NIA: STUDENT NAME:

Useful functions:

```
int close(int fd)
int open(const char *path, int oflag)
int write(int fd, void *buf, int nbyte)
int read(int fd, void *buf, int nbyte)
int wait(int *status)
pid_t fork(void)
void exit(int status);
int pipe(int pipefd[2]);
```

I. **(30 points)** A multi-level queue process scheduler for UNIX uses round-robin in all queues. At time t=0 the state of the queues is the following:

Maximum priority queue:
Process A: 250 instructions + 50 I/O + 300 instructions
Process B: 500 instructions + 200 I/O + 100 instructions

Medium priority queue:
Process C: 750 instructions
Process D: 150 instructions + 300 I/O + 50 instructions

Minimum priority queue
Process E: 1500 instructions

a) What is the process execution order if the quantum of all priority queues is 500 instructions?
b) What is the process execution order if the quantum of all priority queues is 200 instructions?
c) What is the CPU utilization if the context switch time is 50 instructions in case a)
d) What is the CPU utilization if the context switch time is 100 instructions in case b)

Solution:

a)

Time	Executing process	Preemption reason
0-250	A	I/O
250-750	B	I/O
750-1050	A	termination
1050-1150	B	termination
1150-1650	C	end of time share
1650-1800	D	I/O
1800-2050	C	termination
2050-2550	E	end of time share
2550-2600	D	termination
2600-3600	E	termination

b) Round-Robin 200 instrucciones

Time	Executing process	Preemption reason
0-200	A	end of time share
200-400	B	end of time share
400-450	A	I/O
450-650	B	end of time share

NIA: STUDENT NAME:

650-850	A	end of time share
850-950	B	I/O
950-1050	A	termination
1050-1250	C	end of time share
1250-1350	B	termination
1350-1500	D	I/O
1500-1700	C	end of time share
1700-1900	E	end of time share
1900-2100	C	end of time share
2100-2150	D	termination
2150-2300	C	termination
2300-3600	E	termination

c) Number of context switches = 9

CPU util = (3600/ (3600 + 9*50)) *100 = 88,88 %

d) Number of context switches = 15

CPU util = (3600/ (3600 + 15*50)) *100 = 82,75 %

II. **(30 points)** Develop a small program in C, which uses system calls for providing the following functionality.

- A father creates a son S1.
- S1 creates a son S2.
- The father sends an integer to S1, receives an integer from the S2, prints it, expects S1 to finish and terminates.
- S1 receives the integer from the father, sends it to S2, waits for S2 to finish and terminates.
- S2 receives the integer from S1, sends it to the father and terminates.
- All the communication must be done with pipes.

Solution:

```
int p1[2], p2[2]; pipe(p1);pipe(p2);
int i;

if (fork()==0) {
                    if (fork()==0) {
                              read(p2[0],&i,4);
                              write(p1[1],&i,4);
                              return;
                    }
                    else {
                              read(p1[0],&i,4);
                              write(p2[1],&i,4);
                              wait(NULL);
                    }
}
else {
                    write(p1[1],&i,4);
                    read(p2[0],&i,4);
```

```
                wait(NULL);
                printf("i=%d",i);
        }
```

III. (**10 points**) Given the following code:

```
for (i=1; i<4; i++){
  pid=fork();
  if (pid==0){ //Child
    printf ("Child %d, pid =%d, ppid%d\n",i,
   getpid(),getppid());
  }
  else
    printf ("The father created %d children \n",i);
}
```

Show the messages printed on the screen. Assume that the father has the pid 700 and the children the pids 701, 702.

Solution:

The father created 1 children
Child 1 pid=701 ppid=700
The father created 2 children
Child 2 pid=702 ppid=700
The father created 3 children
Child 3 pid=703 ppid=700
The father created 2 children
Child 2 pid=704 ppid=701
The father created 3 children
Child 3 pid=705 ppid=701
The father created 3 children
Child 3 pid=706 ppid=702

IV. **(10 points)** Quiz questions (***NOTE: A correct answer scores 1point, a wrong answer subtracts 0.25 points, an unanswered question scores 0 points***)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Answer	A	B	C	B	A	B	A	A	B	B

1. Round robin scheduling is essentially the preemptive version of:
a) FIFO b) Shortest job first c) Priority scheduling
2. A major problem with priority scheduling is:
a) Deadlock b) Starvation c) Low priority d) None of the above
3. Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?
a) Time-sharing b) Preemptive scheduling c) Multiprogramming
4. The number of processes completed per unit time is known as:
a) Output b) Throughput c) Efficiency d) Capacity
5. Threads are cheaper to create than processes

a) True b) False

6. A blocking kernel-scheduled thread blocks all threads in the process

a) True b) False

7. A blocking user-level thread blocks the process

a) True b) False

8. All kernel-scheduled threads of a process share the same virtual address space

a) True b) False

9. The operating system is not responsible for resource allocation between competing processes

a) True b) False

10. System calls do not change to privilege mode of the processor.

a) True b) False

V. **(20 points)** Give definitions of the following concepts in maximum 3 lines:

1. Multiprogramming
2. Shell
3. Context switch
4. Queue
5. Differences process-thread
6. Process starvation