Name / ID (please PRINT) **Seq#:____** **Seat Number**

CS 3733.001 -- Operating system

Spring 2017 -- Midterm I -- Feb 23, 2017

You have 75 min. Good Luck!

- This is a **closed book/note** examination. But *You can use the 2-page C reference card given to you.*
- This exam has 5 questions in 8 pages. Please read each question carefully and answer all the questions, which have 100 points in total. Feel free to ask questions if you have any doubts about questions.
- **Partial credit will be given, so do not leave questions blank**.

You can get **2pt bonus** credit if you complete the **boldfaced two columns** of the grading table at the end of the exam! Of course first answer the questions in the exam!

---

1. [**20 points]** Answer the following questions with short explanations. You can also draw figures to better explain your reasoning if needed.

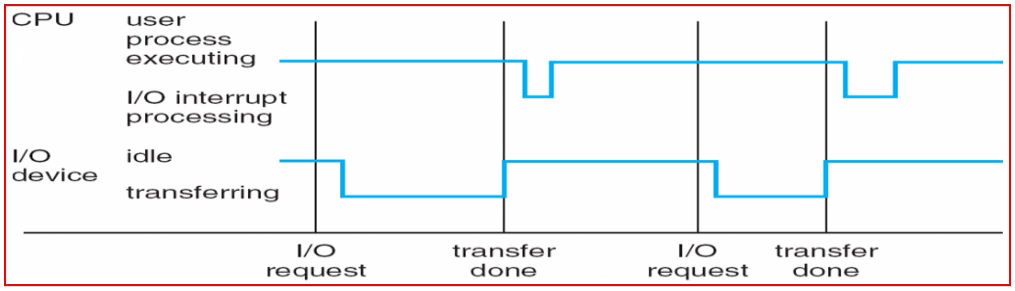
   a. [3pt] What is a **bootstrap** program, and where is it stored?

   [ch 23]Ans: A bootstrap program is the initial program that the computer runs when it is powered up or rebooted. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. Typically, it is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware, within the computer hardware

   b. [5pt] Explain the purpose(s) of Interrupts in an Operating System and how they are handled. You can draw a figure to better explain your steps (control flow) in handling an interrupt.

   The purpose is to let CPU know about asynchrony events such as allowing io devices to inform the CPU about the completion of a task etc.

   1. The interrupt is issued
   2. Processor finishes execution of current instruction
   3. Processor signals acknowledgement of interrupt
   4. Processor pushes PSW(Program Status Word) and PC to control stack
   5. Processor loads new PC value through the interrupt vector
   6. ISR saves remainder of the process state information
   7. ISR executes
   8. ISR restores process state information
   9. Old PSW and PC values are restored from the control stack



c. **[3pt]** Explain the purposes of fork(), exec(), and wait() in process creation and termination.

A process (parent) creates another process (child) using the system call fork()
The new child process has a separate copy of the parent process's address space (code, data etc.).
exec() loads a new program into the adress space of the process that calls it, and passes paramters e.g., argv etc.
wait() : parent blocks until the child finishes/terminates and clears the PCB so there will be no zombie!

d. **[3pt]** Explain the difference between an I/O-bound process and a CPU-bound process.

ch 3- 23 Ans: The differences between the two types of processes stem from the number of I/O requests that the process generates. An I/O-bound process spends more of its time seeking I/O operations than doing computational work. The CPU-bound process infrequently requests I/O operations and spends more of its time performing computational work.

e. **[3pt]** What are the key performance criteria for scheduling algorithms? Explain turnaround time.

Fairness, efficiency, waiting time, response time, throughput, and turnaround time;

Turnaround Time
Time from submission to termination or
Sum of CPU time, I/O time, and waiting time

f. **[3pt]** If a IO system call (e.g. read) returns -1 and error is EINTER, how should we interpret this? And give 1-2 lines of code to show how to handle that error.

This is not an actual error, while waiting to read an interrupt happened. So we need to repeat it;

```
while( ((retval=read())<0) && retval==-1 && errno == EINTER) ;
```

2. [**20 points**] Program and Process, command-line arguments, `static` keyword

a. [**15 points**] Show the memory image of the following program when it is executed (Specifically show where the variables reside, and how their values change). Also give the output of the program.

```
#include <stdio.h>
#include <stdlib.h>

int myfunc(int E);

int A, B=6;

int main(int argc, char *argv[]){
   int C, D;
   A = 4;
   C = myfunc( 5 );
   D = myfunc( 7 );
   printf("A=%d B=%d C=%d D=%d\n",
                A, B, C, D);
   return 0;
}

int myfunc(int E) {
  static int F=0;
  F = F + E;
  return  F;
}
```

OUTPUT: (**5pt**)

....

....

```
A=4 B=6 C=5 D=12
```

(10pt)

Variable name	Memory Image	
argc argv		
C D E	? 5 ? 12 5 7	STACK
		HEAP
A	? 4	Un-Init Data
B F	6 0 5 12	Init Data
	main() code myfunc() code	TEXT

b. [**5pt**] In the above code, suppose we remove the "`static`" from "`static int F=0;`" and just have "`int F=0;`" in `myfunc()`. Explain how this may affect the program and what might be the new output.

F will be allocated in STACK and re-initialized every time when myfunc() is called.
So the output will be

```
A=4 B=6 C=5 D=7
```

3. [20 Points] CPU Scheduling

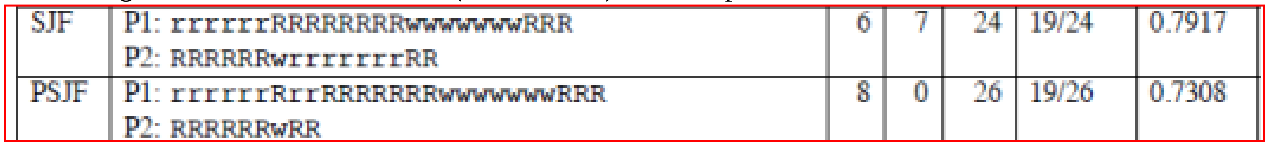
a. [**12 points**] Consider two processes as in Assignment 2/Quiz 4, where each process has two CPU bursts with one I/O burst in between on a single core CPU. Suppose P1 and P2 have the following life-cycles:

P1 has x1=**8**, y1=**7**, z1=**3** units for the first CPU burst, I/O burst, second CPU burst, respectively.

P2 has x2=**6**, y2=**1**, z2=**2** units for the first CPU burst, I/O burst, second CPU burst, respectively.

Both arrives at the same time (in case of ties, pick P1) and there is no other processes in the system.

For each of the scheduling algorithms below, create Gantt charts as you did for the Quiz 4. Fill each box with the state of the corresponding process. Use **R** for **R**unning, **W** for **W**aiting, and **r** for **r**eady. Calculate the waiting times and CPU utilization (as a fraction) for each process and fill in the table below.



SJF	P1: rrrrrrRRRRRRRRwwwwwwwRRR P2: RRRRRRwrrrrrrrrRR	6	7	24	19/24	0.7917
PSJF	P1: rrrrrrRrrRRRRRRRwwwwwwwRRR P2: RRRRRRwRR	8	0	26	19/26	0.7308

Gantt Charts for SJF (Shortest Job First, non-preemptive) [4pt]

a) SJF 5 10 15 20 25 30

P1																																			
P2																																			

Gantt Charts for PSJF (Preemptive SJF) [4pt]

b) PSJF 5 10 15 20 25 30

P1																																			
P2																																			

Waiting time and CPU utilizations [4pt]

Algorithm	Waiting times in ready queue			Finish time		Longest Schedule length	CPU utilization
	Process 1	Process 2	average	Process 1	Process 2		
b) SJF							
c) PSJF							

b. [**8 points**] Suppose we have a system using **multilevel queuing**. Specifically there are two queues and each queue has its own scheduling algorithm: QueueA uses RR with quantum 3 while QueueB uses FCFS. CPU simply gets processes form these two queues in a weighted round robin manner with 2:1 ratio (i.e. it gets **two** processes from QueueA then gets **one** process from QueueB, and then gets **two** processes from QueueA then gets **one** process from QueueB, and so on), But when it gets a process from QueueA, it applies RR scheduling with quantum 3. When it gets a process from QueueB, it applies FCFS scheduling.

Draw the Gantt charts (**5pt**) and compute waiting times (**3pt**) for the following four processes: P1, P2, P3, P4 on a single core CPU. Assume these processes arrived at the same time and in that order. Each process has a single CPU burst time of 5 units. There is no other processes or IO bursts.

	ratio					5					10					15					20	
QueueA RR q=3	2	P1	R	R	R								R	R								
		P2				R	R	R							R	R						
QueueB FCFS	1	P3							R	R	R	R	R									
		P4															R	R	R	R	R	

Compute Waiting times in ready queue				
P1	P2	P3	P4	average
8	10	6	15	39/4= 9.75

4. [**20 points**] Unix I/O and file operations

a. [**10 points**] Suppose that the file "`tmpdata.txt`" contains "`abcdefghijk`". If the following code is executed correctly without generating any errors.

```
1: int fd;
2: char buf[6] = "12345";
3: fd = open("tmpdata.txt", O_RDONLY);
4: fork( );
5: read(fd, buf, 2);
6: read(fd, buf+2, 2);
7: printf("%d: %s  ", (long)getpid(), buf);
```

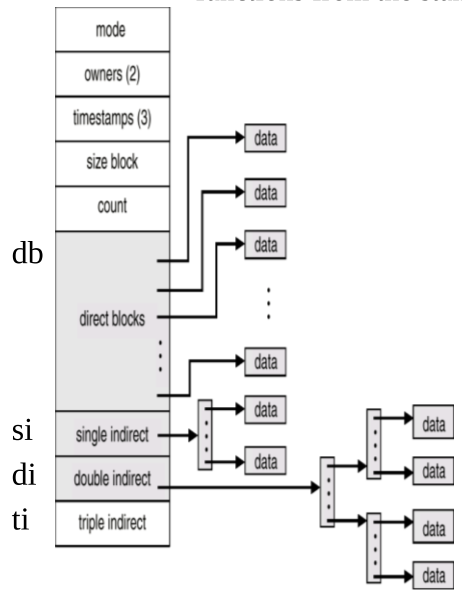
i). [**5pt**] Explain if the following two outputs are possible or not? Why/why not? Suppose parent's pid is 7 while child's pid is 8.

7:a2bc5 **8:a2b45**	**Not possible, since system file table entry and thus offset is shared, they need to read different char!**
7:a2c45 **8:b2de5**	**Yes, possible, since system file table entry and thus offset is shared, each reads 1 or 2 char and advance the offset. accordingly the other reads the next one.**

ii). [**5pt**] What could be the outputs if the lines 3 and 4 are exchanged? Write at least 3 possible outputs. Suppose parent's pid is 7 while child's pid is 8.

```
7:abcd5    7:a2bc5    7:abc45
8:abcd5    8:a2b45    7:a2bc5 ....
```

b. [**10 points**] Suppose you are given the inode structure as shown below. Assume that our inode structure has 10 direct pointers, and that Block size is 8K bytes and pointers are 4 bytes. Write a function **`int count_di(struct inode *myinode, char *word);`** which counts the number of blocks containing the given string **word** only in **double indirect** blocks. *Note that double indirect level might be partially filled with blocks! So if there is no more block or level the corresponding pointer will contain NULL.* Please take that into account! If there is a data block, we assume that this data block contains a string text. Thus, if needed, you can use strcmp() or any other functions from the standard libraries as they are included here.



```
int count_di(struct inode *myinode, char *word){
   int i, j, k;
   int count=0;
   int num_blk =   8 * 1024 / 4;

   if (inode->di)
     for(i=0; i < num_blk; i++)
       if (myinode->di[i])
         for(j=0; j < num_blk; j++)
            if (myinode->di[i][j] &&
                strcmp(word, myinode->di[i][j])==0)
               count++

     return count;
   }
```

5. [20 points] **IPC and pipes -fifo**

If you ever used **script** command in Unix, then you know what it does. If not, here what [man script] says "the **script** command makes a record of everything printed on your screen. The record is written to a log file. The script command forks and creates a sub-shell, according to the value of $SHELL, and records the text from this session. The script ends when the forked shell exits or when CTRL-D is typed."

For example, it is executed as follows and it logs everything on the screen in logfile.txt

```
> script logfile.txt
Script started, file is logfile.txt
> ls
a1.txt b1.txt prog.c
> exit
exit
Script done, file is logfile.txt
```

Suppose we like to have a simple implementation of script.c as follows: script program (parent) creates a child process which excel a shell and interacts with the user. But for parent to do its job, it needs to get everything that child shell prints on the standard output through parent own standard input. Hope you see how a pipe will be useful here: **child[1]-->pipe-->[0]parent**. Accordingly, the parent writes everything that it gets through the pipe to both its standard output and the logfile.

Now you are asked to complete the following program so you can create the necessary pipe, child process and connect them as explained in the above scenario. You can ignore most of the error checking to make your solution clear, but you need to close all unnecessary file descriptors and check what read-write etc return.

```
/* your simple implementation of script.c */
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
int main (int argc, char *argv[]) {
   int mypipe[2];
   int log_fd, childpid, numread, numwrite;
   char buf;
   /* 5pt - create pipe and child process */

        pipe(mypipe);
        childpid = fork();


   if(childpid == 0){ /* child */
      /* 5pt - child sets up the pipe */

      dup2(mypipe[1], STDOUT_FILENO);
      close(mypipe[0]);
      close(mypipe[1]);

     execl("/bin/bash","shell",NULL);
     perror("cannot start shell");
     return 1;
   }

   /* continue in the next page */
```

```
    /* continue problem 5.b here */

    log_fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC,
                           S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );

    /* 5pt - parent sets up the pipe */

    dup2(mypipe[0], STDIN_FILENO);
    close(mypipe[0]);
    close(mypipe[1]);



    /* 5pt - parent reads from the pipe, writes to both */


    while(1){
       numread = read(STDIN_FILENO ,&buf, 1);
       if (numread <= 0) break;          // what if EINTER ?

       vilsnumwrite = write(STDOUT_FILENO, &buf, 1);
       if (numwrite<=0) break;           // what if EINTER ?

       numwrite = write(log_fd, &buf, 1);
       if (numwrite<=0) break;           // what if EINTER ?

  } // end of while

  close(log_fd);
  fprintf(stderr,"Script: finished monitoring....\n");
  wait(NULL);
}
```

You can get **2pt bonus** credit if you complete the **boldfaced two columns** of the grading table below. Please do this after answering the questions in the exam. Thanks!

Question	Topic	Possible Points	**Difficulty level of this question 1: Easiest 5: Most difficulty**	**Student Expects**	Student Received
1	Review questions, short explanations	20			
2	Program and Process, static keyword	20			
3	CPU Scheduling	20			
4	Unix I/O and file operations	20			
5	IPC and pipes -fifo	20			
	Bonus	2			
Total		100+2			