CSSE 332 – Operating Systems
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Exam 1 – Paper Part

Name: Key Section: TT CM: Mine

This exam consists of two parts. The first part is to be done on paper without using your computer. The second part of the exam is to be done on your computer. You have the full lab period (a total of three periods) to complete the entire exam.

Instructions: The paper part of the exam is closed book, open notes limited to one double-sided sheet of hand written notes, but **no computer or electronic devices**. Write all of your answers in the spaces provided.

When you complete the paper part of the exam, turn it in to an exam proctor. You may then begin using your computer. **Use of your computer before turning in the paper part of the exam will be considered academic dishonesty.**

To allow sufficient time to work on the computer part of the exam, we suggest using no more than 50 minutes for the paper part.

Please begin by putting your name on the first page and your initials on every page of the exam. We encourage you to skim the entire exam before answering any questions and show all your work to receive partial credit.

	Points available	Your Points
1	10	
2	10	
3	9	
4	9	
5	12	
C	50	
Total	100	

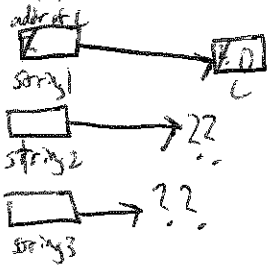
Problem 1 (10 points) Consider the following code.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4
5   void processString1(char *string);
6   void processString2(char **string);
7   void processString3(char **string);
8
9   int main(int argc, char *argv[]) {
10    char c;
11    char *string1, *string2, *string3;
12
13    c = 0;
14    string1 = &c;
15
16    processString1(string1);
17    processString2(&string2);
18    processString3(&string3);
19
20    printf("The first string is: %s\n", string1);
21    printf("The second string is: %s\n", string2);
22    printf("The third string is: %s\n", string3);
23
24    return 0;
25  }
26
27  void processString1(char *string) {
28    char lstring[20] = "Hello world!";
29
30    string = lstring;
31
32    return;
33  }
34
35  void processString2(char **string) {
36    char lstring[20] = "Let's try again!";
37
38    *string = lstring;
39
40    return;
41  }
42
```

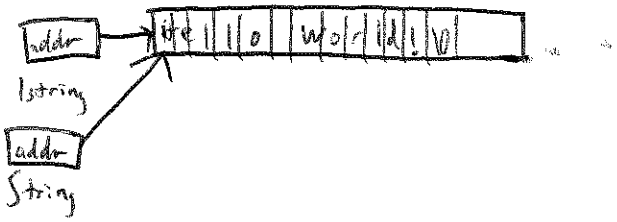
```
43  void processString3(char **string) {
44    char lstring[20] = "Goodbye world!";
45
46    *string = (char *)malloc(sizeof(char)*15);
47    strncpy(*string, lstring, 15);
48
49    return;
50  }
```

Draw the box and pointer diagram at the follow points:

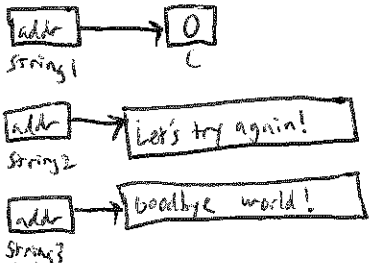
(a) Just before line 16 executes.



(b) Just before line 32 executes.



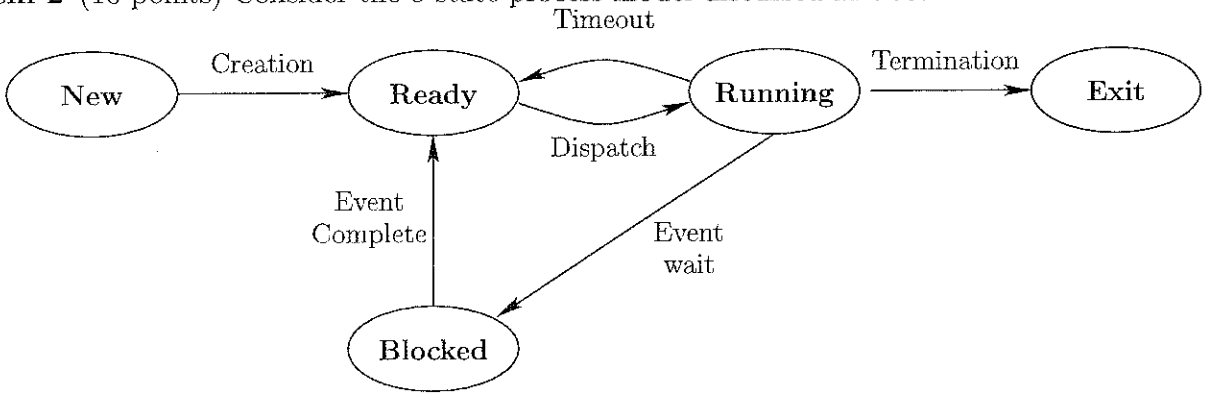
(c) Just after line 21 executes.



(d) Just after line 22 executes.

Same as part c

Problem 2 (10 points) Consider the 5-state process model discussed in class.



(a) (3 points) Which state transitions could cause the currently running process to be preempted?

timeout, event wait

Consider the following sequence of events, which complete in the given order. Assume that the dispatcher runs whenever needed and selects the process at the head of the Ready queue.

- Process P1 is created, admitted, and dispatched.
- Process P2 is created and admitted.
- Process P3 is created and admitted.
- The running process issues I/O request A.
- Process P4 is created and admitted.
- The running process issues I/O request B.
- The running process times out.
- I/O request B completes.
- The running process times out.
- Process P5 is created and admitted.
- The running process exits.

(b) (3 points) Which process is running?

P_2

(c) (4 points) Draw the ready queue.

tail | P_5 | P_4 | head

Problem 3 (9 points) Consider the following set of processes, with their length of CPU bursts and arrival times given in milliseconds. Assume that they are scheduled on a uniprocessor system. For the multilevel feedback queue scheduler, use 3 ready queues indexed from 0 to 2. New processes are admitted in queue 0. Note: for preemptive schedulers, newly admitted processes are added to the ready queue before a preempted process when both events happen at the same time.

Process	Burst Time	Arrival Time
P_1	4	0
P_2	8	1
P_3	3	5
P_4	6	7

(a) (6 points) Draw the Gantt charts that illustrate the execution of these processes using the following scheduling algorithms.

Round Robin (quantum = 2)

0 P_1 2 P_2 4 P_1 6 P_3 8 P_3 10 P_4 12 P_3 13 P_2 15 P_4 17 P_2 19 P_4 21

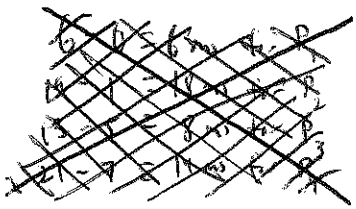
Shortest ~~Remaining Time~~ Job First

0 P_1 4 P_2 12 P_3 15 P_4 21

First Come, First Served

0 P_1 4 P_2 12 P_3 15 P_4 21

(b) (3 points) Compute the average wait time for each algorithm over all processes.
Round Robin (quantum = 2)



$\frac{2+13+5+10}{4} = 7.5$ ms

Shortest Job First $\frac{0+3+7+8}{4} = 4.5$ ms

First Come, First Served

same as SJF 4.5 ms

Problem 4 (9 points) Consider the following code.

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(int argc, char* argv[]){
5    int i=0;
6    pid_t forkpid;
7
8    /* Fork a child process */
9    forkpid = fork();
10
11   while (i++ < 4) {
12     if (forkpid < 0){
13       perror("Fork failed. Exiting!");
14       return 1;
15     } else if (forkpid == 0){
16       if ((i%2) == 1) {
17         printf("Child process, my PID = %d.\n", getpid());
18         printf("Child process, fork PID = %d.\n", forkpid);
19       } else {
20         return 0;
21       }
22     }
23     else{
24       printf("Parent process, my PID = %d.\n", getpid());
25       printf("Parent process, fork PID = %d.\n", forkpid);
26     }
27   }
28   return 0;
29 }
```

For this problem assume that `fork()` **always** succeeds. How many child processes are created?

1

Problem 5 (12 points) **Briefly** answer the following questions.

(a) (3 points) What happens to the other user level threads in a process if one blocks?

all of them are blocked

(b) (3 points) Ignore this one.

(c) (3 points) Why is preemptive scheduling used?

maximize ~~throug~~ throughput and minimize waiting time

(d) (3 points) What is a cpu-bound processes? What is an I/O-bound processes? Which do you want in your system?

A CPU bound proces does computations, and does not need much, if any, I/O.
I/O bound processes do a lot of I/O, and not much computation.
There should be a healthy mix of both types in your system.

CSSE 332 – Operating Systems
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department
Exam 1 — Computer Part

Name: Key Section: TT CM: e

Instructions: This part of the exam is open notes (same as paper part) and open computer.

- You must disable all chat tools (IM, ICQ, IRC, Lync, etc.) before the exam starts.
- **Any communication with anyone other than an exam proctor during the exam could result in a failing grade for the course.**
- You may refer to programs you have written for the course-both those assigned and those you have written for practice. You may not, however, refer to programs written by others except those provided in your textbook or by the course staff.
- Regarding materials on the web, you may only use the course web site and pages linked directly from it. Of course, search engine use is not allowed.
- Write all answers on these pages.
- Submit all code and support files to your svn repository.
- Read the entire examination before starting, then budget your time.
- If you are stuck on problems 2, 3, or 4, you may buy the answer to that question by forfeiting the points for that problem.

	Points available	Your marks
1	10	
2	5	
3	5	
4	5	
5	5	
6	10	
7	5	
8	5	
Total	50	

Checkout the `NumericalAnalysis` project from your svn repository into your csse332 directory. You should be able to do this by simply doing an "`svn update`" on your working copy of the course repository.

The `NumericalAnalysis` project allows you to complete a program for sorting and computing the arithmetic mean of data stored in an input file. The program will be run with 2 command line arguments:

- the path to an input file that contains the data to be sorted and whose arithmetic mean should be computed,
- the path to an output file in which the sorted data will be saved.

A thread will be spawned to read the data from the input file and store it in a buffer. When the data is available in the buffer, **two parallel threads executing in a thread fan** will consume the data in different ways–one will compute the arithmetic mean of the data and the other will sort the data in increasing order, storing the sorted data in a location where it can be accessed to be written to the output file. **The value of the arithmetic mean for the data in the input file provided is 39954.01.**

Another thread will be spawned to write the sorted data to the output file when the data is ready. A summary of the results of the tasks of the various threads will then be displayed to the console.

The following questions all refer to the code in the `NumericalAnalysis` project. The project consists of four C source files, four header files, and one data file as follows.

`main.c` `main.h`	Main module for running the `NumericalAnalysis` application. This module ensures that the input data is read, tasks threads with duties to perform, and displays a summary of the results of the threads' tasks.
`file-processing.c` `file-processing.h`	Provides file reading and writing functionality for the `NumericalAnalysis` application.
`threads.c` `threads.h`	Assigns work to each worker thread based on the type of thread that each is.
`compute.c` `compute.h`	Implements the functions needed to sort data and to compute the arithmetic mean of a list of data.
`data.txt`	Input file containing data to be used by the `NumericalAnalysis` application.

You do not need to understand all of the code provided. You should focus your energy understanding only what you need to answer the questions below. You may assume that each function should do what its name suggests.

Problem 1 (10 points) Write a Makefile to build the executable `main`. Be sure to structure your Makefile so that source files which have not changed are NOT rebuilt. Be sure to add and commit your files to your svn repository.

Problem 2 (5 points) `NumericalAnalysis` does not work as intended. In fact, in it's current state, no thread gets created when run.

Use `gdb` to identify which error is received and where. Note if you run `gdb` in `Emacs` you should start it with "`M-x gud-gdb`."

(a) What error is received?

Segmentation fault

(b) Where (what line number and which instruction) is the error received?

file-processing.c line 16

buffer[count] = value;

Problem 3 (5 points) What is the root cause of the problem you identified in Problem 2?

Cannot access memory at buffer[0]

rawData still point to null

Problem 4 (5 points) Make the necessary modifications to correct the root cause of the problem you identified in Problems 2 and 3. Be sure to commit your changes to svn.

Problem 5 (5 points) Build and run your modified code to continue to test whether the program behaves as intended. Does the program work as intended? If not, describe the issues you noticed.

The data is not sorted, and the mean is 0.00.

Problem 6 (10 points) Make any necessary modifications to address the issues you identified in Problem 5.

Problem 7 (5 points) Describe one additional feature you would add to the NumericalAnalysis program. What changes would you make in the code to support that feature. You are not required to implement the code for this feature; you only need to describe the changes that you would make.

multiple answers, could choose mean, median, mode, maximum, minimum, ...

Problem 8 (5 points) Several macros have been used in this project. Highlight three examples.

NUM_THREADS
LENGTH
MAX_BUFFER_CAPACITY

Have you committed your files and changes to svn?