CS 452 (Spring 23): Operating Systems

Phase 1 - Process Control - Milestone 1a

milestone 1a due at 5pm, Thu 2 Feb 2023
milestone 1b due at 5pm, Thu 9 Feb 2023

NOTE NOTE NOTE
Unlike all of the other Phases, Phase 1 is broken into two "milestones," with different due dates. This spec describes **milestone 1a** (the first one), but it only describes the **changes** from milestone 1b. Make sure to read the main spec, in addition to this file!
NOTE NOTE NOTE

1 Milestone 1a vs. Milestone 1b

Milestone 1a only requires that you implement **part** of Phase 1. Several functions and features can be skipped entirely; a few you must implement, but with reduced features; and there is one **temporary** function that you will implement for Milestone 1a, but then must remove for Milestone 1b.

1.1 Functions and Features You Can Skip

- `blockMe()`, `unblockProc()`

  In Milestone 1a, processes never block.

- `zap()`, `isZapped()`

- The dispatcher (your code that automatically decides what process to run next). Instead of an automatic dispatcher, the testcases will choose when to switch from one process to another.

- `readCurStartTime()`, `currentTime()`, `readtime()`, `timeSlice()`

- Any code to handle the clock interrupt

1.2 Simplified Functions

- `startProcesses()` does not call your dispatcher (since you will not have it, yet). Instead, it uses `USLOSS_ContextSwitch()` to manually switch to the `init` process.

- Once `init` creates the `sentinel` and `testcase_main` processes, it should call `USLOSS_ContextSwitch()`[1] to manually switch to the `testcase_main` process.

[1] If you prefer, `TEMP_switchTo()` is also OK. See below for details.

- If the testcase's main function ever returns, print out message to the console and then halt the simulation. However, this is **normal termination** (not an error), so call `USLOSS_Halt(0)` instead of stopping with an error code.

  If any user function (other than the testcase main) returns, print an error and terminate the simulation (with a nonzero error code), since you cannot possibly know what process would need to run next.

- `fork1()` does not call the dispatcher after it creates the new process. Instead, the testcase will be responsible for chosing when to switch to another process.

- `join()` will never block. If you cannot find any already-dead children to report status about, print out an error and terminate the simulation immediately.

- `quit(int status, int switchToPid)` has **two parameters** instead of one. The first parameter works as in Milestone 1b. The second tells you which process to switch to, after you have cleaned up the process. (In 1b your dispatcher will make the choice.)

- `quit()` never needs to wake up a blocked parent process, or zappers, since the other processes cannot block in Milestone 1a.

1.3 Temporary Function

Instead of a dispatcher, in Milestone 1a, testcases will choose exactly when to switch from one process to another. To tell you when to switch, they will call `TEMP_switchTo(int newpid)`. Switch to the process indicated.

These context switches, although they are triggered manually instead of automatically, must call `USLOSS_ContextSwitch()` and save the old process state, so that it is possible to switch back to them later!

2 Turning in Your Solution

You must turn in your code using GradeScope. While the autograder cannot actually assign you a grade (because we don't expect you to match the "correct" output with perfect accuracy), it will allow you to compare your output, to the expected output. Use this report to see if you are missing features, crashing when you should be running, etc.