\n", 2);

   char buf[1024];
   int n = recv(socket, buf, sizeof(buf), 0);
   while (n > 0)
   {
      write(1, buf, n);
      n = recv(socket, buf, sizeof(buf), 0);
   }

   close(socket);
   return 0;
}
```

16. Write a simple iterative server "echo-server port" that waits for incoming requests in "port" and once it receives a string delimited by "\r\n" it will reply with the same string plus "\r\n" and close the connection.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <assert.h>

int main(int argc, char ** argv)
{
   if (argc != 2)
   {
      printf("Usage: echo-server [port]");
      exit(1);
   }

   int queueLength = 5;
   int port = atoi(argv[1]);
   assert(port >= 0);

   struct sockaddr_in socketAddress;
   memset((char *)&socketAddress, 0, sizeof(socketAddress));
   socketAddress.sin_family = AF_INET;
   socketAddress.sin_addr.s_addr = INADDR_ANY;
   socketAddress.sin_port = htons((u_short)port);

   int masterSocket = socket(PF_INET, SOCK_STREAM, 0);
   assert(masterSocket >= 0);

   int optval = 1;
   int err = setsockopt(masterSocket, SOL_SOCKET, SO_REUSEADDR, (char
   *)&optval, sizeof(int));
   assert(err >= 0);
   err = bind(masterSocket, (struct sockaddr *)&socketAddress,
   sizeof(socketAddress));
   assert(err >= 0);
   err = listen(masterSocket, queueLength);
   assert(err >= 0);

   while (1)
```

```
    {
        struct sockaddr_in client;
        int alen = sizeof(client);
        int clientSocket = accept(masterSocket, (struct sockaddr *)&client,
        (socklen_t *)&alen);
        assert(clientSocket >= 0);

        const int maxLength = 1024;
        char buf[maxLength + 1];
        char newC, lastC;
        int curLen = 0;
        int n;

        while (curLen < maxLength && (n = read(clientSocket, &newC,
        sizeof(newC))) > 0)
        {
            buf[curLen] = newC;
            curLen++;
            if (lastC == '\r' && newC == '\n')
            {
                break;
            }
            lastC = newC;
        }
        buf[curLen] = '\0';

        write(clientSocket, buf, strlen(buf));
        close(clientSocket);
    }
    return 0;
}
```

17. Enumerate 5 of the 12 questions in "Joel's Test".

```
1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
```

18. What is XP programming?

```
Extreme Programming.
It is a practical methodology for software development. It has a list of
rules thathave been proven successful
in software development. It encourages an iterative approach.
_
```

19. From XP Programming, mention 4 items from the Planning List, 4 Items from the Coding List, 4 Items from the Designing List, and 4 Items from the testing List.t.

```
Planning
- User stories
- Release planning
- Frequent small releases
- Measure project development

Coding
- Have customers available
- Follow coding standards
- Code unit tests first
- Use pair programming

Designing
- Keep it simple
- Refactor
- Use CRC cards
- Choose system metaphor

Testing
- All code must have uni tests
- All code must pass uni tests before integration
- When a bug is found, create a test
_
```

20. Explain 5 uses of the source control system.em.

```
- Keep track of changes by multiple programmers.
- Merge code easily.
- Evaluate level of contribution of others.
- Backup of sources.
_
```

21. Describe the advantages and disadvantages of centralized vs. distributed source control systems.ms.

```
Centralized - Advantages:
- One centralized, common source.
- Easy to obtain latest source code.
Centralized - Disadvantages:
```

```
- If server crashes, no one can do anything

Distributed - Advantages:
- Programmers can work for a long time without submitting to common code.
- Easy to make changes since everyone has a local repository.
Distributed - Disadvantages:

_
```

22. Describe the 4 types of tests, who writes thoses tests in the organization, and when do they run.un.

```
1. Unit tests
- Group of tests that test a specific class.
- Written by programmers.
- Run after writing a class. Sometimes even before writing any code.

2. System tests
- Test a specific subsystem of product or multiple classes involved in a
feature.
- Written by QA and programmers.
- Run in daily build.

3. Regression tests
- Test written after bug is fixed to verify that it doesn't break anything
else.
- Written by programmers.
- Run after a bug is fixed.

4. Acceptance tests
- Evaluate quality of software before it is released to tell whether it is
ready for prime time.
- Written by QA.

_
```

23. Explain why it is importan to have a bugtrack system. m.

```
To keep track of all the bugs that are reported and to assign
priority and severity to them. This will make sure that the software has a
reduced number of bugs and improve the quality of the software.

_
```

24. Explain the difference between Priority and Severity in a bug.ug.

```
Priority is how important is the bug to the software developers
and the organization. Severity is how the bug affects the users.
```

25. Mention 5 cases when you can apply refactoring.ng.

```
- Long method
- Long parameter list
- Duplicated code
- Large class
```

26. What is a Software Pattern, what are the parts of a software pattern? What is the name of the book that introduced software patterns and the authors?rs?

```
Software patterns are reusable design solutions for reoccuring problems.
The parts:
- Pattern name
- Synopsis
- Context
- Forces
- Solution
- Consequences
- Implementation
- Related patterns
```

27. Describe the Proxy Pattern and 2 applications.ns.

```
Proxy pattern is to force method calls to an object to occur through
a proxy object.
- Act as logger object
- Act as a cache
```

28. Describe the Command Pattern and two applications.ns.

```
Command pattern encapsulates commands in objects so you can control
selection and sequencing, queue, undo and manipulate them.
- Graphical editor that has do/undo
- Word processor that has do/undo
```

29. What is the difference between Code Instrumentation Profiling and Statistical Sampling Profiling.ng.

```
Code instrumentation profiling adds instructions into the code before
a method starts and after a method returns to measure the execution
of a method. Statistical sampling samples the program at regular intervals
to find out what the program is doing.
_
```

30. Explain why Optimizing should be left until the very end in the software cycle and why you should use an execution profiler before attempting to optimize a program.ram.

```
Optimizing should be left to the very end because it makes the code
difficult to read and understand. Execution profiler should be used
to find out which part of the code takes more time and needs to be
optimized. In general, only small portions of code need to be optimized.
_
```

31. Assume the following table called "customers":rs":

CompanyName	**ContactName**	**Address**	**City**
Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.
Ernst Handel	Roland Mendel	Kirchgasse 6	Graz
FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid
Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona
Island Trading	Helen Bennett	Garden House Crowther Way	Cowes
Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg
Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver
Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo
North/South	Simon Crowther	South House 300 Queensbridge	London
Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris
Rattlesnake Canyon Grocery	Paula Wilson	2817 Milton Dr.	Albuquerque
Simons bistro	Jytte Petersen	Vinbæltet 34	København
The Big Cheese	Liz Nixon	89 Jefferson Way Suite 2	Portland
Vaffeljernet	Palle Ibsen	Smagsløget 45	Århus
Wolski Zajazd	Zbyszek Piestrzeniewicz	ul. Filtrowa 68	Warszawa

Write the result of the following queries (You can use a description when the number of rows in the resultin table is larger than 5, otherwise write down the whole resulting table).

a) SELECT *FROM customers

```
The entire table
_
```

b) SELECT ContactName FROM customersmers

```
The entire table but only has 1 ContactName column
_
```

c) SELECT CompanyName FROM customers WHERE ContactName LIKE Liz%Liz%

```
The Big Cheese
_
```

d) SELECT CompanyName, ContactName WHERE City LIKE Portlandland

```
------------------------------
| The Big Cheese | Liz Nixon |
_
```

e) Write a query to get the companies that are in Spainpain

```
SELECT CompanyName from customers WHERE City LIKE Madrid OR City LIKE
Barcelona
_
```

f) Write a query to get all the companies that start with R or W

```
SELECT CompanyName from customers WHERE CompanyName LIKE 'R%' OR
CompanyName LIKE 'W%'
_
```