**Operating Systems - Midterm exam – March 19th 2014**

***Computer Science Degree***

**RULES:**

* The exam duration is 1:50 **hours.**
***** It is not allowed to use any book or notes. Electronic devices are also not allowed.

---

**Name and surname:**

**Group:**

**Exercise 1 (20 points). QUIZ**

Answer the Quiz questions writing the letters of each correct answer. Each wrong answer deducts 2.5 points from the exercise grade. Non answered questions don't deduct any points (they are ignored by the reviewer).

**Write here the quiz type (A or B):**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

**Exercise 2 (40 points).**

The following table shows a set of processes that have to be executed in a system. There are three possible phases: an initial CPU burst, then an I/O burst and then a final CPU burst. A zero value means that the associated phase is not done.

| Process | Arrival time (ms) | First CPU burst (ms) | I/O burst (ms) | Second CPU burst (ms) | Priority (the bigger the higher) |
|---|---|---|---|---|---|
| Gzip | 0 | 3 | 6 | 2 | 0 |
| Scp | 1 | 3 | 3 | 1 | 2 |
| chkmem | 4 | 3 | 1 | 0 | 8 |
| Comput | 5 | 4 | 0 | 0 | 3 |

Assume that all I/O bursts are on the same device (hard disk) with an immediate response time (if the I/O device is free it processes the data immediately). **The I/O queue is FIFO**. You are asked to complete the following questions. Please provide a reasoned answer:

1. Complete the following execution table for the **Shortest-Job-First (SJF)** scheduling. Assume that it is **nonpreemptive**. Each one of the table cells can have one of the following values: W (waiting in the ready queue), R (CPU burst, running in the CPU), WIO (waiting in the I/O queue), IO (I/O burst, performing I/O), -- (no value, not applied). **Provide a reasoned answer explaining how the SJF scheduling works.**

| | t=0 | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 | t=11 | t=12 | t=13 | t=14 | t=15 | t=16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gzip | | | | | | | | | | | | | | | | | |
| Scp | | | | | | | | | | | | | | | | | |
| chkmem | | | | | | | | | | | | | | | | | |
| Comput | | | | | | | | | | | | | | | | | |

| | Turnaround Time | Waiting Time CPU | Waiting Time IO |
|---|---|---|---|
| Gzip | | | |
| Scp | | | |
| chkmem | | | |
| Comput | | | |

**2.** Complete the following execution table for a **Priority scheduling** algorithm. Assume that it is **preemptive**. Each one of the table cells can have one of the values depicted in the previous point. **Provide a reasoned answer explaining how the Priority scheduling works.**

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gzip | | | | | | | | | | | | | | | | | |
| Scp | | | | | | | | | | | | | | | | | |
| chkmem | | | | | | | | | | | | | | | | | |
| Comput | | | | | | | | | | | | | | | | | |
| | t=0 | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 | t=11 | t=12 | t=13 | t=14 | t=15 | t=16 |

| | Turnaround Time | Waiting Time CPU | Waiting Time IO |
|---|---|---|---|
| Gzip | | | |
| Scp | | | |
| Chkmem | | | |
| Comput | | | |

3. Obtain the system throughput for the Shortest-Job-First (SJF) and the Round-Robin (RR) algorithms.

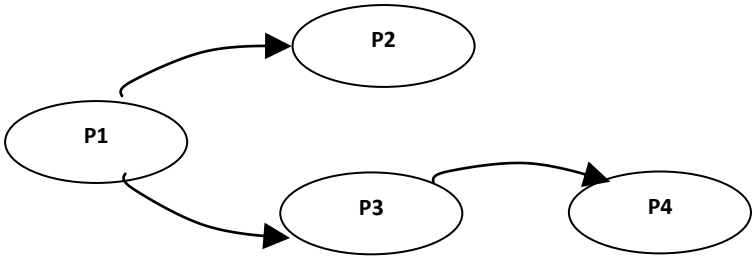
A. [Theoretical question] How the ready-queue is implemented? What kind of structures does it internally manage?

B. [Theoretical question] Can the Shortest-Job-First (SJF) be preemptive? If so, explain how it works with a simple example.

C. [Theoretical question] Is there any difference in the scheduling of the SJF algorithm if a process is running in the background or foreground?

**Exercise 3 (40 points).**
Write a C program that fulfills the following specifications:

1. It has the following process hierarchy: process P1 has two child processes. The first one, P2, doesn't have any child processes. The second one, P3, has another child process (P4).




2. Some of the processes are communicated in the following scheme:

    - Before creating any process, P1 creates a new file (deleting it in case of existing) named output2P2.txt and writes in it the sequence of chars "HelloP2". P1 also creates another new file (deleting it in case of existing) named P4.txt and writes in it the sequence of chars "HelloP4". At this point, P1 doesn't close the files.

    - Following that, P2 reads and displays the contents of P2.txt and P4 reads and displays the contents of output2P4.txt. P3 doesn't read any file. **Note: P2 and P4 are not allowed to create new file descriptors.**

3. After creating P4, **P3 executes** the system command "ls", located in "/bin/ls".

4. When P2 terminates, it exists returning the value of 10.

5. When P2 and P3 terminate, P1 captures exit value of these two processes (it is not necessary to print them) and closes the existing files.

A. Is there any design problems in this code?

B. [Theoretical question] Are there any more efficient alternatives to using files for implementing this code? If so, explain briefly what approach you would follow (it is not necessary to write code).

C. [Theoretical question] Is it possible to write a code based on threads that does the same as this code? Provide a reasoned answer (it is not necessary to write code).

**Operating Systems - Midterm exam – March 19th 2014**

***Computer Science Degree***

**POXIS calls**

```
pid = fork()
pid = waitpid(pid, &statloc, opts)
s = wait(&status)
s = execve(name, argv, envp)
exit(status)
size = brk(addr)
pid = getpid()
pid = getpgrp()
pid = setsid()
l = ptrace(req, pid, addr, data)
s = sigaction(sig, &act, &oldact)
s = sigreturn(&context)
s = sigprocmask(how, &set, &old)
s = sigpending(set)
s = sigsuspend(sigmask)
s = kill(pid, sig)
residual = alarm(seconds)
s = pause() Suspend the caller until the next signal
fd = creat(name, mode)
fd = mknod(name, mode, addr)
fd = open(file, how, ...)
s = close(fd)
n = read(fd, buffer, nbytes)
n = write(fd, buffer, nbytes)
pos = lseek(fd, offset, whence)
s = stat(name, &buf)
s = fstat(fd, &buf)
fd = dup(fd)
s = pipe(&fd[0])
s = ioctl(fd, request, argp)
s = access(name, amode)
s = rename(old, new)
s = fcntl(fd, cmd,
s = mkdir(name, mode)
s = rmdir(name)
s = link(name1, name2)
s = unlink(name)
s = mount(special, name, flag)
s = umount(special)
s = sync()
s = chdir(dirname)
s = chroot(dirname)
s = chmod(name, mode)
uid = getuid()
```