Operating systems tasks and programming lab

A

Lab Activity 1 – Operating Systems Tasks and Programming

a) Future of operating systems.

[Report on the future of operating systems here]

b) Programming activity

C++

#include usingnamespacestd; intmain() {A, Aintid = 6669447; string name = “ Salman Mohammed Fazal”; intcounter = 0; for(inti = 0; i < name. length(); i++) { //for every character cout << name[i] << flush; //print characters on the same line counter++; //increment counterif(counter == id[-1]) { //if counter = last character of IDcout << endl; //jump to next line counter = 0; //reset counter } } }

JAVA

publicclasstest {publicstaticvoidmain(String[] args) {intid = 6669447; String name = “ Salman Mohammed Fazal”; intcounter = 0; for(inti = 0; i < name. length(); i++) { //for every character System. out. print(name. charAt(i)); //print character counter++; //increment counterif(counter == id[-1]) { //when counter is last character of IDSystem. out. println(); //new linecounter = 0; //reset counter}}}}

PYTHON

name = “ Salman Mohammed Fazal” id = “ 6669447” foriinrange (0, len(name), int(id[-1])): #for every n charactersprint(name[i: i+ int(id[-1]))]) #slice and print string

Code Outcome (same for all)

A

Lab Activity 2 – Linux Command Line (Commands and outcomes from a series of small tasks that require use of a number of Linux commands)

How made Portfolio1 directory read/write/executable only for you and your group. That is, not for others. A, A Show evidence of this with ls command.

How downloaded the script http://www. centerkey. com/tree/tree. sh to your home directory using wget and make it executable.

Making DirectoriesHow created a 207se directory in your Portfolio1 directory. How created numbered directories for the labs. i. e. lab1 and lab2 etc. Evidence of transferring lab1 activity into appropriate directoryEvidence of make directory activities using tree. sh

Display todays date and using the cal command show the month that you were born.

Move into the lab1 directory and use the appropriate command to show the current directory

What is talk, write and wall are for

TheTalkcommand is used for exchanging messages with other users who are logged on the same network.

TheWritecommand is used to send messages to users in the same network. This one-way only.

TheWallcommand is used to send messages to all the users in the entire network. This is limited to the admins only.

A, A What command prevents the effects of those three commands from interrupting you.

To prevent users from interrupting you, you can use theMesgcommand. This command basically enables or disables users from writing to your terminal.‘ Mesg y’to Allow access and‘ Mesg n’to deny access.

The song in song. txt. Using wc the number of words and lines in the file. Using grep to get the lines containing “ and” and the number of the lines contain “ and” in the documentUse cat to show the contents of the file. Appropriate Linux command to see if the two files differ and how they differ. Use sort to sort the file and redirect the output to a new file called song2. txtUse sort and rev to reverse the sorted contents of song. txt and append the output to song2. txtTotal memory used and the total memory availableFind out how you can display your username on the screen. List the processes that are running. What are the differences between the Linux commands less, more and most.

The‘ more’command works similarly to the cat command by displaying contents of a file, however the more command, you view bits of the text (a *screenful*), and is in a forward-scrollable manner.

The‘ less’command is similar to themorecommand, however with this it is possible to scroll both, forward and backward.

The‘ most’command is more like thelesscommand, but the only difference is, it is possible to view several files at once with this command.

The basic syntax for these commands are:

more less

A

Lab Activity 4 Bootloader

Brief description of the Lab activity and what you did

This weeks task was to create a bootloader using Assembly which included my student details and a triangle of dots. We then have to boot the bootloader with bochs.

Boot pragma linux with bochs

Make a bootloader that displays your student details and triangleCommented bootloader code to display your student details and triangle

[BITS 16]

[ORG 0x7C00]

```
top:

;; Put 0 into ds (data segment)

;; Can’t do it directly

mov ax, 0x0000

mov ds, ax

;; si is the location relative to the data segment of the

;; string/char to display

mov si, msg

call writeString ; See below

jmp $ ; Spin

writeString:

mov ah, 0x0E ; Display a chacter (as before)

mov bh, 0x00

mov bl, 0x07

nextchar:

Lodsb ; Loads [SI] into AL and increases SI by one

;; Effectively “ pumps” the string through AL
```

```
cmp al, 0 ; End of the string?

jz done

int 0x10 ; BIOS interrupt

jmp nextchar

done:

ret

msg db ‘ Name: Salman Fazal’, 13, 10, ‘ Email: ‘, 13, 10, ‘ Fav Module: 207SE ;)’, 13, 10, ‘ DOB: 01/08/1996 – 21’ , 13, 10, ‘ Std ID: 6669447’ ; Null-terminated

times 510-($-$$) db 0

dw 0xAA55
```

Output from Bochs showing student details and triangle

A

Lab Activity 6 Memory Management

Memory Allocation Activities

Due to the code outputs being too long, I have just snipped a portion of the result, however all of my answers match with the results in the code.

* NULL = Memory address not allocated.

First-Fit

A

Best-Fit

A

Worst-Fit

A

Paging Activities

FIFO

4

2

7

7

5

6

3

9

3

2

2

Page Entry 0

4

4

4

4

5

5

5

9

9

9

9

Page Entry 1

2

2

2

2

6

6

6

6

2

2

Page Entry 2

7

7

7

7

3

3

3

3

3

Page Fault

F

F

F

H

F

F

F

F

H

F

H

Page Fault Total: 8

4

2

7

7

5

6

3

9

3

2

2

Page Entry 0

4

4

4

4

4

6

6

6

6

6

6

Page Entry 1

2

2

2

2

2

3

3

3

3

3

Page Entry 2

7

7

7

7

7

9

9

9

9

Page Entry 3

5

5

5

5

5

2

2

Page Fault

F

F

F

H

F

F

F

F

H

F

H

Page Fault Total: 8

RANDOM

4

2

7

7

5

6

3

9

3

2

2

Page Entry 0

4

4

4

4

5

6

6

6

6

2

2

Page Entry 1

2

2

2

2

2

2

9

9

9

9

Page Entry 2

7

7

7

7

3

3

3

3

3

Page Fault

F

F

F

H

F

F

F

F

H

F

H

Page Faults Total: 8

4

2

7

7

5

6

3

9

3

2

2

Page Entry 0

4

4

4

4

4

4

4

9

9

9

9

Page Entry 1

2

2

2

2

2

2

2

2

2

2

Page Entry 2

7

7

7

7

3

3

3

3

3

Page Entry 3

5

6

6

6

6

6

6

Page Fault

F

F

F

H

F

F

F

F

H

H

H

Page Fault Total: 7

The random algorithm did not give me the same result as the way I solved it. This is because There is no specific rule/method on what memory block to be

replaced. Each time a process needs to be moved to a memory block, a random memory block is chosen in order to swap the process.

Evidence of running code

A

Lab Activity 7 Buffer

Brief description of the Buffer Activity

This weeks task involved using buffers in terms of reading and writing from a file

Commented Buffer. c code#include //library for file control options#include //library for general purpose tools#include //header file#include //file IO#define BUF_SIZE 500 //sets buffer size to 500#define OUTPUT_MODE 0700 //defines the output mode, sets file permissionsintmain(intargc, char*argv[]) {//Define variablesintin_fd, out_fd; //hold associated numbers on both filesintrd_size = 1; //hold amount of bytes in buffer (final should be 500)intwr_size; //hold amount of bytes on the output filecharbuf[BUF_SIZE]; //initialise buffer and its sizeif(argc != 3) //check for correct number of argumentsexit(1); //exit if too many or too few parametersin_fd = open(argv[1], O_RDONLY); //open file to read from (read only)//if file isn’t found, variable will have a negative numberif(in_fd < 0) exit(2); //exit if file’s empty out_fd = creat(argv[2], OUTPUT_MODE); //create the output file //if file isn’t created, variable will have a negative numberif(out_fd < 0) exit(3); //exit if cannot write to the filewhile(rd_size > 0) { //while characters (in file) still exist: rd_size = read(in_fd, buf, BUF_SIZE);

//read the file into bufferif(rd_size < 0) exit(4); //exit if error while reading wr_size = write(out_fd, buf, rd_size); //write from buffer into fileif(wr_size <= 0) { close(in_fd); close(out_fd); //close both of the files if error found exit(5); //exit if error while writing } } } Update the code to so that it prints if an error has occurred or if a file is successfully created with the content of the review in it.

After running code what is in hamlet. txt

#include //library for file control options#include //library for general purpose tools#include //header file#include //file IO#define BUF_SIZE 500 //sets buffer size to 500#define OUTPUT_MODE 0700 //defines the output mode, sets file permissionsintmain(intargc, char*argv[]) {//Define variablesintin_fd, out_fd; //hold associated numbers on both filesintrd_size = 1; //hold amount of bytes in buffer (final should be 500)intwr_size; //hold amount of bytes on the output filecharbuf[BUF_SIZE]; //initialise buffer and its sizeif(argc != 3) { //check for correct number of argumentsprintf(“ Error: Invalid number of parameters passed.”); exit(1); //exit if too many or too few parameters}in_fd = open(argv[1], O_RDONLY); //open file to read from (read only)//if file isn’t found, variable will have a negative numberif(in_fd < 0) { printf(“ Error: File not found.”); exit(2); //exit if file’s empty } out_fd = creat(argv[2], OUTPUT_MODE); //create the output file //if file isn’t created, variable will have a negative numberif(out_fd < 0) { printf(“ Error: File not created.”); exit(3); //exit if cannot write to the file }while(rd_size > 0) { //while characters (in file) still exist: rd_size = read(in_fd, buf, BUF_SIZE); //read the file into bufferif(rd_size < 0) { printf(“ Error: Can’t read from file.”); exit(4); //exit if error while reading } wr_size = write(out_fd, buf, rd_size);

//write from buffer into fileif(wr_size <= 0) { close(in_fd); close(out_fd); //close both of the files if error found exit(5); //exit if error while writing } } printf(“ Operation Successful!”); }

After compiling and running the code, the hamlet. txt file contained the exact same text that was in the main review. txt file

Evidence:

Updated buffer. c code to show how many character are read to buffer, how many character read at a time into the buffer, how many words in the document and how many times the buffer is filled#include //library for file control options#include //library for general purpose tools#include //header file#include //file IO#define BUF_SIZE 500 //sets buffer size to 500#define OUTPUT_MODE 0700 //defines the output mode, sets file permissionsintmain(intargc, char*argv[]) {//Define variablesintin_fd, out_fd; //hold associated numbers on both filesintrd_size = 1; //hold amount of bytes in buffer (final should be 500)intwr_size; //hold amount of bytes on the output filecharbuf[BUF_SIZE]; //initialise buffer and its sizeint rd_count = 0, buf_count = 0, wd_count = 0; if(argc != 3) { //check for correct number of argumentsprintf(“ Error: Invalid number of parameters passed.”); exit(1); //exit if too many or too few parameters}in_fd = open(argv[1], O_RDONLY); //open file to read from (read only)//if file isn’t found, variable will have a negative numberif(in_fd < 0) { printf(“ Error: File not found.”); exit(2); //exit if file’s empty } out_fd = creat(argv[2], OUTPUT_MODE); //create the output file //if file isn’t created, variable will have a negative numberif(out_fd < 0) { printf(“ Error: File not created.”); exit(3); //exit if

cannot write to the file }while(rd_size > 0) { //while characters (in file) still exist: rd_size = read(in_fd, buf, BUF_SIZE); //read the file into bufferif(rd_size < 0) { printf(“ Error: Can’t read from file.”); exit(4); //exit if error while reading } if (rd_size == 500){ //if read, rd_count += 500; buf_count += 1; //increment character and buffer counter } else{ rd_count += rd_size; } for (int i= 0; i if (wr_size <= 0) { printf(“ Successfully written to file! n”); close(in_fd); close(out_fd); //close both files printf(“ Total number of characters read is: %d. n”, rd_count); printf(“ Total number of words: %d. n”, wd_count); printf(“ Buffer filled %d times. n”, buf_count); exit(5); } } }

Impact of changing buffer size

I altered the buffer 3 times, the first was changing the size to 1000, the second was 1300 and the third was 2000. What I noticed was as the buffer size increases, the amount of times the buffer is filled decreases as the buffer is able to fill in more characters each time. Below is a screen shot of when the buffer size was set to 2000:

We can see the buffer at this time was filled 0 times, meaning the entire text was placed into the buffer.

Updated buffer. c code to compare if two files are the same#include //library for file control options#include //library for general purpose tools#include //header file#include //file IO#define BUF_SIZE 500 //sets buffer size to 500#define OUTPUT_MODE 0700 //defines the output mode, sets file permissionsintmain(intargc, char*argv[]) {//Define variablesintin_fd, in_fd2; //hold associated numbers on both filesintrd_size = 1; //hold amount of bytes in buffer (final should be 500)intrd_size2 = 1; charbuf[BUF_SIZE];

//initialise buffer and its sizecharbuf2[BUF_SIZE]; //initialise buffer and its size//int rd_count = 0, buf_count = 0, wd_count = 0; if(argc != 3) { //check for correct number of argumentsprintf(“ Error: Invalid number of parameters passed.”); exit(1); //exit if too many or too few parameters}in_fd = open(argv[1], O_RDONLY); //open first file to read fromif(in_fd < 0) { printf(“ Error: File not found.”); exit(2); //exit if file’s empty } in_fd2 = open(argv[2], O_RDONLY); //open second file to read from //if file isn’t found, variable will have a negative numberif(in_fd2 < 0) { printf(“ Error: File not found.”); exit(3); //exit if file’s empty }while(rd_size > 0 && rd_size2 > 0) { //while characters (in file) still exist: rd_size = read(in_fd, buf, BUF_SIZE); //read file 1 to bufferrd_size2 = read(in_fd2, buf2, BUF_SIZE); //read file 2 to bufferif(rd_size < 0 || rd_size2 < 0) { //error if any file is empty printf(“ Error: Can’t read from file.”); exit(4); }for(inti= 0; i if(buf[i] != buf2[i]){ //if characters of the 2 files are not the same: printf(“ The files are not the same!”); close(in_fd); close(in_fd2); exit(5); //print message and close}}}printf(“ YESSSSSS!! The files are the same.”); //if code passes the loop, they’re the same!}Comparison of review. txt and hamlet. txtComparison of hamlet. txt and review_observer. txt

A

Lab Activity 8 Cache Buffer

Brief Description of Cache Buffer Activity

This weeks task was to alter the cache_reader. c file and complete the cr_read_byte function. Additionally, we also had to add a count of the total number of bytes and the number of times the buffer was filled.

Commented implementation of the cr_read_byte

functioncharcr_read_byte(cr_file* f){/* 1. check if buffer needs refilling2. return currently pointed character3. move pointer to next character */if(f-> usedbuffer >= f-> bufferlength){refill(f);}charcurrentChar = f-> buffer[f-> usedbuffer]; f-> usedbuffer++; returncurrentChar;}

Comment updated code to show that each byte is being read, and when the buffer is being refilled.

To show that the buffer is being refilled, I added a print statement each time the buffer refills in the cr_read_byte() function. Also the next part of the question will show that the buffer is being refilled and the bytes are being read by keeping a count each time that happens.

*The code for this part of the question is combined with the next part (cache_reader. c).

(I set the buffer to 200)

Commented updated code showing to show how many bytes were read in total, and how many times the buffer was refilled

In this part, I had to update the code in 3 different files, below are the screenshots of every update I made:

cache_reader. h

In this section, I initialized two variables, one to keep count of the number of times the buffer was refilled and the second to keep count of the total number of bytes.

cache_reader. c

For the second section, I incremented the buffer (filled) count each time the buffer was refilled.

cache_example. c

Lastly, each time a byte is printed to the screen, a byte count is incremented. And at the end, before closing the files, I printed both the counts.

Cache_reader. h

#include #include //The internals of this struct aren’t important//from the user’s point of viewtypedefstruct{FILE* file; //File being readintbufferlength; //Fixed buffer lengthintusedbuffer; //Current point in the bufferchar* buffer; //A pointer to a piece of memory// same length as “ bufferlength” intbuffer_count; //buffer filled countintbyte_count; //number of bytes count} cr_file;//Open a file with a given size of buffer to cache withcr_file* cr_open(char* filename, intbuffersize);//Close an open filevoidcr_close(cr_file* f);//Read a byte. Will return EOF if empty. charcr_read_byte(cr_file* f);//————————————————————————//Refill an empty buffer. Not intended for usersintrefill(cr_file* buff);

cache_reader. c

#include “ cache_reader. h”//http://www. phim. unibe. ch/comp_doc/c_manual/C/SYNTAX/struct. html//http://vergil. chemistry. gatech. edu/resources/programming/c-tutorial/structs. htmlintrefill(cr_file*

buff) {//Refills a buffer//Only works when completely used bufferif(buff-> usedbuffer != buff-> bufferlength)return0; else{buff-> usedbuffer = 0; intlen = fread(buff-> buffer, sizeof(char), buff-> bufferlength, buff-> file);//If we didn’t fill the buffer, fill up with EOFif(len < buff-> bufferlength)for(inti = len; i < buff-> bufferlength; i++)buff-> buffer[i] = EOF; //Accessing like an array! returnlen;}}voidcr_close(cr_file* f) {free(f-> buffer); fclose(f-> file);}cr_file* cr_open(char* filename, intbuffersize) {FILE* f; if((f = fopen(filename, “ r”)) == NULL) {fprintf(stderr, “ Cannot open %sn”, filename); return0;}cr_file* a = (cr_file*) malloc(sizeof(cr_file)); a-> file = f; a-> bufferlength = buffersize; a-> usedbuffer = buffersize; //Start off with no characters, so refill will work as expecteda-> buffer = (char*) malloc(sizeof(char)*buffersize); refill(a); returna;}//————————————————————————

charcr_read_byte(cr_file* f) {/* 1. check if buffer needs refilling-if refilled, increase buffer count. 2. return currently pointed character3. move pointer to next character */if(f-> usedbuffer >= f-> bufferlength) {refill(f); f-> buffer_count++; printf(“ nnBuffer Refilled.. nn”);}charcurrentChar = f-> buffer[f-> usedbuffer]; f-> usedbuffer++; returncurrentChar;}

Cache_example. c

#include “ cache_reader. h”#import “ cache_reader. h”//Simple file display to show how easy it is to use the cached reader functionsintmain() {charc;//Open a filecr_file* f = cr_open(“ text”, 200);//While there are useful bytes coming from itwhile((c = cr_read_byte(f)) != EOF) {//Print themf-> byte_count++; //increase count for every charprintf(“%c”, c);}//Then close the fileprintf(“ nnTotal number of bytes: %d”, f-> byte_count); printf(“ Buffer filled %d times.”, f-> buffer_count); cr_close(f);//And finishreturn0;}

A

Lab 10: The Cache Buffer from week 8 with system calls

Brief description of the activity

This weeks task involved changing the library to use system calls (open(), close(), read()) instead of fopen, fread and fclose.

Changes the cache_reader library from using the fopen, fread, fclose functions to the system call versions open, read, close

The changes I made to the code are:

Firstly, I added the includes to all the codes.

cache_reader. c

Changed the commented code to the second one

A

Cache_reader. h

RUNNING CODE

Changes cache_reader library to remove (as far as possible) the effects of caching on the library.

[Commented code outlining your changes to the . h and . c files here]

[Output from running code here and if possible prove not using cache here]

References