Zachery Creech

Dr. Beck

COSC361

17 March 2021

Project 1

Modifications to the following files:

Makefile:

- Addition of 2 new terminal commands, ‘test’ for testing and ‘getpinfo’ that prints all currently running processes, their total time running (in ticks), and each one’s ticket total
  - _getpinfo\
    _test\
- Modification to use only 1 CPU
  - CPUS := 1

test.c:

- A simple program that can be ran from the command line by typing “test” that spawns 3 child processes with fork(). Each process is allocated a different number of tickets (10, 20, 30) to show that those processes will get 1x, 2x, and 3x the CPU time respectively. Parent program prints process info with getpinfo every ~10 seconds.

proc.c:

- Addition of global variable ‘total_tickets’ to track total pool of tickets for all processes
  - int total_tickets = 0;
- Additions to allocproc() to setup default tickets and ticks and update total_tickets accordingly
  - p->tickets = 1;
    p->ticks = 0;
    total_tickets++;

- Additions to fork() to set child's tickets equal to parent's and update total_tickets accordingly
  - total_tickets--;
    total_tickets += curproc->tickets;
    np->tickets = curproc->tickets
    - Decrement total_tickets to remove ticket that was added to the pool when the child was created by allocproc()
- Addition to exit() to update total_tickets when a process exits
  - total_tickets -= curproc->tickets;
- Additions to scheduler() to implement lottery scheduling
  - Before the main for loop, seed the random number generator
    - srand(29051453);
  - Within the loop before choosing the next process, find the winning number and set the accumulated "minimum ticket" to 0
    - cur_min = 0;
      winner = rand() % (total_tickets + 1);
  - Modify the if statement that chooses the next process to run with checks for the winning ticket
    - if(p->state != RUNNABLE || winner > p->tickets + cur_min || winner < cur_min)
    - Body of if statement: cur_min += p->tickets; continue;
  - Before switching to the winning process, save the current number of ticks
    - start_time = ticks;
  - After the process ends, compute how long it was running and update the process's ticks field
    - p->ticks += ticks – start_time;
  - Add a "break" at the end of the inner for loop to make sure lottery restarts every time a new process needs to be chosen
- Addition of new function settickets() to set a process's tickets
  - See proc.c at the bottom
- Addition of new function getpinfo() to fill pstat structure and print all currently running processes,

their total time running (in ticks), and each one's ticket total
    - See proc.c at the bottom

sysproc.c:

- Addition of two functions sys_settickets(void) and sys_getpinfo(void) to do trap security checking
    - See sysproc.c at the top

syscall.c:

- Addition of external sys_ function declarations for settickets() and getpinfo()
    - extern int sys_settickets(void);
      extern int sys_getpinfo(void);
- Addition of sys_ function tags in static syscalls[]
    - [SYS_settickets] sys_settickets,
      [SYS_getpinfo] sys_getpinfo,

getpinfo.c:

- Program that calls getpinfo from the command line
    - See getpinfo.c

rand.h:

- Random number generator from https://rosettacode.org/wiki/Linear_congruential_generator#C
    - See rand.h

defs.h:

- Addition of declaration for pstat
    - struct pstat;
- Addition of function signatures for settickets() and getpinfo()
    - int settickets(int);
      int getpinfo(struct pstat*);

proc.h:

- Modification of proc struct to add tickets field and ticks field
    - int tickets;
      int ticks;

user.h:

- Addition of declaration for pstat
  - struct pstat;
- Addition of function signatures for settickets() and getpinfo()
  - int settickets(int);
    int getpinfo(struct pstat*);

syscall.h:

- Define trap numbers for SYS_settickets and SYS_getpinfo
  - #define SYS_settickets 22
    #define SYS_getpinfo 23

pstat.h:

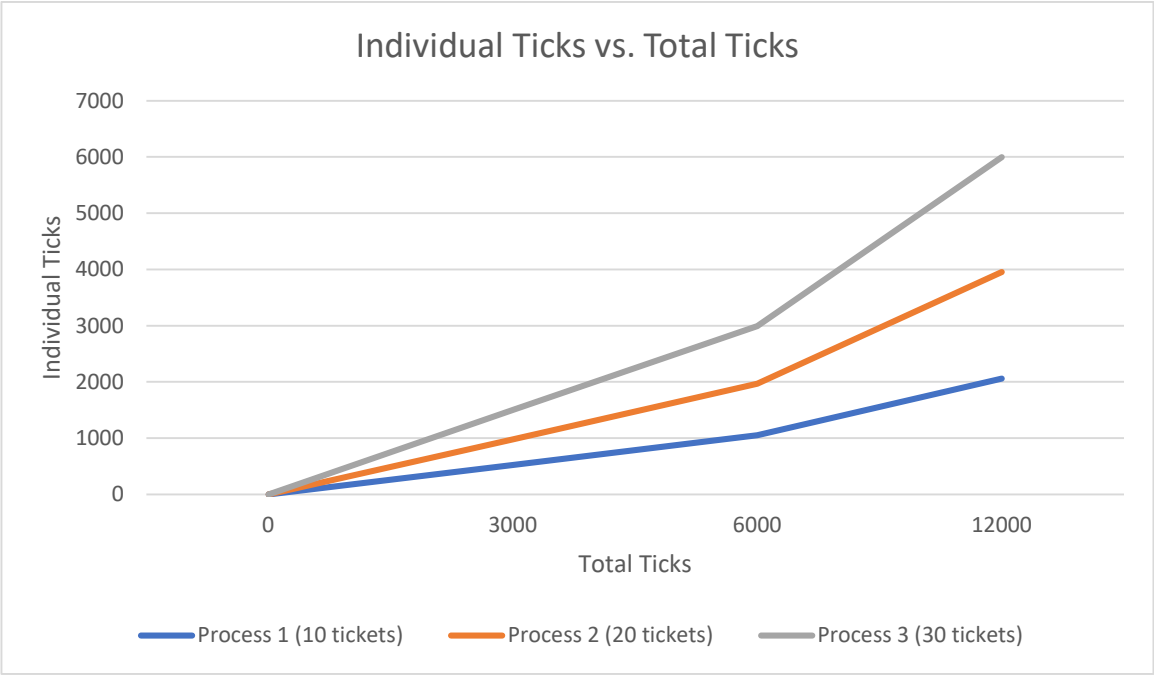
- Header file for pstat struct
  - See pstat.h

usys.S:

- Addition of settickets() and getpinfo() to SYSCALL script list
  - SYSCALL(settickets)
    SYSCALL(getpinfo)

The biggest “trick” to getting this to work was figuring out how the kernel vets arguments that are passed from user space. Creating a sys_ function that pairs with each system call that ends up calling the non-_sys version after using argint() and argptr() to verify all data is correct seemed complicated at first, but it wasn’t too difficult. Getting everything to compile together with the correct #include’s was the more difficult part.

It was also important to realize that the ptable had to be locked anytime a process struct was modified, such as when a process’s tickets are set with setticket() or when traversing the ptable during getpinfo(). The lock had to be obtained within proc.c as well, since the ptable was not defined in other files (such as in sysproc.c. I thought I could do the locking before calling the non-_sys version, but it had to be done from within the non-_sys version in proc.c).


Individual Ticks vs. Total Ticks
7000
6000
5000
4000
3000
2000
1000
0
Individual Ticks
0
3000
6000
12000
Total Ticks
Process 1 (10 tickets)
Process 2 (20 tickets)
Process 3 (30 tickets)