Name: ....

Informatics Engineering/Computer Science – University of Minho

Test Model of Operating Systems

April 16, 2025 – Duration:

Number: ....

0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6
7	7	7	7	7	7
8	8	8	8	8	8
9	9	9	9	9	9

Instructions: *Fill in the student's name and number at the top of this sheet. In addition, completely color in (■) the boxes corresponding to the digits of the number in the grid next to it (units digit in the rightmost box). Do the same to mark the answers you think are correct.*
Do not use shaded areas!

Important: *This document contains several examples of possible test/exam questions. Note that the number and type of questions in the actual test/exam may vary (e.g., it will be adjusted to the time to do it)*

In the source code examples, assume that the system calls execute correctly and without errors (unless the question explicitly states otherwise).

Some relevant system calls

Processes

- pid_t fork(void);
- void _exit(int status);
- pid_t wait(int *status);
- pid_t waitpid(pid_t pid, int *status, int options);
- WIFEXITED(status);
- WEXITSTATUS(status);
- int execlp(const char *file, const char *arg, ...);
- int execvp(const char *file, char *const argv[]);

File System

- int open(const char *pathname, int flags, mode_t mode);
- int close(int fd);
- int read(int fd, void *buf, size_t count);
- int write(int fd, const void *buf, size_t count);
- long lseek(int fd, long offset, int whence);
- int pipe(int filedes[2]);
- int mkfifo(const char *path, mode_t mode);
- int dup(int oldfd);
- int dup2(int oldfd, int newfd);

Group I

10 values

Example questions on the theoretical component

Question 1 The process scheduler of an operating system typically tries to have a balanced mix of CPU-intensive and I/O-intensive processes ready to be executed by the CPU. What is the reason for this concern on the part of the process scheduler? Justify your answer.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 2 Consider a service that allows the execution of programs (*e.g.* text editors, *bash*, *chatbots*) that require frequent interaction with users. As this service does not always have the capacity to run all the programs simultaneously, it is necessary to stagger their execution.

Assuming that this service supports preemption of processes (*i.e.*, it can interrupt a program during its execution and resume it later), what scheduling algorithm would you choose for it? Justify your answer by indicating **an advantage** and **a possible concern or disadvantage** of your choice.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 3 Modern operating systems use paging to manage the memory of processes in user space. In addition, paging is often combined with mechanisms for swapping pages from memory to disk.

1. Indicate **an advantage** of combining pagination with *swapping* mechanisms. Also indicate **a concern** that the operating system should have in order to make *swapping* of pages efficient. Justify your answer.
2. When setting up the *swap* area on a given computer, is it preferable to use a file system, or directly use a disk partition (*raw*) to support it? Justify your answer.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 4 While analyzing the behavior of a server's hard disk drive (HDD) access, you noticed that some applications exhibit poor performance (*i.e.*, requests to the disk from these applications take a long time to be served). When you analyzed the operating system, you noticed that it was configured with a Shortest Seek Time First (SSTF) scheduling algorithm. Remember that this algorithm chooses the next requests to serve according to the proximity of the disk cylinders they access (*i.e.*, minimizes disk head movement).

Indicate **a possible reason** for the poor performance of the applications. Also, indicate **an alternative scheduling algorithm** that could be fairer and improve the performance of these applications. Justify your answer.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 5 Imagine that you wanted to develop a service for orchestrating (*i.e.*, executing and scheduling) tasks (programs) on a computer. This service knows, from the outset, the execution time of each task, but does not support preemption of processes (*i.e.*, cannot interrupt a process during the execution of a task). Taking the above into account, name the scheduling algorithms you could use for the service in question.

- A Multi-Level Feedback Queue (MLFQ).
- B Shortest Job First (SJF).
- C First-Come First-Served (FCFS).
- D Round-Robin (RR).
- E *None of these answers are correct.*

Question 6 The *paging* and *segmentation* techniques aim to solve different problems regarding the allocation of central memory by the processes running on a given operating system. Internal and external fragmentation of central memory are two common phenomena that can lead to wasteful use. Assuming that the paging technique is configured with an identical page size and *frame*, check the true answers.

- A Segmentation and pagination do not suffer from any kind of fragmentation.
- B Memory segmentation can suffer from external and internal fragmentation.
- C Memory paging can generate external fragmentation, but not internal fragmentation.
- D Memory paging can generate internal fragmentation, but not external fragmentation.
- E *None of these answers are correct.*

Question 7 A file system is responsible for deciding how best to store file data on disk. Considering a mechanical hard disk drive (HDD), it is advantageous to store the data for each file contiguously on the disk. Do you agree with this statement? Mark the correct answers.

- A No, sequential or random accesses on an HDD disk have the same performance.
- B Yes, random access times are higher than sequential access times on HDD disks.
- C Yes, since it tends to increase the average *seek* time of the HDD disk.
- D Yes, since it tends to decrease the average *seek* time of the HDD disk.
- E *None of these answers are correct.*

Group II 10 values

Example questions on the practical component

Question 8 Consider a company that uses a single file written in **binary** format to store all the individual records of its employees. Each record contains an employee's name, job title and salary in the company, according to the following structure:

```
struct RegistoF {
    char name[20];
    char position[20];
    int salary;
};
```

1. Write the function `void increaseSalaries(char* file, char* position, int value, long N, int P)` that updates the *file* with N records in order to increase the salary of employees with a given *position* by *value*. The function must trigger a concurrent update with P processes.

2. Write the function `int validateSalaries(char* file, char* position)` which validates the salary of all employees with a given *position*. The function should return 1 if any employee is paid less than the minimum wage. Otherwise, it should return 0. To solve this exercise, you should take advantage of the following **executable files**:

   - `./filterPosition <file> <position>` - searches the *file* of records for employees of a given *position* and writes the respective records to the `stdout`.
   - `./validateMin` - reads records from `stdin`, in the same format produced by the executable file `filterPosition`, and ends with the exit code 1, if any employee receives less than the minimum wage, or 0 otherwise.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 9 Consider a program `translator` that reads lines of text from `stdin`, translates them into Portuguese, and writes the translated lines to `stdout`. This program takes no arguments.

1. Implement the **function** `void translate_and_filter(char* file_path, char* password)` that should translate the contents of a given file, whose path is passed as an argument (*file_path*), and write to the `sdtout` only the translated lines that contain a certain keyword (*password*).

   You should take advantage of the `translator` program and the `grep` command when implementing this function. Remember that when you invoke the `"grep SO"` command, it will read lines from `stdin` and only write the lines containing the word *"SO"* to `stdout`.

2. Implement the **function** `void filterN(char* file_paths[], int total_files, char* password)` which should trigger the processing (*i.e.,* translation and filtering) of several files in parallel. This function receives as arguments a list of file paths (*file_paths*), the total number of files (*total_files*), and a keyword (*keyword*). Your solution should use the `translate_and_filter` function developed in the previous question.

   Assume that only N processes can run simultaneously.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 10 Consider a search engine based on Artificial Intelligence which, given a user query (argument *prompt*), responds in textual format. This engine is made up of a chain of programs.

- For each request, the executable file `filter` (*i.e.,* `filter <prompt>`) selects files that contain content related to the question, writing their paths in binary format to `stdout`.
- Meanwhile, the executable file `execute` (*i.e.,* `execute <prompt>`) reads from `stdin` the paths of the filtered files, one by one, and generates binary content according to the user's query, which is written to `stdout`. Assume that the binary content generated is always less than `PIPE_BUF` bytes.
- The executable file `merge` (*i.e.,* `merge <prompt>`) reads, from `stdin`, binary content resulting from interrogations and generates an answer for the user in textual format (written in `stdout`).

1. Write the `SOGPT` program to implement the behavior described above. Optimize the `execute` phase by triggering its concurrent processing with N processes.
2. Consider the client program below (`search_prompt`) which uses the named pipe *fifo_server* to send a search request to the server program. The request (structure *Req q*) contains the client program's *pid* and the *prompt* to be processed by the server. Finally, the client program notifies the user when the request is complete, indicating its identifier.

```
1   int main (int argc, char * argv[]) {
2       Req q;
3       q.pid=getpid();
4       q.prompt=strdup(argv[1]);
5
6       char fifoc_name[30];
7       sprintf(fifoc_name, "fifo_client_%d", q.pid);
8       mkfifo(fifoc_name, 0666);
9
10       int fds = open("fifo_server", O_WRONLY);
11       write(fds, &q, sizeof(q));
12       close(fds);
13
14       int req_id;
15       int fdc = open(fifoc_name, O_RDONLY);!
16       read(fdc, &req_id, sizeof(int));
17       printf("Request %d completed\n", req_id);
18       close(fdc);
19
20       unlink(fifoc_name);
21       return 0;
22   }
```

   (a) Write the program `server` which, through the named pipe *fifo_server*, should receive search requests from clients (*i.e.*, from the program `search_prompt`) and execute them sequentially, using the program `SOGPT`. For each request, the server must assign it a unique identifier to be sent to the client as soon as it finishes.

   (b) Based on your solution to *(a)*, indicate whether it would work if the named pipe `fifoc_name` (line 15) was opened immediately after it was created (line 8). Justify your answer.

.... 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1

Question 11 Consider the following source code of a program and select the true statements:

```
int i=0;
while(i<10) {
    if (fork()==0) {
        op(i);
        i++;
        _exit(0);
    } else {
        i++;
    }
 }
```

A The operation `op(...)` executes a maximum of 5 times.

B `op(i)` only executes when `i` is even

C The operation `op(...)` executes a maximum of 10 times.

D The operation `op(...)` is never executed.

E *None of these answers are correct.*

Question 12 Consider the following source code of a program and select the true statements:

```
int main(int argc, char * argv[])
{
    for (int i = 0; i < 3; i++) {
        if (fork() == 0) {
            printf("process %d\n", i);
            func1();
            _exit(0);
        } else {
            wait(NULL);
            printf("process %d terminou\n", i);
         }
     }

     return 0;
 }
```

A The program creates processes that execute func1() sequentially.

B The program creates processes that execute func1() concurrently.

C The expected output is:
```
process 0
process 1
process 2
process 2 ended
process 1 ended
process 0 ended
```

D The expected output is:
```
process 0
process 0 ended
process 1
process 1 ended
process 2
process 2 ended
```

E *None of these answers are correct.*

Question 13 Consider a definition of struct **LicensePlate** and the following source code:

```
1   int main(int argc, char * argv[])
2   {
3       int fd = open("license_plates.bin", O_CREAT | O_RDWR, 0666);
4       LicensePlate l;
●         (...)
6       if (read(fd, &l, sizeof(LicensePlate)) < 0) {
7           perror("read");
8           exit(1);
9       }
10       write(1, l.plate, strlen(l.plate));
11       close(fd);
12       return 0;
13   }
```

Consider a binary file *license_plates.bin* with 50 sequential license plate records, numbered from 1 to 50. Select the options that complete the code above, *i.e.*, which replace the line of code 5 identified by (...), so that the variable `m` contains the twelfth record, after reading.

- [A] `lseek(fd, sizeof(LicensePlate) * 12, SEEK_SET);`
- [B] `lseek(fd, sizeof(LicensePlate), SEEK_SET);`
- [C] `lseek(fd, sizeof(LicensePlate) * 11, SEEK_SET);`
- [D] `lseek(fd, -sizeof(LicensePlate) * 11, SEEK_END);`
- [E] *None of these answers are correct.*

Question 14 Consider the following pseudo-code for a client program that sends messages to a server program.

Client:

```
int fd =  open("fifo", O_WRONLY);
write(fd, ...);
close(fd);
```

Server:

```
char buf[...];
int fd = open("fifo", O_RDONLY);
while(read(fd, buf,  ...) > 0){
    write(1,buf, ...);
}
close(fd);
```

Assume that communication between client and server is carried out via a named pipe, which was previously created with the name *"fifo"*. Also, assume that the server program is always executed before the client program. Indicate which of the following statements are true.

- [A] After a client has finished running, and if there are no more clients running, the server also ends its program and no longer serves any clients.
- [B] After a client has finished running, if other client programs are running and writing messages to the server, the server will also finish its program and will not serve these clients.
- [C] After a client has finished running, and if there are no other clients running, the server continues to run its program and serve other clients who may connect to it later.
- [D] After a client has finished running, if there are other client programs sending messages to the server, the server continues running its program and serving other clients until they have all finished.
- [E] *None of these answers are correct.*

Question 15 Consider the following source code and select the true statements:

```
1   int pfd[2][2];
2   pipe(pfd[0]);
3
4   if(fork()==0){
5       close(pfd[0][0]);
6       dup2(pfd[0][1], 1);
7       close(pfd[0][1]);
8       execlp("cat", "cat", "/etc/passwd", NULL);
9       _exit(0);
10   }
11
12   close(pfd[0][1]);
13   pipe(pfd[1]);
14
15   if(fork()==0){
16       close(pfd[1][0]);
17       dup2(pfd[0][0], 0);
18       close(pfd[0][0]);
19       dup2(pfd[1][1], 1);
20       close(pfd[1][1]);
21       execlp("sort", sort", NULL);
22       _exit(0);
23   }
24
●     (...)
26
27   if(fork()==0){
28       dup2(pfd[1][0], 0);
29       close(pfd[1][0];
30       execlp("wc", wc", -l, NULL);
31       _exit(0);
32   }
33
34   close(pfd[1][0]);
```

A The following code is missing (line 25)

```
1    close(pfd[1][1]);
2    close(pfd[1][0]);
```

B The following code is missing (line 25)

```
1    close(pfd[0][0]);
2    close(pfd[1][1]);
```

C The following code is missing (line 25)

```
1    close(pfd[0][0]);
2    close(pfd[1][0]);
```

D The program emulates the command `cat etc/passwd | sort | wc -l`

E *None of these answers are correct.*