CS252 Midterm Homework

Jeffrey Hughes - 0026991582

### **Part 1. True False Questions**

\_\_T\_\_ The loader is also called “Runtime Linker”

\_\_T\_\_ COFF is a format for executable files

\_\_F\_\_ The command "chmod 440 file" makes a file readable and writable by user, group, and others.

\_\_F\_\_ strace is a UNIX command that shows the tree of processes in the system.

\_\_F\_\_ All processes have a parent process. (PID 0, init)

### **Part 2. Short questions.**

*2. Enumerate and describe the memory sections of a program.*

Text- Instructions that the program runs

Data – Initialized global variables.

Bss – Uninitialized global variables. They are initialized to zeroes.

Heap – Memory returned when calling malloc/new. It grows upwards.

Stack – It stores local variables and return addresses. It grows downwards.

Dynamic libraries – They are libraries shared with other processes.

*3. Enumerate and describe the contents of an inode*

Major number – Identifier number that tells which device a file is on

Minor number – Identifier number that tells which actual file it is on the device

Permission modes – The permissions for owner, group, and others

Owners – ID of users and groups that own the file

Size – Size of file in bytes

Timestamps – Creation time, access time, modification time

Reference count – Number of times this file is referenced as hard link

*4. Enumerate the 5 Memory Allocation Errors and describe them.*

Memory Leaks - Memory leaks are objects in memory that are no longer in use by the program but that are not freed.

Premature Free - A premature free is caused when an object that is still in use by the program is freed.

Double Free - Double free is caused by freeing an object that is already free.

Wild Frees - Wild frees happen when a program attempts to free a pointer in memory that was not returned by malloc.

Memory Smashing - Memory Smashing happens when less memory is allocated than the memory that will be used.

*5. List and explain the attributes of an Open File Object.*

I-Node - It uniquely identifies a file in the computer. An I-nodes is made of two parts:

Major number – Determines the devices

Minor number –It determines what file it refers to inside the device.

Open Mode -

How the file was opened:

Read Only

Read Write

Append

Offset - The next read or write operation will start at this offset in the file.

Reference count - It is increased by the number of file descriptors that point to this Open File Object.

### **Part 3. Programming questions.**

|  |
| --- |
| 6. From your lab 1 Memory Allocator write the functions int getSize(char \* ptr), int canCoalesce(char \*ptr), void coalesceWithRight(char \* p) The descriptions of these functions are in the procedures below. |

// Header of an object. Used both when the object is allocated and freed

struct ObjectHeader {

size\_t \_objectSize; // Real size of the object.

int \_allocated; // 1 = yes, 0 = no 2 = sentinel

struct ObjectHeader \* \_next; // Points to the next object in the freelist (if free).

struct ObjectHeader \* \_prev; // Points to the previous object.

};

struct ObjectFooter {

size\_t \_objectSize;

int \_allocated;

};

// Returns the size of the object p passed in the parameter.

// p is a pointer returned by malloc().

int getSize(void \* p) {

struct ObjectHeader \* o = (struct ObjectHeader \*) ( (char \*) p - sizeof(struct ObjectHeader) );

// Substract the size of the header

return o->\_objectSize;

}

// Takes a pointer p returned by malloc and canCoalesce returns:

// 0 - The object p cannot be coalesced

// 1 - The object p can be coalesced only with the left object

// 2 - The object p can be coalesced only with the right object

// 3 - The object p can be coalesce with both left and right.

int canCoalesce(char \*ptr) {

struct ObjectHeader\* header = (struct ObjectHeader\*) ((char\*) ptr - sizeof(struct ObjectHeader));

struct ObjectFooter\* footer = (struct ObjectFooter\*) ((char\*) header + (header->\_objectSize - sizeof(struct ObjectFooter)));

if (header->\_prev->\_allocated == 0 && header->\_next->\_allocated == 0) {

return 3;

} else if (header->\_prev->\_allocated == 0) {

return 1;

} else if (header->\_next->\_allocated == 0) {

return 2;

} else {

return 0;

}

}

// Coalesce the object pointed by p with the object in the right (higher in memory).

// The object in the right is assumed to be free.

// p is a pointer returned by malloc().

void coalesceWithRight(char \* p) {

struct ObjectHeader\* header = (struct ObjectHeader\*) ((char\*) ptr - sizeof(struct ObjectHeader));

header->\_allocated = 0;

struct ObjectFooter\* footer = (struct ObjectFooter\*) ((char\*) header + (header->\_objectSize - sizeof(struct ObjectFooter)));

footer->\_allocated = 0;

struct ObjectHeader\* freeList = \_freeList->\_next;

struct ObjectHeader\* bottom = freeList->\_prev;

struct ObjectHeader\* top = freeList;

struct ObjectFooter\* bottomFooter = (struct ObjectFooter\*) ((char\*) header - sizeof(struct ObjectFooter));

struct ObjectHeader\* topHeader = (struct ObjectHeader\*) ((char\*) header + header->\_objectSize);

bottom->\_objectSize += header->\_objectSize;

bottom->\_allocated = 0;

struct ObjectFooter\* newFooter = (struct ObjectFooter\*) ((char\*) bottom + (bottom->\_objectSize - sizeof(struct ObjectFooter)));

newFooter->\_objectSize = bottom->\_objectSize;

newFooter->\_allocated = 0;

return;

}

|  |
| --- |
| 7. Write a shell script "monitorprocess pid maxmem *emailaddr*" that will send e-mail to*emailaddr*  when the process with pid exceeds maxmem or when with pid exits. |

function calculate\_mem\_usage

{

#Let us extract the VmRSS value from /proc/{pid}/status

mem\_usage=$(egrep VmRSS /proc/$PID/status | awk '{print $2}')

#Return the memory usage

echo "$mem\_usage"

}

function notify

{

mem\_usage\_int=$(printf "%.f" $2)

if [[ "$mem\_usage\_int" -gt "$MEM\_THRESHOLD" ]]; then

echo "Memory Usage Exceeded - Sending Email" >&2

cd reports\_dir

message=$(ls -t | tail -n 1 | xargs cat)

echo $message | mailx -s "Memory Usage Exceeded" $EMAIL

cd ..

fi

}

#The monitor runs forever

while [ -n "$(ls /proc/$PID)" ] #While this process is alive

do

PID=$1

MEM\_THRESHOLD=$2

EMAIL=$3

mem\_usage=$(calculate\_mem\_usage)

#Call the notify function to send an email to $3 if the thresholds were exceeded

notify $mem\_usage

done

|  |
| --- |
| 8. From your shell project write the simplified function "execute" that will execute the command passed as argument. Each command is made of multiple SimpleCommands that communicate with pipes. simpleCommand[0] will take the input from file "input" and it will pass its output to simpleCommand[1] and so on. The output of the last SimpleCommand will be passed to the file in "output" if any. Then it will wait until the last command finishes. If "input" or "output" are NULL then use the default input/output. |

typedef struct SimpleCommand {

const char \* arguments[]; // Command and arguments of this SimpleCommand. Last argument is NULL.

};

typedef struct Command {

int numberOfSimpleCommands; // Number of simple commands

SimpleCommand simpleCommand[]; // Array of simpleCommands

const char \* input; // Input file or NULL if default input

const char \* output; // Output file or NULL if default output

int background;

};

void execute( Command \* command) {

for ( int i = 0; i < command->numberOfSimpleCommand; i++) {

dup2(fdin, 0);

close(fdin); //setup output

if (i == numsimplecommands-1) { // Last simple command

if(outfile){

fdout=open(outfile,……);

} else {

// Use default output

fdout=dup(tmpout);

}

} else { // Not last

//simple command

//create pipe

int fdpipe[2];

pipe(fdpipe);

fdout=fdpipe[1];

fdin=fdpipe[0];

}// if/else

// Redirect output

dup2(fdout,1);

close(fdout);

// Create child process

ret=fork();

if (ret==0) {

execvp(scmd[i].args[0],

scmd[i].args);

perror(“execvp”);

exit(1);

}

}

}

|  |
| --- |
| 9. From your shell project write the function that does the wildcard expansion. Assume that the function wildcardToregularExpression is given. |

void expandWildcardIfNecessary(char\* arg) {

// Return if arg does not contain ‘\*’ or ‘?’

if (strchr(arg, '\*') != NULL || strchr(arg, '?') != NULL) {

Command::\_currentSimpleCommand->insertArgument(arg);

return;

}

char \* reg = (char\*)malloc(2\*strlen(arg)+10);

char \* a = arg;

char \* r = reg;

\*r = ‘^’; r++;

// match beginning of line

while (\*a) {

if (\*a == ‘\*’) {

\*r=‘.’; r++; \*r=‘\*’; r++;

} else if (\*a == ‘?’) {

\*r=‘.’ r++;

} else if (\*a == ‘.’) {

\*r=‘\\’;

r++;

\*r=‘.’;

r++;

} else {

\*r=\*a;

r++;

}

a++;

} \*r=‘$’;

r++;

\*r=0;// match end of line and add null char

// 3. List directory and add as arguments the entries

// that match the regular expression

DIR \* dir = opendir(“.”);

if (dir == NULL) {

perror(“opendir”);

return;

}

struct dirent \* ent;

while ( (ent = readdir(dir))!= NULL) {

// Check if name matches

if (regexec(ent->d\_name, expbuf ) ==0 ) {

// Add argument

Command::\_currentSimpleCommand->insertArgument(strdup(ent->d\_name));

}

}

closedir(dir);

}

|  |
| --- |
| 10. (20 pts.) 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**. Do error checking. Notice that the output is appended to arg3. |

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

const char \*usage = "Usage:\tgrepsort arg1 arg2 arg3\n\tgrep arg1 | sort < arg2 >> arg3\n";

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

int defaultin = dup(STDIN\_FILENO);

int defaultout = dup(STDOUT\_FILENO);

int pipefd[2];

pipe(pipefd);

int infd = open(argv[2], O\_RDONLY);

dup2(infd, STDIN\_FILENO);

close(infd);

dup2(pipefd[1], STDOUT\_FILENO);

close(pipefd[1]);

int ret = fork();

//child

if (ret == 0) {

close(pipefd[0]);

char \*args[3];

args[0] = "grep";

args[1] = argv[1];

args[2] = NULL;

execvp(args[0], args);

perror("execvp");

\_exit(1);

}

int outfd = open(argv[3], O\_WRONLY | O\_APPEND | O\_CREAT, 0666);

dup2(outfd, STDOUT\_FILENO);

close(outfd);

dup2(pipefd[0], STDIN\_FILENO);

close(pipefd[0]);

ret = fork();

if (ret == 0) {

close(pipefd[1]);

char \*args[2];

args[0] = "sort";

args[1] = NULL;

execvp(args[0], args);

perror("execvp");

\_exit(1);

}

dup2(defaultin, STDIN\_FILENO);

dup2(defaultout, STDOUT\_FILENO);

close(defaultin);

close(defaultout);

waitpid(ret, 0, 0);

return 0;

}