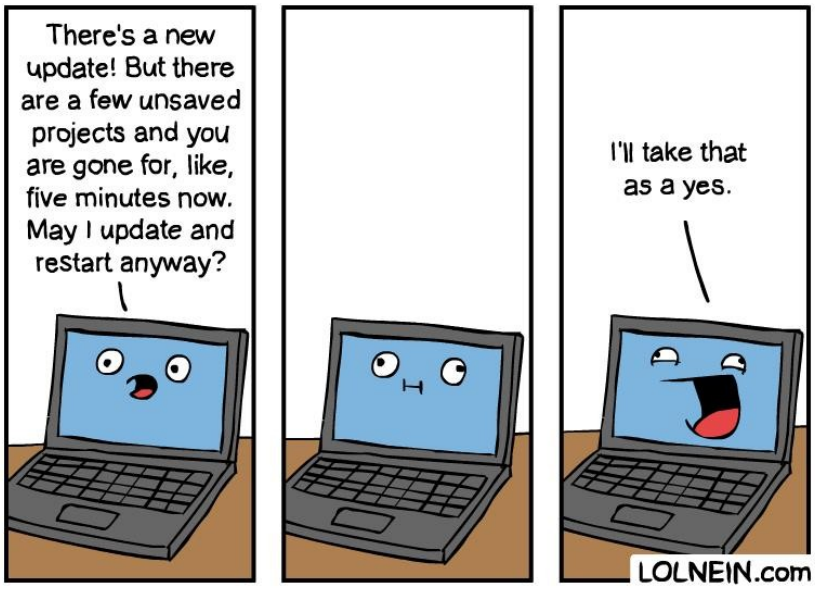
CS 370 – Project #4, Part B

Purpose: Become familiar with operating system process scheduling through **xv6**
Points: 300 (200 code / 100 final report)

Introduction:

Any operating system is likely to run with more processes than the computer has CPUs, so a plan is needed to time-share the CPUs among the processes. Ideally the sharing would be transparent to user processes. A common approach is to provide each process with the illusion that it has its own virtual CPU by *multiplexing* the processes onto the hardware CPUs.



Resources:

The following resources provide more in depth information regarding **xv6**. They include the **xv6** reference book, tools for installing, and guidance to better understand **xv6**.

1. xv6 Reference Book: https://pdos.csail.mit.edu/6.828/2020/xv6/book-riscv-rev1.pdf
2. Lab Tools Guide: https://pdos.csail.mit.edu/6.828/2020/tools.html
3. Lab Guidance: https://pdos.csail.mit.edu/6.828/2020/labs/guidance.html

Project:

Complete the following steps.

Become familiar with **xv6** scheduling

- Implement **setbkg()** system call.
- Implement **psmb** system program (process status with memory, background).
- Develop a user-level scheduler test program.
- Implement foreground/background based round-robin scheduling in **xv6**.
- Prepare a final summary report.

The following sections provide additional detail about the specific technical requirements.

Submission

When complete, submit:

- A copy of the zipped **xv6** folder via the class web page.
- *Note*, late submissions will not be accepted.

Project Setup

Project #4 will be done entirely in **xv6**. This part of the project is an extension of Part A.

Set Background System Call

We will add a new system call, **setbkg()**, which will allow a process to tell the OS scheduler that is is a background process and is of less importance than an foreground job (and thus is expected to take longer to complete). In Part A, a **`priority`** (**`uint64`**) attribute to the **`PCB`** of each process was added and priority **2** was established as the base or default priority. This is considered the foreground. Decide on a priority for background processes (something other than 2). It may be higher or lower than 2 as there will only be two possibilities (foreground and background). The **setbkg()** system call should change the current priority to the new background priority and **yield()**. You can review other system calls for the specific usage of **yield()**. The **setbkg()** system call should be implemented in the **`kernel/sysprocs.c`** with the usual updates to **`user/user.h`**, **`user/usys.pl`**, **`kernel/syscall.h`**, and **`kernel/syscall.c`** (as per the previous projects).

Implement **psmb** system program (process status with memory, background). This is a clone of the **psm** system program that uses the **setbkg()** system call as a test.

An example of the output is shown below.

```
$ psmb
Process Status BK (ed)
pid     state     prior memory  process name
-----------------------------------------------
1       SLEEPING  0     12288   init
2       SLEEPING  2     16384   sh
4       RUNNING   ?     12288   psmb
$
```

The "?" is the priority value you chose for a background process.

Testing

Create a user-level program with a function, **primeCount()**, that computes the count of prime numbers between 2 and some **LIMIT** (initially set to 100,000). Test this function to ensure it works correctly. Most algorithms stop at square root of N, but since **xv6** does not support the square root function, you will need to go to N/2.

When working, create a user-level program, **`schTest.c`**, that fork's **MAX_PROCESS** (initially set to 6) children all doing the same CPU-bound task (computing a count of prime numbers between 2 and **LIMIT**), but with different priorities. Specifically, the program should loop to fork child which will call the **primeCount()** function, display the final result, and terminate. When displaying the final result, the output should indicate **fg** (foreground) or bk (background). Refer to he sample output for formatting examples. For debugging purposes, the index should also be displayed. As the children are created, every other new child process should set itself as background by calling the **setbkg()** system service. Ensure that the first new child (0^{th}, 2^{nd}, 4^{th}, etc.) is set as background and the next left as foreground (1^{st}, 3^{rd}, 5^{th}, etc.).

Before any scheduler changes are a typical output is shown below.

```
$
$ schTest
Scheduler Tests.
Starting children...
 Prime count (bk): 9592 2
 Prime count (fg): 9592 1
 Prime count (bk): 9592 0
 Prime count (fg): 9592 3
 Prime count (fg): 9592 5
 Prime count (bk): 9592 4

Scheduler tests done.
$
```

Note, how the job mix (foreground and background) are intermixed (and likely different on different executions). You may review **`forktest.c`** for some examples of using the **fork()** system service call.

Priority Based Round-Robin Scheduler

For this part of the project, we will modify the **xv6** scheduler from strict round-robin to a foreground background round-robin scheduler. This will involved scheduling all foreground jobs in a round-robin manner before the background jobs (which are then scheduled in a round-robin manner). Thus, a process with a foreground priority should get the CPU more than a process with background priority.

Now that the priority attribute has been assigned to each process, we will modify the **`scheduler`** function (**`kernel/proc.c`**). Scheduler will first have to find a **`RUNNABLE`** process and select if this process identified as foreground. This will require adding an additional inner loop. The first process found with the foreground priority will then be scheduled. If no foreground processes are found, the first background process found should be scheduled.

It is suggested that you comment out the original round-robin scheduler code before adding your new version of the scheduler. This will allow you to switch back for testing in preparation for the final report.

Before any scheduler changes are a typical output is shown below.

```
$
$ schTest
Scheduler Tests.
Starting children...
 Prime count (fg): 9592 5
 Prime count (fg): 9592 3
 Prime count (fg): 9592 1
 Prime count (bk): 9592 4
 Prime count (bk): 9592 2
 Prime count (bk): 9592 0

Scheduler tests done.
$
```

Note the job mix (foreground completed before background).

Final Report

Prepare a final report including the following sections.

- Summary
  - Explanation of changes made.
  - Rational for priority value of background process.
- Results
  - Example output of scheduler test program before and after the scheduler changes.
  - Execute the scheduler test program a number of time and discuss any output anomalies.
- Starvation
  - Discuss any potential starvation issues.
  - Provide a suggestion for addressing the starvation issue.

The report should include appropriate titles and section headers. Additionally, spelling and grammar will be part of the final report score.