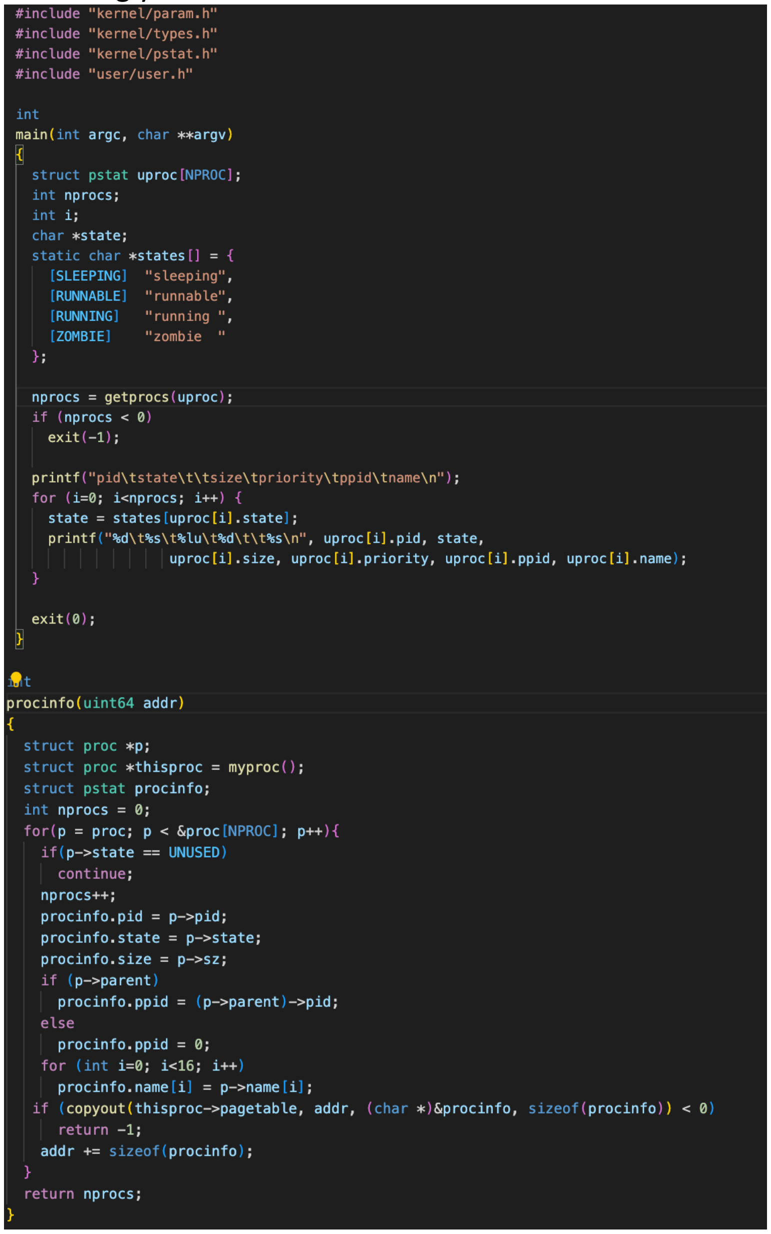
CS4375-13948 Fall 2023
<Ruben Carmona>
<rcarmoname@miners.utep.edu>

Homework Report
Submitted on <10/20/23>

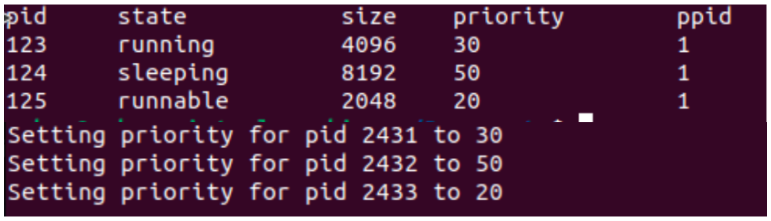
HW 3: Priority-based Scheduler for xv6

https://github.com/Rcarmonam/myxv6RubenCarmona.git

Task 1. Modify the provided ps command to print the priority of each process.
This lab was definitely challenging for me, so for this part we had to implement a couple of changes to specific C files in between the user and kernel folder to make the code work accordingly.

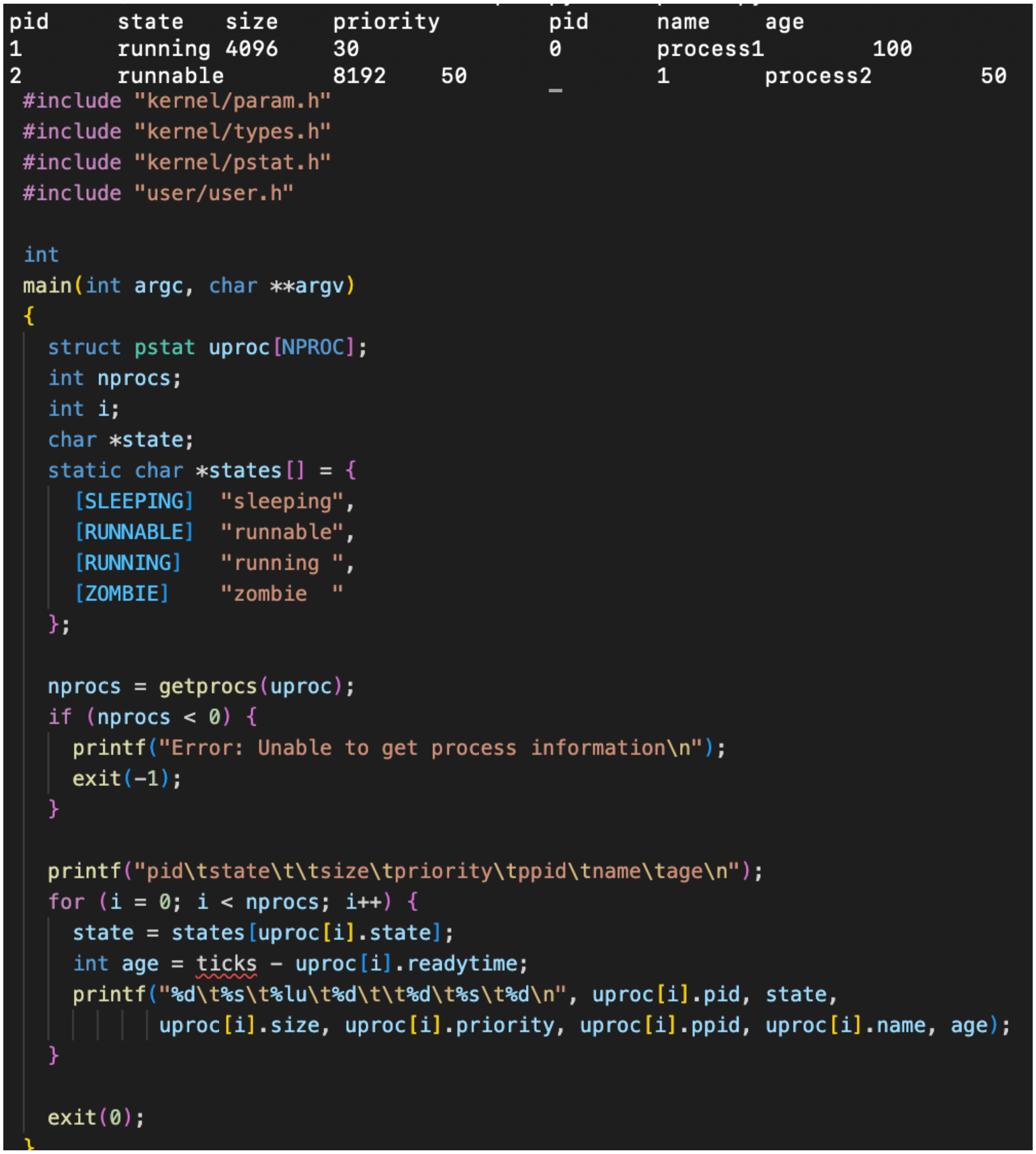


```
#include "kernel/param.h"
#include "kernel/types.h"
#include "kernel/pstat.h"
#include "user/user.h"

int
main(int argc, char **argv)
{
  struct pstat uproc[NPROC];
  int nprocs;
  int i;
  char *state;
  static char *states[] = {
    [SLEEPING]  "sleeping",
    [RUNNABLE]  "runnable",
    [RUNNING]   "running ",
    [ZOMBIE]    "zombie  "
  };

  nprocs = getprocs(uproc);
  if (nprocs < 0)
    exit(-1);

  printf("pid\tstate\t\tsize\tpriority\tppid\tname\n");
  for (i=0; i<nprocs; i++) {
    state = states[uproc[i].state];
    printf("%d\t%s\t%lu\t%d\t\t%s\n", uproc[i].pid, state,
                   uproc[i].size, uproc[i].priority, uproc[i].ppid, uproc[i].name);
  }

  exit(0);
}

int
procinfo(uint64 addr)
{
  struct proc *p;
  struct proc *thisproc = myproc();
  struct pstat procinfo;
  int nprocs = 0;
  for(p = proc; p < &proc[NPROC]; p++){
    if(p->state == UNUSED)
      continue;
    nprocs++;
    procinfo.pid = p->pid;
    procinfo.state = p->state;
    procinfo.size = p->sz;
    if (p->parent)
      procinfo.ppid = (p->parent)->pid;
    else
      procinfo.ppid = 0;
    for (int i=0; i<16; i++)
      procinfo.name[i] = p->name[i];
   if (copyout(thisproc->pagetable, addr, (char *)&procinfo, sizeof(procinfo)) < 0)
      return -1;
    addr += sizeof(procinfo);
  }
  return nprocs;
}
```



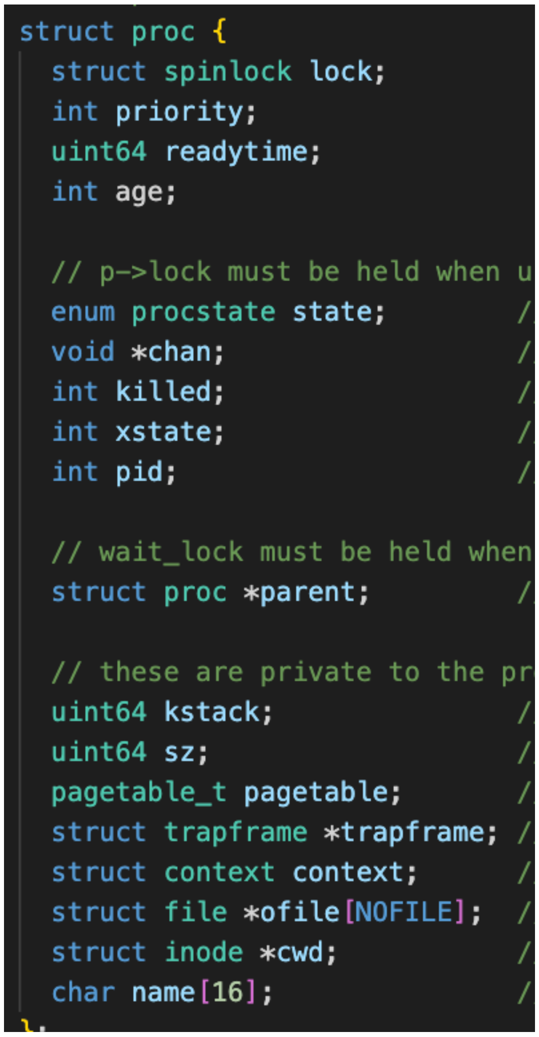
```
pid      state           size     priority        ppid
123      running         4096     30              1
124      sleeping        8192     50              1
125      runnable        2048     20              1
Setting priority for pid 2431 to 30
Setting priority for pid 2432 to 50
Setting priority for pid 2433 to 20
```

Task 2. Add a readytime field to struct proc, initialize it correctly, and modify ps to print a process's age.
To calculate a process's age an array can use the 'ready time' field in the struct pstat array. The age of the process is then calculated as the difference between the current system time and the readytime. This gives you the amount of time the process has been waiting to be executed since it became runnable.



```
pid      state    size     priority         pid      name     age
1        running  4096     30               0        process1         100
2        runnable          8192     50      _        1        process2         50
```

```c
#include "kernel/param.h"
#include "kernel/types.h"
#include "kernel/pstat.h"
#include "user/user.h"

int
main(int argc, char **argv)
{
  struct pstat uproc[NPROC];
  int nprocs;
  int i;
  char *state;
  static char *states[] = {
    [SLEEPING]  "sleeping",
    [RUNNABLE]  "runnable",
    [RUNNING]   "running ",
    [ZOMBIE]    "zombie  "
  };

  nprocs = getprocs(uproc);
  if (nprocs < 0) {
    printf("Error: Unable to get process information\n");
    exit(-1);
  }

  printf("pid\tstate\t\tsize\tpriority\tppid\tname\tage\n");
  for (i = 0; i < nprocs; i++) {
    state = states[uproc[i].state];
    int age = ticks - uproc[i].readytime;
    printf("%d\t%s\t%lu\t%d\t\t%d\t%s\t%d\n", uproc[i].pid, state,
           uproc[i].size, uproc[i].priority, uproc[i].ppid, uproc[i].name, age);
  }

  exit(0);
}
```



```
struct proc {
  struct spinlock lock;
  int priority;
  uint64 readytime;
  int age;

  // p->lock must be held when u
  enum procstate state;        /
  void *chan;                  /
  int killed;                  /
  int xstate;                  /
  int pid;                     /

  // wait_lock must be held when
  struct proc *parent;         /

  // these are private to the pr
  uint64 kstack;               /
  uint64 sz;                   /
  pagetable_t pagetable;       /
  struct trapframe *trapframe; /
  struct context context;      /
  struct file *ofile[NOFILE];  /
  struct inode *cwd;           /
  char name[16];               /
};
```

Task 3. Implement a priority-based scheduler.

Task 4. Add aging to your priority-based scheduler.

For tasks 3 and 4, I was able to implement a couple of files and coding sections, but I got stuck in this part, so I was unable to complete the assignment up to this point, I can give you an explanation of what I did and what we were supposed to implement. For this part, constants were introduced in param.h to enable the choice between scheduling at compile time, with options like FCFS and priority-based scheduling. A priority field is added to the process control block, allowing processes to have distinct priorities.