Homework 2: Due 09/29/2022 at 11:59 PM

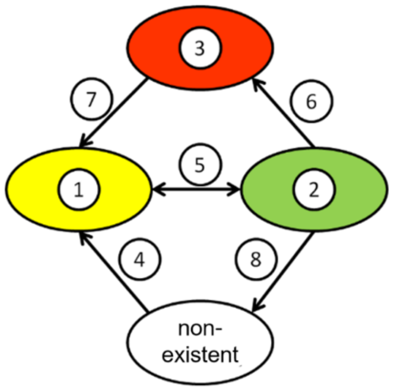
Your program (if requested) must compile with gcc and execute on snowball.cs.gsu.edu! Please see https://cscit.cs.gsu.edu/sp/guide/snowball for more details. You may use any IDE/ text editors you prefer, **but the source code (.c file) must be submitted via iCollege**.

1. The following steps in interrupt handling are out of place. Correctly arrange them in the order in which they are carried out to handle the interruption of a running user process by assigning a number from 1 through 9 to indicate the precedence order for the following steps (**9 points**):

   For example, 1 would indicate the first step in the process, 2 would indicate the second in the process and so on.

Step	**Step number from 1 to 9**
Handle further interrupts (if applicable).	
The appropriate interrupt service routine is identified and started based on the source of the interrupt.	
The interrupt service routine is terminated with a special command (return from interrupt, RTI).	
Switch to kernel mode.	
The state of the currently running process is saved.	
Switch to user mode and continue the execution of the interrupted process.	
The established blocking of further interrupts (if applicable) is released again.	
If necessary, the handling of further interrupt requests is stopped (blocked).	
The state of the interrupted process is restored.	

2. a) **Process states**: In this task, we work with the process state model described in the lecture. For reasons of clarity, the state "non-existent" has been represented somewhat more compactly than in the lecture (this state represents both the "new" and the "terminated" states). Likewise, state transition 5 subsumes two activities (the job is either scheduled for execution or an interrupt is raised). Assign the following terms to the respective numbered elements in the process state model (**8 points**).



Term	**Numbered element**
scheduler dispatch or interrupt	
ready	
wait for I/O or event	
admitted	
waiting	
I/O or event completion	
exit	
running	

b) Explain why there is no direct transition from the state "waiting" to the "running" state. (**3 points**)

3. **a)** Write a C program to Implement a system of three processes which read and write numbers to a file. Each of the three processes P1, P2, and P3 must obtain an integer from the file (these instructions must be executed 200 times). The file only holds one integer at any given time. Given a file F, containing a single integer N, each process must perform the following steps (**25 points**):

1. Fork two processes

For 200 times:

2. Open F
3. Read the integer N from the file
4. Close F
5. Output N and the process' PID (On the screen)
6. Increment N by 1
7. Open F
8. Write N to F (**overwriting** the current value in F)
9. Close F

b) Briefly describe why the processes P1, P2, and P3 obtain/read duplicates of numbers (why does a particular integer x appear in the output of more than one process)? (**3 points**)

c) Suggest a solution (**you do not need to implement it**) to guarantee that no duplicate numbers are ever obtained by the processes. In other words, each time the file is read by any process, that process reads a distinct integer. (**2 points**)

Please use pseudocode or C instructions to describe your solution. Any standard function used must use the correct prototype.

NOTE: Programs must compile and execute on the Snowball Server. Submit all .c code files along with the word document or pdf with your answers to questions 1 & 2 onto iCollege.