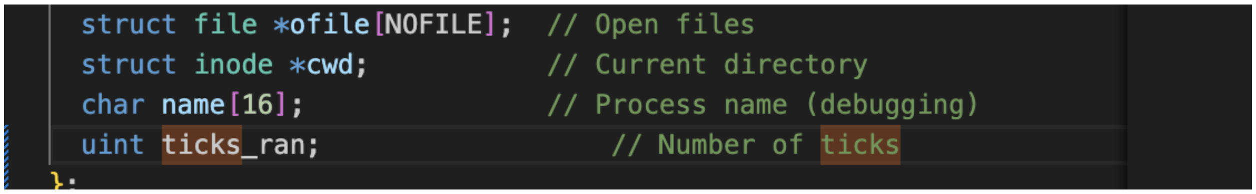
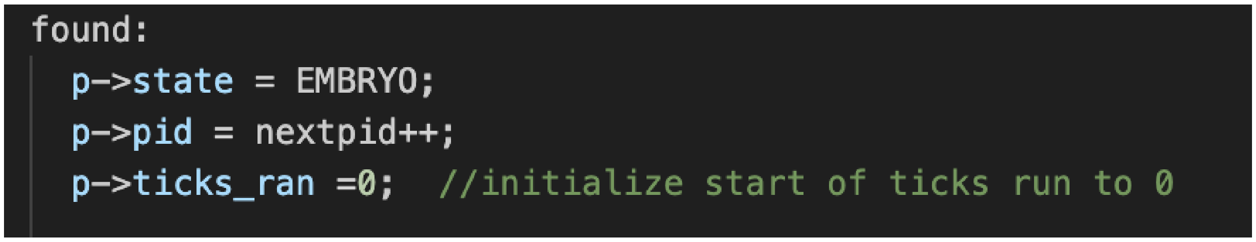
Initialize ticks_ran in proc.h



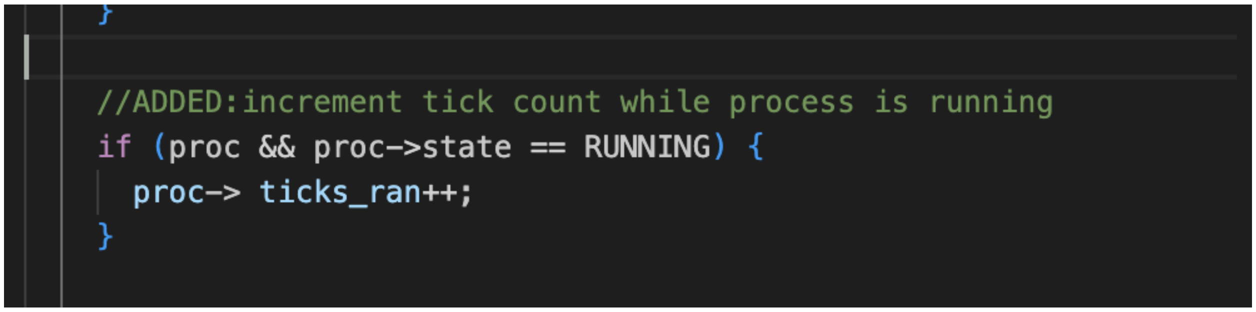
```
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  uint ticks_ran;                  // Number of ticks
};
```

Initialize ticks_ran as 0 in proc.c



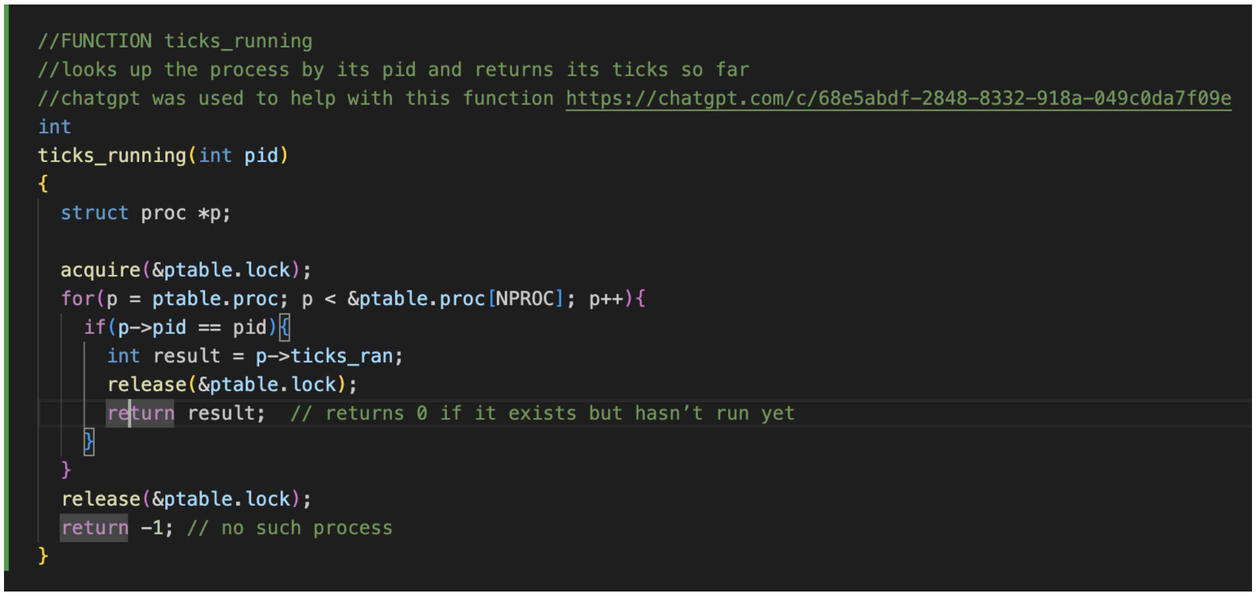
```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->ticks_ran =0;  //initialize start of ticks run to 0
```

Incrementing ticks



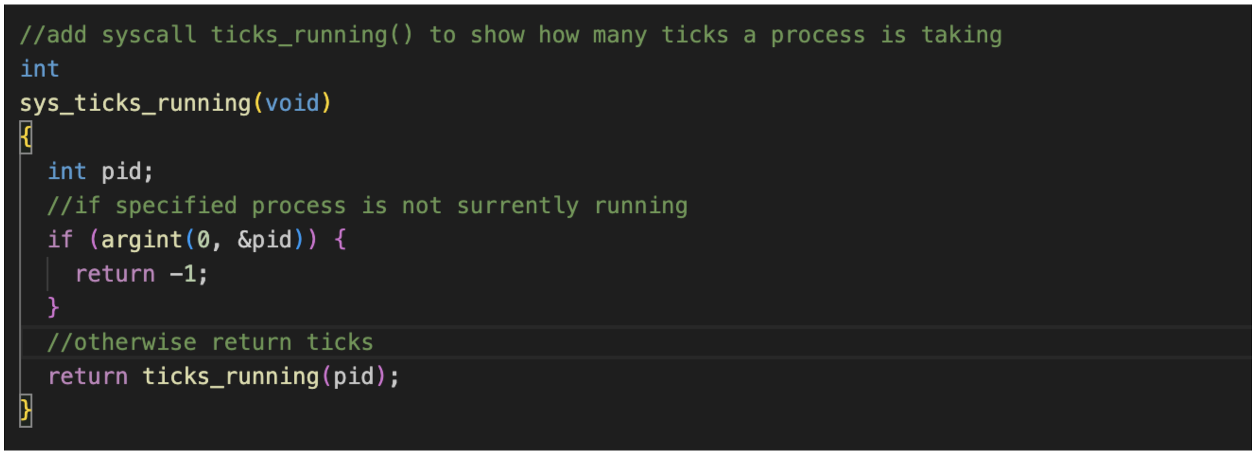
```
    }

    //ADDED:increment tick count while process is running
    if (proc && proc->state == RUNNING) {
      proc-> ticks_ran++;
    }
```

Ticks_running function in proc.c

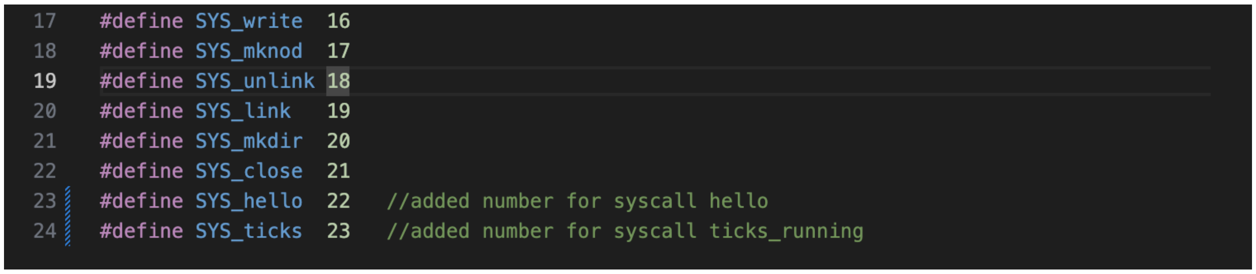


```
//FUNCTION ticks_running
//looks up the process by its pid and returns its ticks so far
//chatgpt was used to help with this function https://chatgpt.com/c/68e5abdf-2848-8332-918a-049c0da7f09e
int
ticks_running(int pid)
{
  struct proc *p;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->pid == pid){
      int result = p->ticks_ran;
      release(&ptable.lock);
      return result;  // returns 0 if it exists but hasn't run yet
    }
  }
  release(&ptable.lock);
  return -1; // no such process
}
```

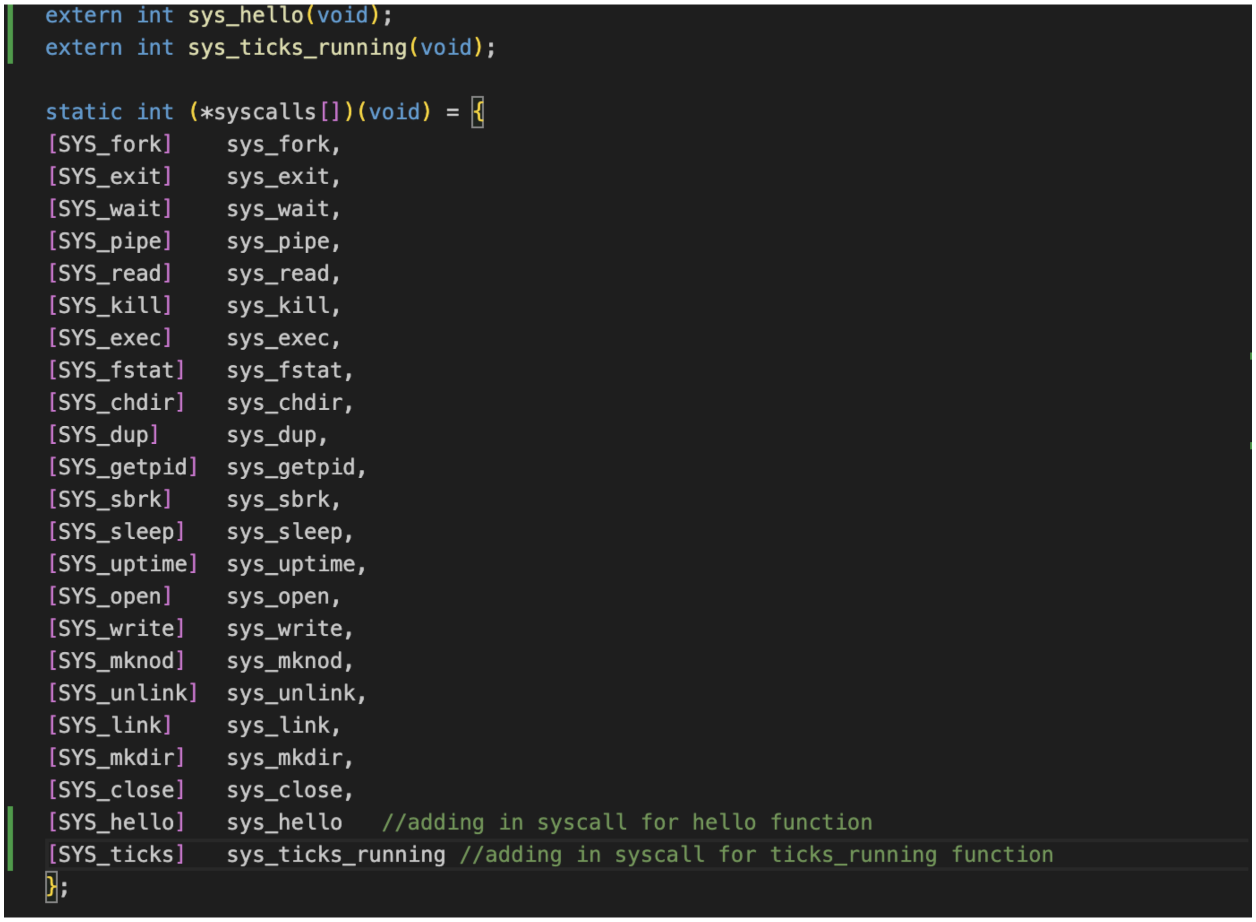
Declare ticks_running in files so it can work as syscall



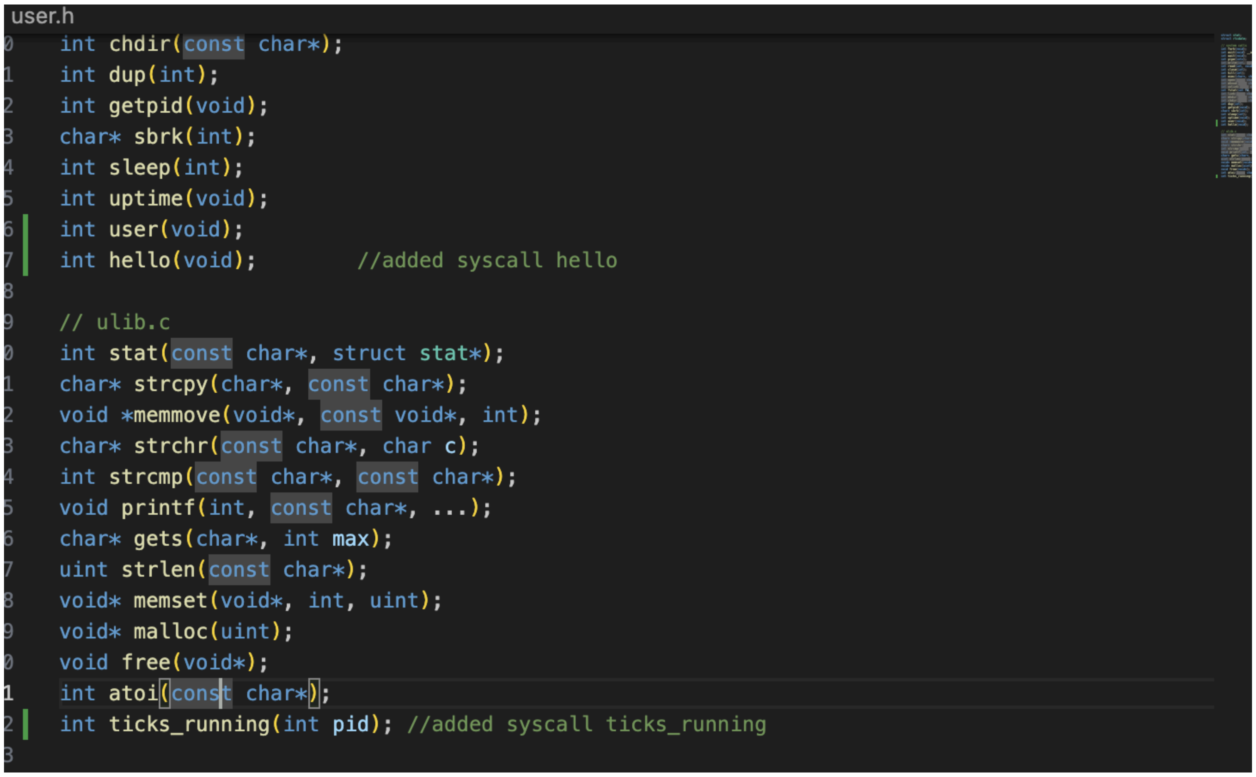
```c
//add syscall ticks_running() to show how many ticks a process is taking
int
sys_ticks_running(void)
{
  int pid;
  //if specified process is not surrently running
  if (argint(0, &pid)) {
    return -1;
  }
  //otherwise return ticks
  return ticks_running(pid);
}
```



```c
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_hello  22   //added number for syscall hello
#define SYS_ticks  23   //added number for syscall ticks_running
```

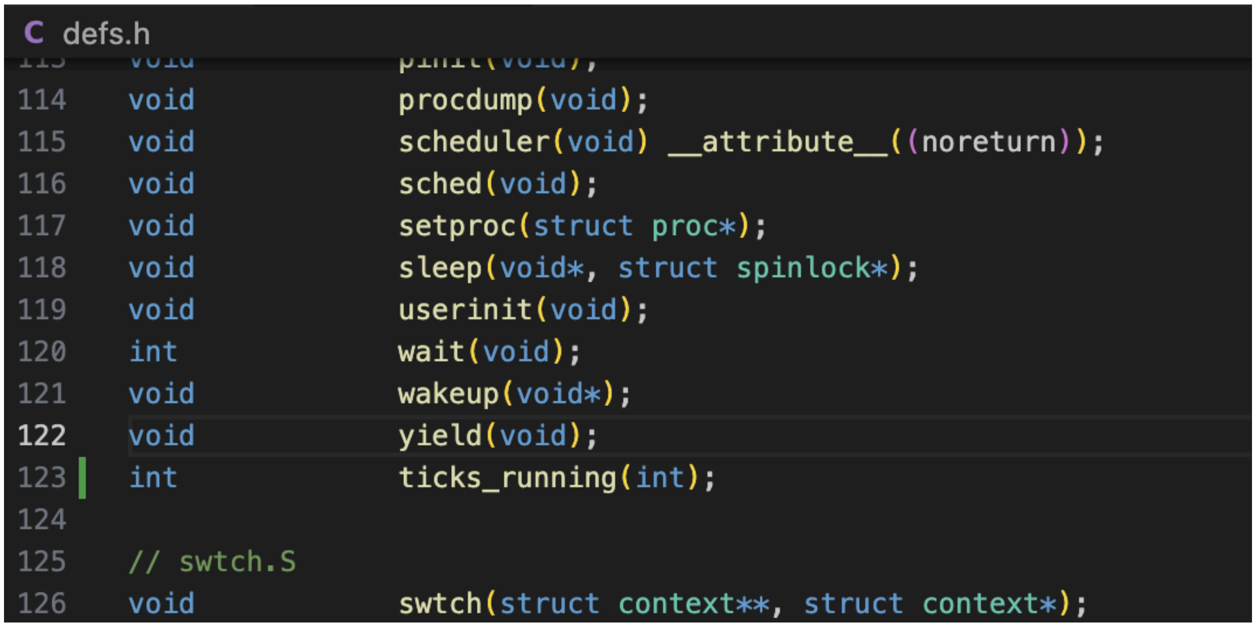


```c
extern int sys_hello(void);
extern int sys_ticks_running(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_hello]   sys_hello   //adding in syscall for hello function
[SYS_ticks]   sys_ticks_running //adding in syscall for ticks_running function
};
```

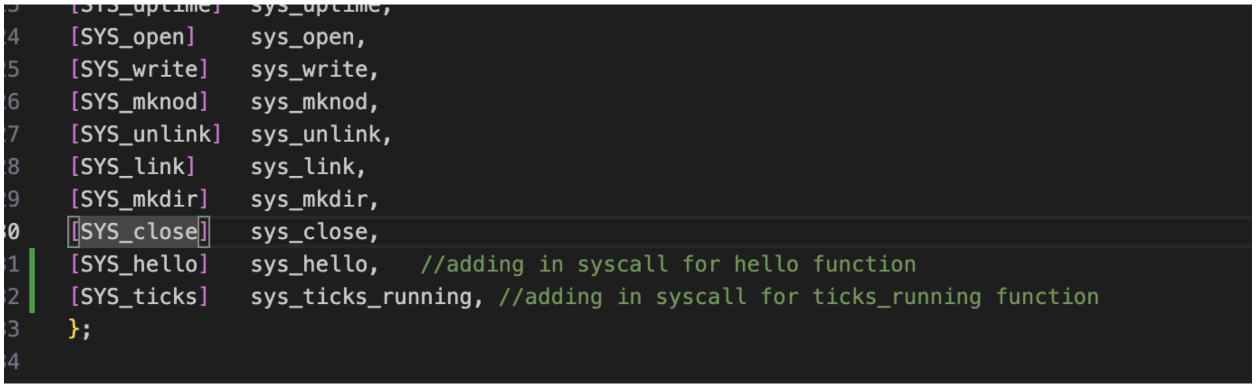


user.h

```c
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int user(void);
int hello(void);        //added syscall hello

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
int ticks_running(int pid); //added syscall ticks_running
```

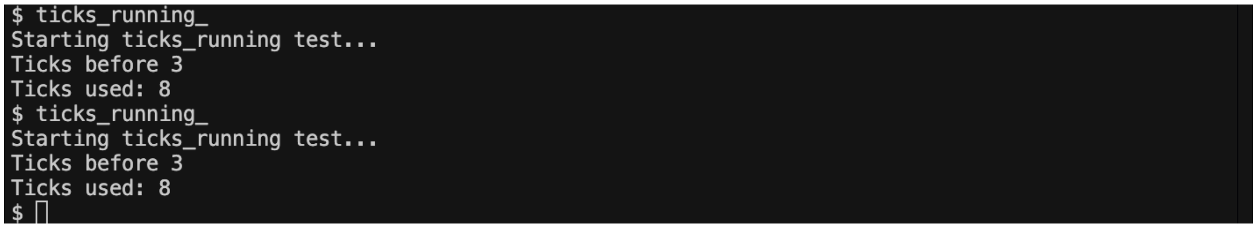


```
C defs.h
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
int             ticks_running(int);

// swtch.S
void            swtch(struct context**, struct context*);
```



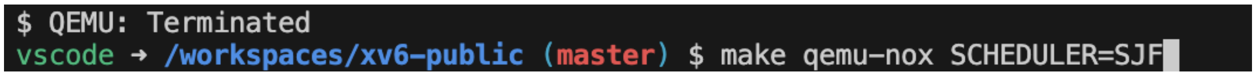
```
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_hello]   sys_hello,   //adding in syscall for hello function
[SYS_ticks]   sys_ticks_running, //adding in syscall for ticks_running function
};
```

Test for ticks_running function output



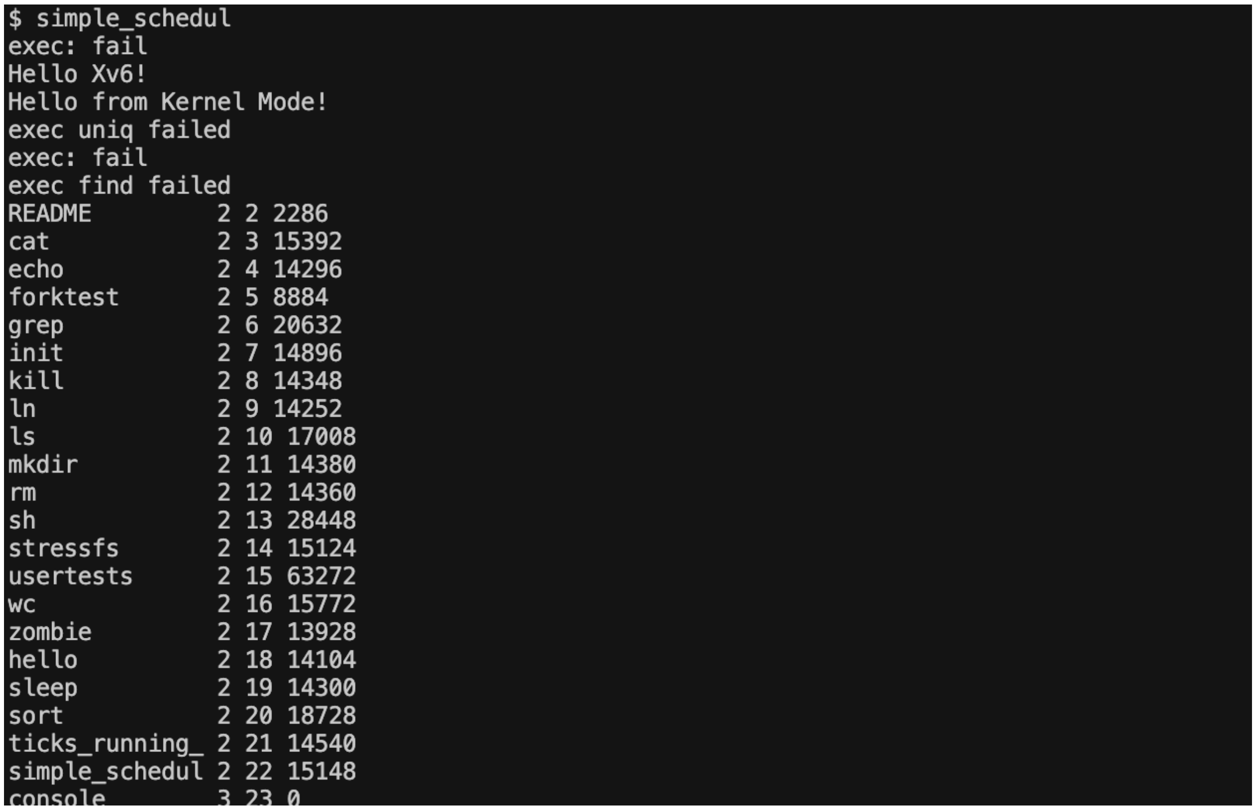
```
$ ticks_running_
Starting ticks_running test...
Ticks before 3
Ticks used: 8
$ ticks_running_
Starting ticks_running test...
Ticks before 3
Ticks used: 8
$
```

Make qemu command with sjf scheduler set



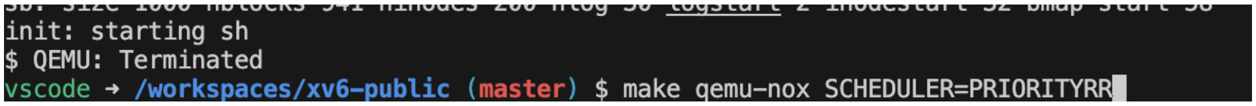
```
$ QEMU: Terminated
vscode ➜ /workspaces/xv6-public (master) $ make qemu-nox SCHEDULER=SJF
```

Simple_scheduler_test command output showing the commands being executed



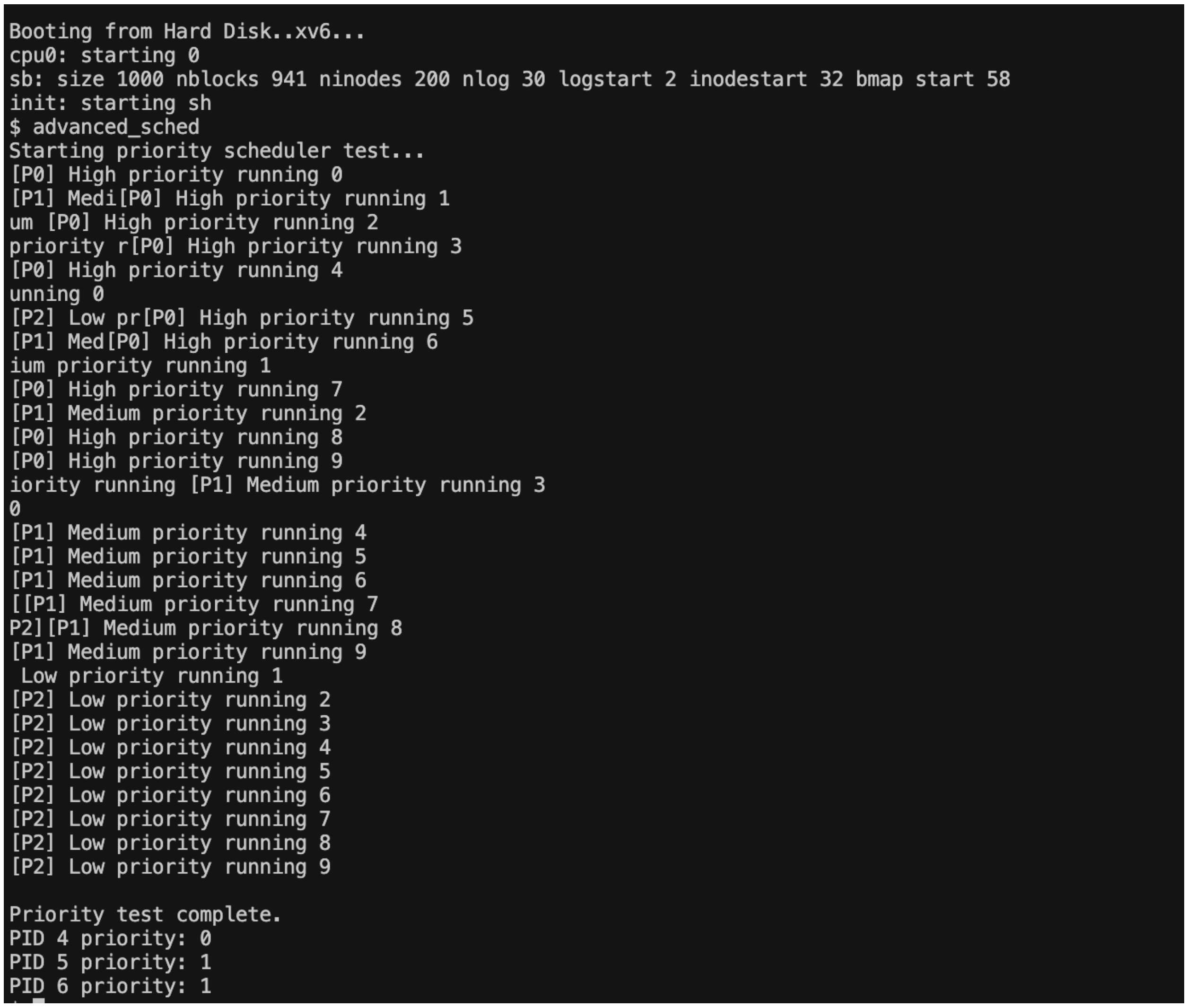
```
$ simple_schedul
exec: fail
Hello Xv6!
Hello from Kernel Mode!
exec uniq failed
exec: fail
exec find failed
README         2 2 2286
cat            2 3 15392
echo           2 4 14296
forktest       2 5 8884
grep           2 6 20632
init           2 7 14896
kill           2 8 14348
ln             2 9 14252
ls             2 10 17008
mkdir          2 11 14380
rm             2 12 14360
sh             2 13 28448
stressfs       2 14 15124
usertests      2 15 63272
wc             2 16 15772
zombie         2 17 13928
hello          2 18 14104
sleep          2 19 14300
sort           2 20 18728
ticks_running_ 2 21 14540
simple_schedul 2 22 15148
console        3 23 0
```

Make qemu command for the multi level priority with round robin scheduler set



```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ QEMU: Terminated
vscode ➜ /workspaces/xv6-public (master) $ make qemu-nox SCHEDULER=PRIORITYRR
```

Advanced_scheduler_test command output



```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ advanced_sched
Starting priority scheduler test...
[P0] High priority running 0
[P1] Medi[P0] High priority running 1
um [P0] High priority running 2
priority r[P0] High priority running 3
[P0] High priority running 4
unning 0
[P2] Low pr[P0] High priority running 5
[P1] Med[P0] High priority running 6
ium priority running 1
[P0] High priority running 7
[P1] Medium priority running 2
[P0] High priority running 8
[P0] High priority running 9
iority running [P1] Medium priority running 3
0
[P1] Medium priority running 4
[P1] Medium priority running 5
[P1] Medium priority running 6
[[P1] Medium priority running 7
P2][P1] Medium priority running 8
[P1] Medium priority running 9
 Low priority running 1
[P2] Low priority running 2
[P2] Low priority running 3
[P2] Low priority running 4
[P2] Low priority running 5
[P2] Low priority running 6
[P2] Low priority running 7
[P2] Low priority running 8
[P2] Low priority running 9

Priority test complete.
PID 4 priority: 0
PID 5 priority: 1
PID 6 priority: 1
```