Lab 1: OS structure and Scheduler

Due: Friday April 30

Preliminaries: You may work in groups of 2 or individually. Please attend the class introducing xv6 on Friday, and look for instructions to download xv6 and to understand its structure on the class website (start here: https://www.cs.ucr.edu/~nael/cs202/labs.html). You can work on your assignment on your own machine or on sledge.cs.ucr.edu. If you do not have a cs.ucr.edu account, please let me know so that we can ask systems to make you one.

I may be posting additional hints depending on how I see the class progressing towards the assignment (let me know how things are going!). Please start early – OS code is difficult to run and debug, and it will take a while for you to understand xv6. I expect this assignment to take 20-40 hours (perhaps even more if you are unlucky or inexperienced with C or systems programming). I guess these will be roughly broken into 40% understanding xv6 and what you need to do, 20% implementation and 40% debugging and evaluation. Please start early or it is highly likely that you will not be able to finish on time. Ask me for help if you are struggling.

Part A: System call and xv6 basic structure

The goals of this part of the assignment is to learn how to (1) add a system call; and (2) to get exposure into how xv6 supports processes, memory and system calls by tracking information about them. Add a new system call info(int param) that takes one integer parameter with value 1, 2 or 3. Depending on the value, it returns:

(1) A count of the processes in the system; Hint: you should be able to come up with a strategy by finding the data structure that keeps track of the list of processes in the system.

(2) A count of the total number of system calls that the current process has made so far; Hint: this is a value that you can keep track of for each process, incrementing count for that process with every system call.

(3) The number of memory pages the current process is using; Hint: this can be tricky if you try to do it without understanding the memory image allocation of a process. Look at the sz variable in the process control block and see how that value is set.

Part B: Scheduling

The goal of this part of the project is to have you investigate modifications to the scheduling by implementing lottery and stride scheduling which we will discuss in class. In this part, you will implement lottery scheduler and stride scheduler. Implement only basic operation (no ticket transfers, compensation tickets, etc...). It is up to you to design the interface to assign the number of tickets, but it should be sufficient to illustrate that the schedulers work correctly.

To get started, look at the file "proc.c". In there you will find a simple round robin CPU scheduler implemented. You should change the logic there to use lottery and stride scheduling. After you complete the schedulers, you can use the following program to demo your work for each one separately. Feel free to study other configurations to illustrate the difference between the schedulers.

Run the following program with for 3 different ticket values:

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc, char *argv[])
{
    FUNCTION_SETS_NUMBER_OF_TICKETS(30);    // write your own function here
    int i,k;
    const int loop=43000;
    for(i=0;i<loop;i++) {
        asm(nop);      //in order to prevent the compiler from optimizing the for loop
        for(k=0;k<loop;k++) {
            asm(nop);
        }
    }
    exit();
}
```

Output should look like this:

for example: The program prog1.c has 30 tickets, add prog2.c with 20 tickets and prog3.c with 10 tickets. Run all of them simultaneously:

Input

```
$ prog1&;prog2&;prog3
```

Assuming that you run 3 test programs, each with a different number of tickets, when each of these programs finishes execution, your system call should print the number of times the processes was scheduled to run (number of ticks).

Output: (not necessarily in the same order)

```
Ticks value of each program

prog1 : xxx

prog2 : xxx

prog3 : xxx
```

Use the number of ticks to calculate the allocated ratio per process : (number of ticks per program)/(total number of ticks by all 3 programs)

Approximately prog1 ratio will be 1/2 , prog2 ratio will be 1/3 and prog3 will be 1/6

Compare your implementations of lottery scheduler and stride scheduler. Consult Section 5 of the stride scheduling paper on how to compare and evaluate these two schedulers. Try to reproduce Figure 8 and Figure 9 for your implementations. Do your results support the conclusions regarding the two schedulers?

What to submit:

You need to submit the following:

(1) A detailed explanation what changes you have made. You can use screenshots or list relevant pieces of code to show your work and results

(2) A tarball/zipball of your xv6 directory after you make clean.

Grade breakdown:

Grading will be by interview. Both group members are expected to be able to answer questions about any aspect of the implementation.

- Correct implementation and demo of info() system call: 20%
  - count of the processes in the system: 10%
  - report the number of system calls this program has invoked: 10%
  - report the number of memory pages the current process is using: 10%
- Working lottery scheduler (with clear explanation): 20 %
- Working stride scheduler (with clear explanation): 20%
- Comparison figures and clear explanation how to produce them: 10%
- Bonus (up to 10%): implement additional features described in the paper for lottery scheduling and demonstrate them

Total: 100% + (10% bonus)