Homework Midterm Exam Midterm 2018

Midterm Preparation Homework

(Based on Midterm Exam CS252 Fall 2017)

Answer and submit your homework before your midterm exam.

1. Answer True/False (T/F) (1 pt. each)

T data is a memory section

T elf is a type of an executable file format

F The command "chmod 110 file" makes a file executable by others.

F printf() is a system call.

F The interrupt vector can be modified in user mode .

2. (5 pts.) Using the following program fill up the table with the corresponding memory sections where the indicated addresses are located

int a = 5; int b[20]; char \*q;

int main() {

int x;

int \*p =(int\*)

malloc(sizeof(int));

q = “Hello”;

}

|  |  |
| --- | --- |
| Address | Section |
| &a | Data |
| &b[0] | Data |
| &x | Stack |
| p | Stack |
| q | BSS |
| &main | Text |
| &p | Stack |
| b | Data |
| &p[0] | Stack |
| &q | BSS |

3. (5 pts.) Enumerate the 5 memory allocation error and give a code example of each of them.

a) Memory Leaks

while () {

ptr = malloc(1000);

}

b) Premature Free

int \* p = (int\*) malloc(sizeof(int));

\*p = 5;

free(p);

\*p = 6;

c) Double frees

int \* p = (int\*) malloc(sizeof(int));

free(p);

free(p);

d) Wild free

int q;

int \*p = &q;

free(p);

e) Memory smashing

char \* s = malloc(5);

strcpy(s, “Hello World!”);

4. (5 pts.) Describe all the steps that a program and the OS will do to execute the following system call int fd = write(int fd, char \* buffer, int length)

* The write function is called by the program
* The wrapper in libc makes a software interrupt handler for system call
* The OS checks the interrupt handler arguments
* The OS allows write to fd
* The OS switches to other processes after current process is put into wait state until the disk operation finishes
* An interrupt is generated after the disk finishes
* The current process is put into ready state and the process is then scheduled by the OS

5. (5 pts)

a) Explain why a bug such as the following one is difficult to debug:

int \* array = new int[40];

array[7]=9;

….

delete [] array;

array[2]=5;

The code given is the equivalent of a premature free. This is particularly difficult to debug because

an error is not thrown and if only looking at a piece of the code it will look like everything is set right.

b) What can you as a programmer do to make it easy to detect this kind of problems.

Set the array to NULL after calling delete and the compiler will then throw an error.

6. (5 pts) Explain how Multilevel Feedback-Queue Scheduling works.

Multilevel Feedback-Queue Scheduling has multiple queues of different priorities. The scheduler

schedules ready processes in highest priority order. In processes of the same priority round robin

is used.

7. (5 pts) Describe what each of the following commands and arguments do:

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ls -al** | ls lists files and directories and -al lists all entries including names beginning with a period in one column. |
| **cp -R dir1 dir2** | cp copies files and directories and -R does it recursively. Therefore, files and directories and copied recursively from dir1 to dir2. |
| **rm -rf dir1** | rm removes files or directories. -rf removes recursively and ignores nonexistent files. Therefore, the files or directories of dir1 are removed recursively and ignore nonexistent files. |
| **mkdir -p dir1/dir2** | mkdir makes directories and -p makes parent directories if needed. Therefore, dir1/dir2 directory is made and makes parent directories if needed. |
| **tail -f a.log** | tail outputs the last part of files and -f outputs appended data as the file grows. Therefore, the last part of the file a.log is then appended as the file grows. |
| **which gcc** | which returns the location of the command. Therefore, the file location of gcc is returned. |
| **find .** | find searches for files in a directory hierarchy. Therefore, all files are searched for in a directory hierarchy. |
| **ps -e** | ps reports a snapshot of the current processes and -e select all processes. Therefore, all selected processes are returned as a snapshot. |

Malloc

Below is a diagram showing current state of a memory allocator like the one implemented in lab 1. The only difference is that the arena size is only 128 bytes, to simplify the arithmetic. All of the data structures are the same as they were in the lab and the code is being run on a 64-bit linux system, like the lab machines or data. The top diagram shows how the blocks are laid out in memory, the lower diagram is a representation of the free list, and the table contains the metadata about each block. Addresses are truncated and given in decimal for simplicity.

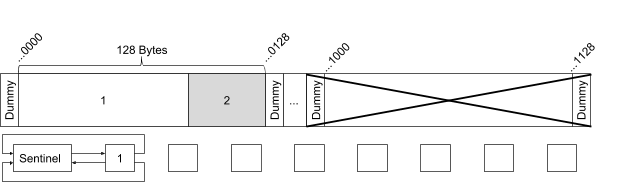
In the diagram given below there are two blocks in a single arena. Block 1 is not allocated and is the only node in the free list. Block 2 has been allocated. There is space for a second arena to be allocated if necessary. If the second arena is not required simply cross it out as in the diagram below. For each malloc/free call update the memory space diagram, free list, and table as necessary to contain the state of the allocator after performing the requested operation.

Note: However you choose to number the blocks doesn’t matter - just make sure that your numbers are consistent between all three diagrams. Additionally, you can denote “shading” in the diagram however you wish

- just make sure it’s clear it’s different.

Example Diagram:

Block Offset Size Left Size Allocated



1 ...0000 88 24 0

2 ...0088 40 88 1

...

...

Question 8

Block Offset Size Left Size Allocated

1 ...0000 88 24 0

2 ...0088 40 88 1

...

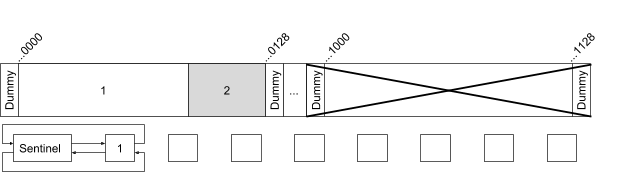
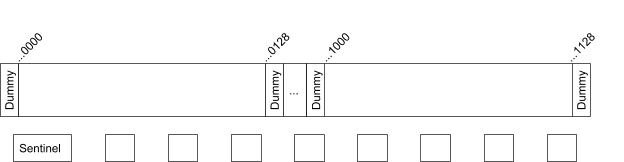
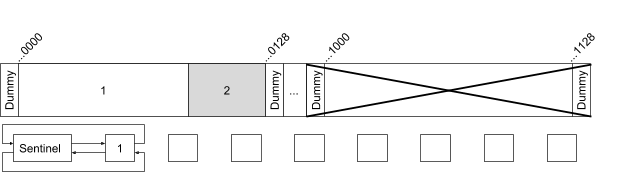
...

malloc(60);

Block Offset Size Left Size Allocated

...

...



...

...

What is the value returned by the operation above, assuming the beginning of the allocable memory in the first arena is address 10000?

Question 9

Block Offset Size Left Size Allocated

1 ...0000 88 24 0

2 ...0088 40 88 1

...

...

malloc(112);

Block Offset Size Left Size Allocated

...

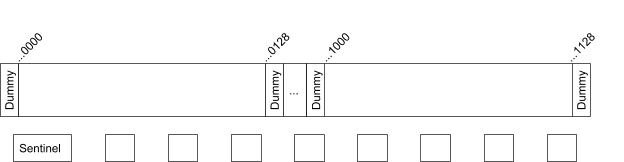
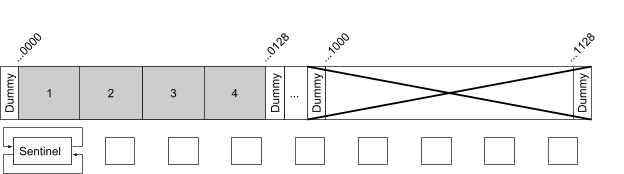
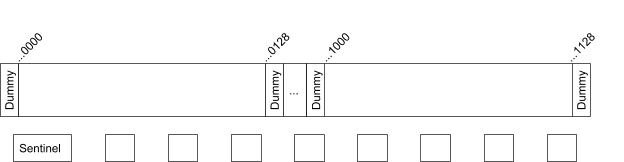
...

...

...

What is the value returned by the operation above, assuming the beginning of the allocable memory in the first arena is address 10000?

Question 10



Block Offset Size Left Size Allocated

1 ...0000 40 24 1

2 ...0040 40 40 1

3 ...0080 40 40 1

4 ...0120 40 40 1

free(...0000 + sizeof(BoundaryTag)) // free block 1

Block Offset Size Left Size Allocated

...

...

...

Question 11

...

Block Offset Size Left Size Allocated

1 ...0000 40 24 0

2 ...0040 40 40 1

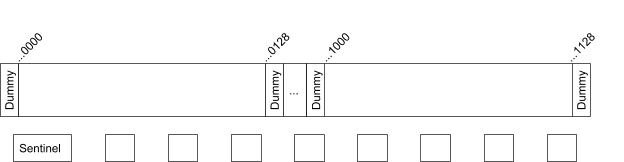
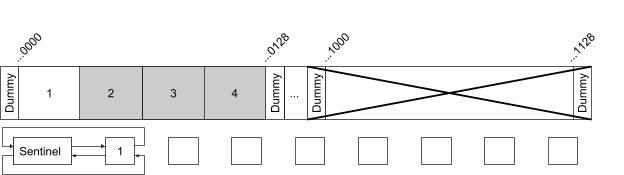
3 ...0080 40 40 1

4 ...0120 40 40 1

free(...0032 + sizeof(BoundaryTag)) // free block 2

Block Offset Size Left Size Allocated

...



...

...

...

Shell Scripting

12. Write a shell command to read the current user’s lab2 grade from a file organized as follows:

/homes/cs252/grades.txt

user lab1 lab2 lab3 midterm final hays1 75 100 90 85 75

... ... ... ... ... ...

function get\_grade() {

labPosition=$(ls | grep "lab2" grades.txt | cut -d "2" -f1 | wc -w)

grades=$(ls | grep "$USER" grades.txt)

grades=( $grades )

echo ${lgrades[$lab2Spot-1]}

}

13. Given two versions of a file, write a command to send an email containing the differences to a specified user.

# Usage: changelog <file1> <file2> <send\_to>

function changelog() {

file1=$1

file2=$2

sendTo=$3

difference=$(diff $file1 $file2)

echo "$difference" > tmp-message

echo >> tmp-message

/usr/bin/mailx -s "File Differences" $sendTo < tmp-message

}

14. Given the name of a program, kill an instance of the program the current user has running

# Usage: killByName <name>

function killByName() {

name=$1

pkill -f $name

}

Unix System Calls

Below are a series of small programs. For each one, answer the following three questions:

a) Does the program’s behavior match the specified behavior?

b) If not why is it different?

c) How can it be changed to work as described?

You may make a few assumptions about the programs. They were compiled and run on a 64-bit Linux system like one of the lab machines and data. Also, we may assume that any system calls that are being made will be successful. In the case of the program having different behavior than described, this could be due to either some kind of crash/hang, an issue that causes the behavior to be nondeterministic, or simply a bug causing incorrect output.

15. A program that prints “Hello?” followed by “Can you hear me now?” to standard error, each from a different process.

a) Is the current behavior of this program the behavior described above?

No, it is not.

b) If not, why is it different?

Currently, only “hello?” will print and then the program will hang before reaching second print

c) How can it be changed to work as described?

// hello.c

#include <stdio.h>

#include <unistd.h>

#define MSG\_LEN 128

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

char msg[MSG\_LEN];

int \* pipeFd = malloc(2 \* 4);

pipe(pipeFd);

dup2(pipeFd[0], 0);

dup2(pipeFd[1], 1);

if (fork() == 0) {

puts("Hello?");

fflush(stdout);

fgets(msg, MSG\_LEN, stdin);

} else {

fgets(msg, MSG\_LEN, stdin);

puts("Can you hear me now?");

fflush(stdout);

}

fprintf(stderr, "%s\n", msg);

}

16. A program that prints “M|Ms are the best”.

a) Is the current behavior of this program the behavior described above?

Yes, however it is not exactly ideal.

b) If not, why is it different?

The parent does not wait for the child process and the shell prompt will display before

the message.

c) How can it be changed to work as described?

All that needs to be done is to add in a waitpid or increase the sleep time so that the

shell prompt doesn’t display before the message.

// candy.c

#include <stdio.h>

#include <unistd.h>

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

const char \* str = "M|Ms";

if (fork() != 0) {

str = "Skittlz";

} else {

sleep(1);

printf("%s are the best\n", str);

}

}

17. A program that prints “Something!” 100 times followed by “Empty!”

a) Is the current behavior of this program the behavior described above?

No, it is not.

b) If not, why is it different?

The program prints “Something!” 100 times, but will not print “Empty!”

c) How can it be changed to work as described?

// flow.c

#include<stdio.h>

#include<unistd.h>

#define BUF\_SIZE 128

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

char buf[BUF\_SIZE];

int \* pipeFd = malloc(2 \* 4);

pipe(pipeFd);

if (!fork()) {

dup2(pipeFd[1], 1);

for (int i = 0; i < 100; i++) {

printf("Something!\n");

}

} else {

dup2(pipeFd[0], 0);

while(fgets(buf, BUF\_SIZE, stdin) != NULL) {

printf("Pipeline: %s", buf);

}

printf("Pipeline: Empty!\n");

}

}

Published by Google Drive – Report Abuse – Updated automatically every 5 minutes