CS252 Final Review Homework

Ryan Everett

1. Write a program "grepsort arg1 arg2 arg3" that implements the command "grep arg1 | sort  < arg2 >> arg3". The program should not return until the command finishes. "arg1", "arg2", and "arg3"  are passed as arguments to the program. Example of the usage is"grepsort hello infile outfile". This command will print the entries in file infile that contain the string hello and will append the output sorted to file outfile. Use pipes. Do error checking. Notice that the output is appended to arg3.

#!/bin/sh

touch $2

touch $3

grep $1 < $2 >> $3

2. 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. 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 pipefd[2];

char \*args[2] = {command, NULL};

pipe(pipefd);

if (fork() == 0) {

close(pipefd[0]);

dup2(pipefd[1], 1);

dup2(pipefd[1], 2);

close(pipefd[1]);

execvp(command, args);

}

else {

close(pipefd[1]);

while (read(pipefd[0], outputBuffer, maxBufferSize) != 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 );

}

3. 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.

#include <pthread.h>

struct List {

  int val;

  int next;

};

struct List \* head = NULL;

pthread\_mutex\_t mutex;

sema\_t full;

sema\_t empy;

main()

{

 pthread\_mutex\_init(&mutex, NULL);

sema\_init(&full, 20, USYNC\_THREAD, NULL);

sema\_init(&empty, 0, USYNC\_THREAD, NULL);

}

void insert( int val )

{

sema\_wait(&full);

pthread\_mutex\_lock(&mutex);

  List tmp = new List;

  tmp->val = val;

  tmp->next = head;

  head = tmp;

pthread\_mutex\_unlock(&mutex);

sema\_post(&empty);

}

Struct List \* removeFirst()

{

sema\_wait(&empty);

pthread\_mutex\_lock(&mutex);

  List tmp = head;

  head = tmp->next;

sema\_wait(&full);

pthread\_mutex\_unlock(&mutex);

  return tmp;

}

4. Using C++ and Semaphores write a class SynchronizedStackSemaphores of int values where pop() will block  if the stack is empty and push will block if the stack is full. Write the member variables that you think are necessary. Implement the stack with an array of int's and allocate it dynamically in the constructor. Hint: Use the "Bounded Buffer Problem" with semaphores as an example in your implementation.

#include <synch.h>

#include <pthread.h>

class SynchronizedStackSemaphores {

    // Add your member variables here

    int top;

    int \* stack;

    public:

        SynchronizedStackSemaphores(int maxStackSize);

        void push(int val);

        int pop();

};

SynchronizedStackSemaphores::SynchronizedStackSemaphores(int maxStackSize) {

        top = 0;

        stack = new int[maxStackSize];

}

void SynchronizedStackSemaphores::push(int val) {

}

int SynchronizedStackSemaphores::pop(){

}

5. 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) {

int slaveSocket = recieve(masterSocket);

dispatchHTTP(slaveSocket);

}

}

void forkServer( int masterSocket) {

struct sigaction sa1;

sa1.sa\_handler = zombie;

sigemptyset(&sa2.sa\_mask);

sa1.sa\_flags = SA\_RESTART;

if(fork() == 0) {

int slaveSocket = recieve(masterSocket);

dispatchHTTP(slaveSocket);

}

}

void poolOfThreads( int masterSocket) {

int i = 0;

pthread\_t threads[4];

for (i = 0; i < 4; i++) {

pthread\_create(&threads[i], NULL, iterativeServer, masterSocket);

}

iterativeServer(masterSocket);

}

void createThreadForEachRequest( int masterSocket ) {

int slavesocket = receive(masterSocket);

pthread\_attr\_t attributes;

pthread\_attr\_init(&attributes);

pthread\_attr\_setdetachstate(&attributes, PTHREAD\_CREATE\_DETACHED);

pthread\_create(NULL, &attributes, dispatchHTTP, slavesocket);

}

// Other procedures

void zombie(int sig) {

int pid = wait3(0,0,NULL);

while(waitpid(-1, NULL, WNOHANG) > 0) {

}

extern void dispatchHTTP(int slaveSocket);

int recieve(masterSocket) {

struct sockaddr\_in clientIPAddress;

int slen = sizeof(clientIPAddress);

return accept(masterSocket,

(struct sockaddr \*)&clientIPAddress,

(socklen\_t\*)&slen);

}

6. 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);

}

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

IP Address, Default Gateway, Subnet Mask, and Default DNS Server. The IP address is used as the place to receive messages. The Default Gateway is the address for which the client sends requests to. The Subnet Mask identifies what network the client belongs to and where valid addresses are. The Default DNS Server is where the client sends DNS requests to so a client can more easily access websites.

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

A packet can typically be delivered directly if it’s a LAN packet. All external packets, such as to Google or Facebook, must go through a router. LAN packets can be delivered directly if the two clients have a direct connection to each other (via switching also works). A router will only be involved for addresses outside of the subnet.

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

**A**ddress **R**esolution **P**rotocol. A client wants to send a message to client 2 with a given IP address. It first looks in a cached arp table, if the MAC address for client 2 exists then the message is sent to the MAC address. If the MAC address doesn’t exist in the cache, a broadcast arp is sent from the client to the entire network. Upon broadcast, client 2 will send client 1 its arp details so that the client can send its message.

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

**D**omain **N**ame **S**ystem. Takes a domain (e.g. google.com) and resolves it to an IP address that can be communicated with.

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

**D**ynamic **H**ost **C**onfiguration **P**rotocol. A device sends a request for an IP address to the router, the router returns with an offer, the client can then accept the request or deny it. If the client accepts the response, the client can used that address to communicate with.

12. What does UDP mean?

**U**ser **D**atagram **P**rotocol.

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

**T**ransmission **C**ontrol **P**rotocol. Adaptive Retransmission, Cumulative Acknowledgements, Fast Retransmission, Flow Control, Congestion Control, Reliable Connection and Shutdown.

14. When should you use TCP and when should you use UDP?

You should use TCP if the packet cannot be lost or the client is expecting a response. UDP should be used whenever the client doesn’t necessarily have to be given a response and the client expects to send many more packets.

15. 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.

**N**etwork **A**ddress **T**ranslation. A client wants to establish a TCP connection with a server X. 2. A sends a packet to the NAT box. The NAT box choses an unused port and substitutes the source port/address with the box’s ip/new port and then sends packet to X. The old port, old source address, and new port are added to the NAT table. When X responds to the NAT box, the NAT box does a lookup on the table and swaps the port/address info for A. Finally, the NAT box forwards the packet back to A.

16. 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.

NAT boxes can be used as a firewall because any unrequested address will be ignored. A client can send a request to address 1 and get a response, but if address 2 has never been requested then address 2 cannot send messages to the client. This action would normally disallow web servers to be ran because unrecognized addresses are normal behavior for a webserver. This problem is solved by forwarding all traffic on a specific port (80) to a specific local address (webserver).

17. 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.

#!/bin/sh

curl -X POST $1:$2 -d $3

18. 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.

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

Do you use source control?

Do you track bugs?

Do you have testers?

Do you have daily builds?

Do programmers have quiet working conditions?

20. What is XP programming?

Extreme programming. It is a practical methodology for software development and gives a list of rules that have been proven successful for software development.

21. 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.

Planning List: User Stories, Release Planning, Iterative Development, fix XP when it breaks.

Coding List: Have customer available, no overtime, integrate often, and write unit-test first.

Design List: Keep it simple, refactor, use spike solutions, and choose a system architecture.

Testing List: Mandatory unit tests, must pass unit tests, tests for all bugs, and acceptance testing

22. Explain 5 uses of the source control system.

Keep track of changes, backup sources, peer review changes, and check who changed what.

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

Distributed can be used for programmers that have to work a very long time on sources without having to submit to the common code. This is good for whenever a programmer might not have a consistent connection to the repo but requires a selected good repo for release. Centralized allows all programmers to contribute to the same repository and easily pull in changes. Centralized requires each client to have a copy of the repo which might be large.

24. Describe the 4 types of tests, who writes these tests in the organization, and when do they run.

Unit Tests – Programmers

System Tests – QA and Programmers

Regression Tests – Programmers and QA

Acceptance Tests – QA

25. Explain why it is important to have a bug tracking system.

A bug tracking system is important for keeping track of bugs, how to show bug, their progress on getting fixed, who is fixing the bug, the fix for the bug, and other tracking features such as priority/severity. Without a bug tracking system, a bug may be submitted but never fixed because it was forgotten. Maybe the fix applied introduced other bugs and it’s harder to figure out why.

26. Explain the difference between Priority and Severity in a bug.

Priority describes when to fix a bug; if the priority is “Urgent” then the bug must be fixed as soon as possible. Severity describes what the bug affects; if the severity is high then critical components might need to be fixed. If a bug has slow severity, then it might not be worth investigating and fixing.

27. Mention 5 cases when you can apply refactoring.

Long method, long parameter list, duplicated code, bad method name, and a large class.

28. 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?

A Software Pattern is a reusable design solution for reoccurring problems. Software Patterns were first described in the book “Design Patterns: Elements of Reusable Object-Oriented Software”. It includes pattern name, synopsis, context, forces, solution, consequences, implementation, and related patterns.

29. Describe the Proxy Pattern and 2 applications.

The proxy pattern means using a proxy instead of accessing the object natively. A proxy implements the same behaviours but can also offload logic away from the caller and allow the implementer to add additional logic. Such additional logic could be controlling read write access to a file with multiple connections. Additionally logging can be implemented for a request receiver.

30. Describe the Command Pattern and two applications.

The command pattern describes storing commands as objects so you can manipulate them further than execution. This pattern can be used to be implement undo and redo commands.

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

Code Instrumentation profiling relies on additional code being injected prior to method execution and return to measure the execution of a method. Statistical Sampling Profiling does not require any code injection and instead relies on sampling the program at regular intervals to evaluate execution. Code Instrumentation is more precise at evaluating method timing but requires more overhead. Statistical Sampling has very little overhead and can still find the most expensive function.

32. 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.

Optimization should be left to the end of the software cycle because it negatively affects readability. You should use a profiler to accurately examine what needs optimizing versus what can be more inefficient. Optimizing an operation that isn’t executed often uses time for little benefit.

33. Assume the following table called "customers":

|  |  |  |  |
| --- | --- | --- | --- |
| 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 resulting table is larger than 5, otherwise write down the whole resulting table).

a) SELECT \*FROM customers

Returns all the rows and all the columns in the table *customers*.

b) SELECT ContactName FROM customers

Returns all the rows but with only the column ContactName.

c) SELECT CompanyName FROM customers WHERE ContactName LIKE Liz%

|  |
| --- |
| The Big Cheese |

d) SELECT CompanyName, ContactName FROM customers WHERE City LIKE Portland

|  |  |  |  |
| --- | --- | --- | --- |
| The Big Cheese | Liz Nixon |  |  |

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

SELECT \* FROM customers WHERE City=’Madrid’ || City=’Barcelona’;

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

SELECT \* FROM customers WHERE CompanyName LIKE ’R%’ || CompanyName LIKE ‘W%’;