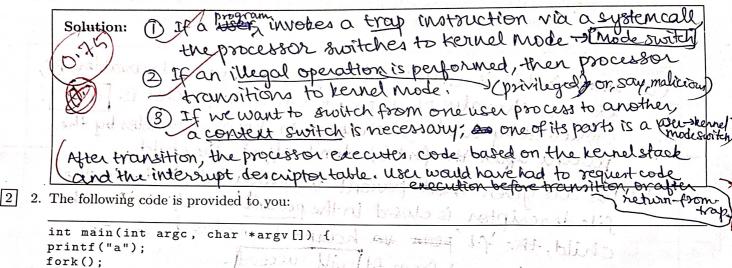
1112	OT.	u	Ü	IU	115

- 1. Please write the answer for each question ONLY in the space provided below the question.
- 2. No electronic gadgets are allowed.
- 1. Name three ways in which the processor can transition form user mode to kernel mode? Can the user execute arbitrary code after transition?



return 0;
}

a) Assuming fork() succeeds and printf() prints its outputs immediately (no buffering occurs), what are possible outputs of this program?

b) Assuming fork() might fail (by returning an error code and not creating a new process) and printf() prints its outputs immediately (no buffering occurs), what are possible outputs of the same program as above?

Solution: a) With successful fork, trusput (fully) is: abb

The two b's are each printed by parent and child,
in an order that depends upon the system.

(2) b) If forker fails, no child process is created,
so only one b' is printed, resulting in a

(complete) output of: ab

2 3. Consider a parent process P that has forked a child process C in the program below.

```
trus la sign
 int a = 5;
                                                   without After about 8
 int fd = open(...) //opening a file
 int ret = fork();
 if (ret >0) { // parent block
close(fd);
_a = 6;
 else if(ret==0) { //child theel
 printf("a=%d\n", a);
 read(fd, something);
```

After the new process is forked, suppose that the parent process is scheduled first, before the child process. Once the parent resumes after fork, it closes the file descriptor and changes the value of a variable as shown above. Assume that the child process is scheduled for the first time only after the

(a) What is the value of the variable a as printed in the child process, when it is scheduled next? Explain.

(b) Will the attempt to read from the file descriptor succeed in the child? Explain.

- (a) when fork() is called, child gets a copy of all variables, PC, et. So, the value of a printed in child process is a=5. (followed by newline). changes in local variables by the parent after the fork() don't affect the child. (b) We are given that parent is scheduled first. So, the
- file descriptor is closed in the parent's context. For the child, the 'fd' point to hasn't been closed, so, the attempt to read from fd will succeed. Lo As long as file is in read made, and file is valid (path, et.)
- 1 4. Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and 'X'. In what order should they be run to minimize average response time (of course your answer will depend on X). Explain the different possible cases based on the value of 'X'.

Solution: For minimum overage response time, [Round Robin] scheduling would be ideal. Since Round Robin deals with time slices, better assume (which we can assume to be I here), the average response time is (0+1+2+3+4) as long as X≥1. This is 2 time what if we were not allowed to do time straing? We were could take a look at how SJF or STCF works. For SJF, we would try to run the jobs in increasing order of runtimes. If x>9, then avg. response time= 0+3+8+14+23 = 48=19.6 time units For STCF, the same thing would occur since we assume that alliots

could show more cases if space 1th was more feasible here

- 2 5. Consider the following preemptive-priority-scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the processor (in the ready queue, but not running), its priority changes at a rate  $\Lambda$  [i.e P(t) = P0 +  $\Lambda$  \* (t - t0) where to is the time at which the process joins the ready queue]; when it is running, its priority changes at a rate B. All processes are given a priority of zero when they enter the ready queue. The parameters A and B can be set to give many different scheduling algorithms.
  - (A) What is the algorithm that results from  $B > \Lambda > 0$ ? Explain how
  - (B) What is the algorithm that results from A < B < 0? Explain how

	Solution: (A) If B>A>O, priority increases faster in running state
	thour in ready state. This tends towards first come of
1 -	4 Since running processes get more-and-more first serve 4
	The state of the s
	b) + 1000, 18102-ity decreases taste to symming many
	The true of the ready quette, this things to the
	I round Robins stille sturning processes are more than
	to get "kicked ond", and resumed later on e

- [2] 6. The Multi-level Feedback Queue (MLFQ) is a fancy scheduler that does lots of things. Are the below given things TRUE/FALSE about the MLFQ approach? Give reasons for your answer. Please note, merely stating TRUE or FALSE without justification/ reason will not fetch any marks.
  - a) MLFQ learns things about running jobs
  - b) MLFQ starves long running job
  - c) MLFQ uses different length time slices for jobs
  - d) MLFQ forgets what it has learned about running jobs sometimes

a) [TRUE]. since MLFQ decreases priority level of jobs that Solution: run until the end of their time slice, and doesn't do the same for so ones that relinquish CPUCIO Hence, MLFQ Learns about running jobs. — 60200 Life their CPU-intensive/ Ip bound nature) FALSE, subject to the fact that post, on a periodic/segular basis, all jobs are sent to topmost queue. This is Rule Bir c) FALSE Atmestices stay the same, and after each time slice, a decision is made based on the MLFQ, of which job to sun. Lo Rote that time slice differs from CPU burst) . This "forgetting" follows from (6b), where the priorities are periodically reset. This prevents long jobs from storvation