Assignment #5
proc virtual file structure

/proc is very special in that it is also a virtual filesystem. It's sometimes referred to as a process information pseudo-file system. It doesn't contain 'real' files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). For this reason it can be regarded as a control and information centre for the kernel.

The purpose and contents of each of these files is explained below:

- /proc/PID/cmdline
  Command line arguments.
- /proc/PID/cpu
  Current and last cpu in which it was executed.
- /proc/PID/cwd
  Link to the current working directory.
- /proc/PID/environ
  Values of environment variables.
- /proc/PID/exe
  Link to the executable of this process.
- /proc/PID/fd
  Directory, which contains all file descriptors.
- /proc/PID/maps
  Memory maps to executables and library files.
- /proc/PID/mem
  Memory held by this process.
- /proc/PID/root
  Link to the root directory of this process.
- /proc/PID/stat
  Process status.
- /proc/PID/statm
  Process memory status information.
- /proc/PID/status
  Process status in human readable form.

One example:

```
# cat status
```

```
Name: sshd

State: S (sleeping)
Tgid: 439
Pid: 439
PPid: 1
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
FDSize: 32
Groups:
VmSize:     2788 kB
```

```
VmLck:         0 kB
VmRSS:      1280 kB
VmData:      252 kB
VmStk:        16 kB
VmExe:       268 kB
VmLib:      2132 kB
SigPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 8000000000001000
SigCgt: 0000000000014005
CapInh: 0000000000000000
CapPrm: 00000000fffffeff
CapEff: 00000000fffffeff
```

Assignment #1: First complete previous lab exercise (hint use file to store information using open(), read(), write() system call).

Assignment #2: You need to create virtual file system in directory /proc. Next, create directory with name “PID” under /proc directory (Hint: edit scheduler). Finally, you need to implement any three files (your choice) on run time under directory /proc/PID/”filename”.

Hint: Printing Out Running Process ID

First, we add in the scheduler() function, which is in proc.c a line of code (highlighted in blue) to print out the name and pid of the currently running process:

```
void scheduler(void)
{
struct proc *p;
}
for(;;){
......
p->state = RUNNING;
cprintf("Process %s with pid %d running\n", p->name, p->pid);
swtch(&cpu->scheduler, p->context);
switchkvm();
......
}
```

Hint: Add more features to your scheduler so that you can implement time-stamp of processes

Add in the file proc.h the struct proc> the timestamps and the priority:

```
struct proc {
......
int killed;                    //If non-zero, have been killed
struct file *ofile[NOFILE];          //Open files
```

```
struct inode *cwd;      //Current directory
char name[16];          //Process name (debugging)
// add timestamps and others
uint createTime;        // process creation time
int sleepTime;          // process sleeping time
int readyTime;          // process ready (runnable) time
int runTime;            // process running time
int priority;           // process priority
int tickcounter;
char dum[8];
};
```

Add to the function allocproc() in proc.c the timestamp statements (highlighted in blue):

```
static struct proc*
allocproc(void)
{
.....
release(&ptable.lock);
return 0;
found:
p->state = EMBRYO;
p->pid = nextpid++;
p->createTime = ticks;
p->readyTime = 0;
p->runTime = 0;
p->sleepTime = 0;
}
```

At this point, you should recompile the OS to make sure there is no typo error. Then modify the cprintf statement added in scheduler()

```
cprintf("Process %s with pid %d running with createTime %d\n", p->name, p->pid, p->createTime);
```