Fundamentos de Sistemas de Operação - Exam 15/01/2008

No consulting of external support elements. **Duration: 3h00 hours**

Question 1. Explain the meaning of:

a) a "**file descriptor**" in Unix**.**
b) a "(hard) **link"** in Unix**.**
c) a "**process"** in Unix.

For each item, give a list of all the Unix operating system calls which manipulate the corresponding concept.

Question 2. Explain in detail what are the actions performed by the following Unix system calls, and also explain their effects upon the internal operating system data structures that you find more relevant. Also describe the meaning of all arguments of each call:

a) open
b) waitpid
c) fork
d) pipe

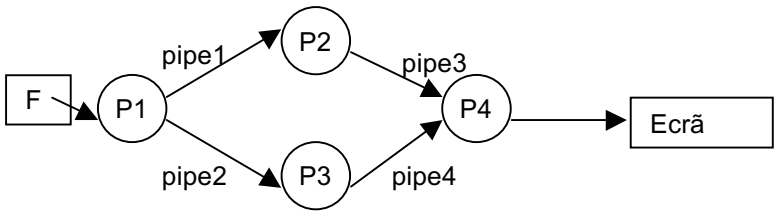
Question 3. Based on Unix system calls, write, in C, the code for the following function: **transfer_file(char *filename1, char *filename2)** that duplicates the contents of the file, with absolute pathname given by **filename1**, onto a new file with absolute pathname given by **filename2**, and then destroys the original file **filename1.**

Question 4.

a) Based on Unix system calls, write, in C, the code for the following function: **transfer_process()** that delegates the continuation of the execution of the program associated to the invoker process, into a newly created process, and destroying the original process. The new process must have exactly the same execution context and working environment as the original one and must continue the execution at the same point in the program where the original process was executing.

b) Based on Unix system calls, write, in C, the modified code for the following function: **transfer_process()** with a single modification comparing to the text in question a): once created, the new process must start its execution at the **first instruction** of the program associated to the invoker process (instead of at the same point in the program where the original process was executing, as was the case for question a)). Assume that the program in execution by the original process is contained in executable file "**/usr/bin/f**".

Question 5. In order to implement an Unix application to display videos using multiple processors, several phases were identified and allocated to distinct processes, which communicate using Unix *pipes*, in the following scheme: (where 'Ecrã" means the "Screen")



- Process P1 is responsible for reading from file F, successive blocks, 16Kbytes each, and distributing each one of them, in an alternate way, to P2 and P3, that is, one block to P2, next block to P3, next block to P2, ...

- Processes P2 and P3 are responsible for processing each block, as they receive them, and will be sending the results, also in the form of 16KB blocks, to process P4.

- Process P4 will be reading, also in an alternate way the received blocks from P2 or P3, and will be displaying them to the screen.

a) Write, in C, the code for the actions of the program, corresponding to the launching of all the above processes and all the required *pipes*. Assume that each process P1, P2, P3 and P4, once created, immediately starts executing, respectively, one of the functions **proc1(), proc2(), proc3()** and **proc4()** (assume for this question a) that those functions are already defined and they perform all the required initializations to connect processes to the *pipes*).

b) Write, in C, the code for function **proc1()** which is executed by **P1**. This function should read, from the file with name given in variable **char *F**, successive blocks, 16KB each, and then write each read block, in an alternate way, as explained above, into one of the pipes **pipe1** or **pipe2**.

c) Assume that processes **P2** and **P3** execute the same program, included in executable file "**/usr/bin/decod**", and are responsible for processing the blocks, as they will be arriving in their respecitive **standard input** channels, and then putting the resulting blocks (also 16kB each) in their corresponding **standard output** channels. Write, in C, the code for functions **proc2()** and **proc3()**, that must perform the required input/output channels redirections, before starting executing the program "**decod**" by each process P2 and P3.

Questão 6. Consider the same problem as in **Question 5** but assume now that communication between processes P2 and P3 with process P4 takes place **exclusively through a single and unique** *Unix message queue,* named **FM.** Rewrite the programs for functions **proc2()** and **proc3()** and also give the code for function **proc4()**, which must impose the alternate reading of messages coming from P2 and P3, and should rely exclusively on Unix system calls using the FM message queue for communication. Assume that the FM queue was previously created and also that each exchanged message has exactly the size of a 16KB block.

Questão 7. Consider a memory area that is shared by **N** concurrent processes, and has the maximum capacity of one 16KB block. One of the processes, named **P1,** produces one block at a time and writes the block in the shared memory area. Each one of the remaining processes **P2, ... PN** must read the contents of that block. Only after all processes (P2, ... PN) having read that block, then this will be considered as being read and then the shared memory area will be marked as avaliable (and free for further writing of another block). In order to achieve the above behavior, the following functions are defined:

- Process **P1** invokes function **putBlock(buf)** such that P1 blocks until the shared memory area is available, and then, P1 writes the contents of **buf** into the shared memory.
- Each one of the remaining processes **P2, ... PN** invokes function **readBlock(buf),** such that the process is blocked until there is a new block to read. Then the process gets a copy of the block into **buf**, and waits, blocked until all the remaining processes of the set **P2-PN** have also read that block

Write, in C, the code for functions **putBlock(buf) and readBlock(buf),** exclusively relying on communication by shared memory and on semaphores, as defined by Dijkstra.

Questão 8. Write the pseudo-code for a **monitor** in a concurrent programming language, such that