COMP 530 Introduction to Operating Systems

Fall 2017  
Kevin Jeffay

Worksheet 4, September 13

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Your Name: |  | You worked with: |  | +1/blank/-1: |  |
|  | Aaron Zhang |  | Nillas Erik Peissan |  |  |  |
|  |  |  | Harrison Krome |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. In the implementation of a context switch, the actual “switch” takes place in the *dispatch* function. In this function, the value of the variable *runningProcess* is saved and then restored in the assignment statements:

*me := runningProcess*

*…*

*runningProcess := me*

*a*) Given that there are no other statements involving the variables *runningProcess* and *me* in *dispatch* other than these two statements, one could claim that these statements have no effect and could be deleted. Explain why this is not the case (*i.e.*, explain why these statements are needed). In particular, just before the execution of the statement

*runningProcess := me*

in the procedure *dispatch*, what is the value of the variable *runningProcess*?

These statements are needed to save the current running process onto memory before loading the next process, the running process needs to be assigned to the new me which is received from the next process.

*b*) The implementation of a context switch relies on the fact that the variable “me” in the procedure *dispatch* that stores the process ID of the running process is a local variable. Why is it essential that *me* be a local variable and not a global variable?

If me was a global variable the running process will never change and there would only be one process in the loop

2. Consider the run-time stacks used by user processes to support procedural programming. Specifically, consider both the user-level stack (the stack of activation records for user-level functions called by the process), and the system stack associated with the user process (the stack of activation records for system level functions called [directly or indirectly] by the process).

For both the user stack and the system stack, characterize the top of the stack (the activation record on the top of each stack) for each process in the *ready* and *waiting* states (in the three state process state transition diagram described in lecture).

By in large, everyone will have dispatch at the top of the stack.

3. Explain the concept of a “forced system call” in the context of implementing a context switch in a timesharing system? What is the “forced system call”? Speculate on why we’re “forcing” a system call on a user process (as opposed to some other means of causing a context switch).