In user space, following a program that calls getpid():
getpid() is declared in user.h.

- int getpid(void);

getpid() is defined in usys.S.

- SYSCALL(getpid)
- This file contains a macro that expands to:
    - #define SYSCALL(getpid)

```
 .globl getpid;                    #Declares getpid as a global symbol.
  getpid:                          #Entry point of getpid.
    movl $SYS_ getpid, %eax;       #Put system call number in eax register.
    int $T_SYSCALL;                #Trigger a software interrupt, enter the kernel.
    ret                            #Return result to the caller of getpid.
```

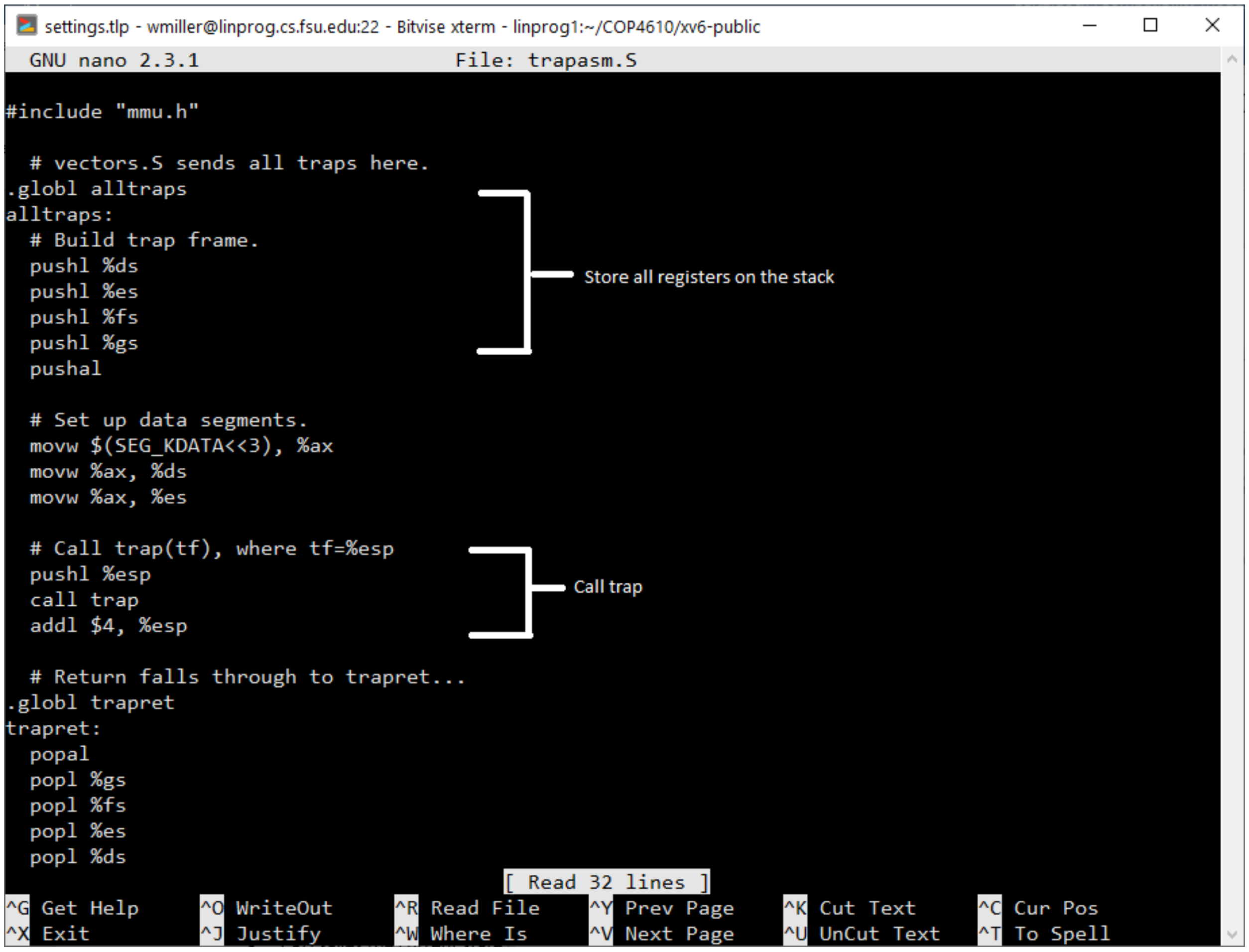
$SYS_getpid is defined in syscall.h.

- #define $SYS_getpid 11

$T_SYSCALL is defined in traps.h.

Int $T_SYSCALL triggers a software interrupt and begins preparing to change from user space to kernel space.

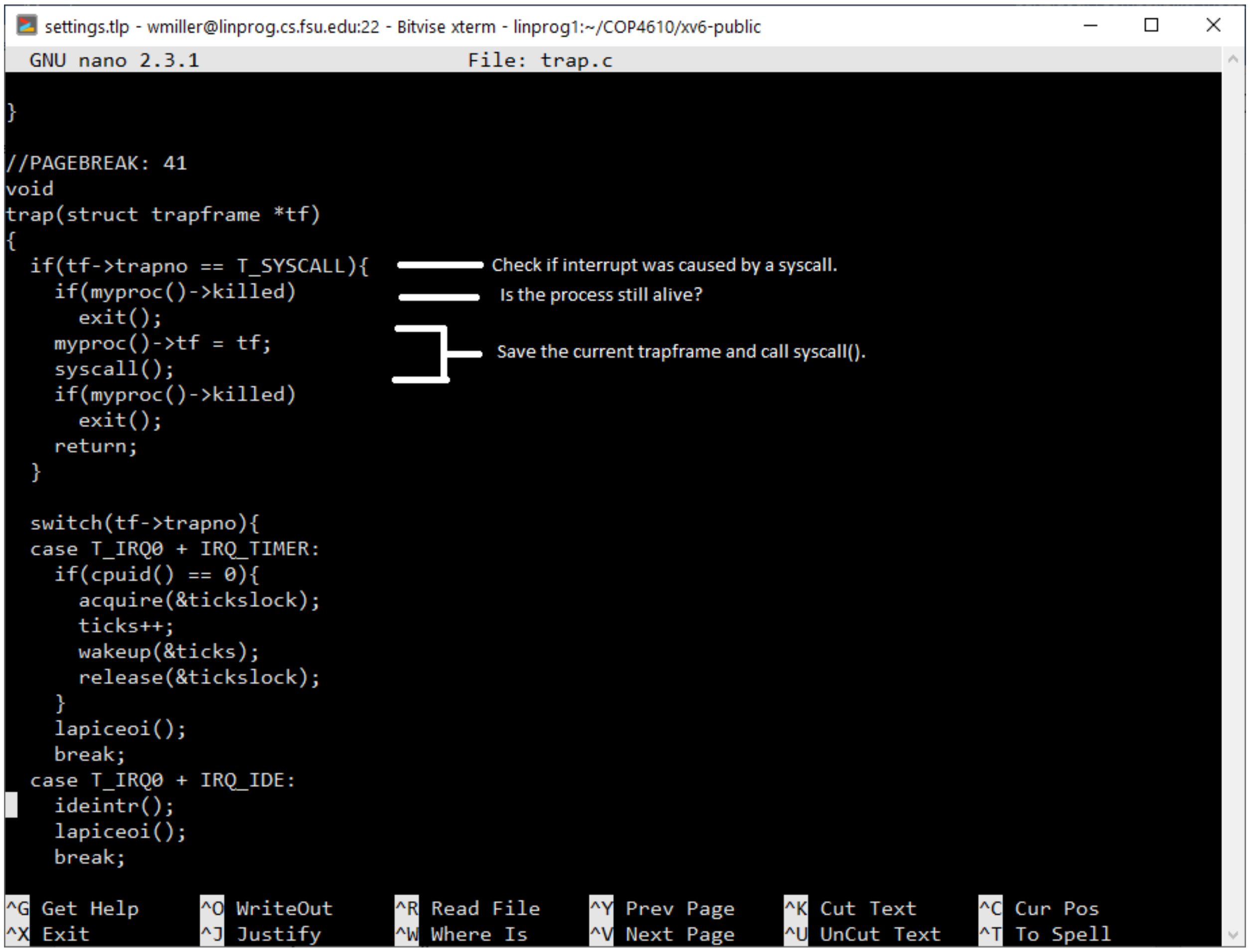
- CPU saves the current state and calls the interrupt handler.
- The interrupt handler for T_SYSCALL is a vector64.
    - Found in vector.S, which is generated by vectors.p1.
- Vector64 jumps to alltraps function (trapasm.S).
- alltraps creates a trapframe and calls trap(struct trapframe *tf). (trap.c).
- struct trapframe (x86.h) saves the user-space registers and tf->eax contains the system call number (SYS_getpid).



```
#include "mmu.h"

  # vectors.S sends all traps here.
.globl alltraps
alltraps:
  # Build trap frame.
  pushl %ds
  pushl %es
  pushl %fs
  pushl %gs
  pushal

  # Set up data segments.
  movw $(SEG_KDATA<<3), %ax
  movw %ax, %ds
  movw %ax, %es

  # Call trap(tf), where tf=%esp
  pushl %esp
  call trap
  addl $4, %esp

  # Return falls through to trapret...
.globl trapret
trapret:
  popal
  popl %gs
  popl %fs
  popl %es
  popl %ds
```

trap(struct trapframe *tf) first checks if trap was invoked by a system call.

- if(tf->trapno == T_SYSCALL)
  - In this case, it was so function enters condition.
- trapframe tf is saved to the current process control block (PCB).
  - myproc()->tf = tf;
- Calls syscall().
- Then returns to alltraps.
  - Which restores all the user registers from the stack.
  - Returns to user space with iret.


settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

```
  GNU nano 2.3.1                     File: trap.c

}

//PAGEBREAK: 41
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }

  switch(tf->trapno){
  case T_IRQ0 + IRQ_TIMER:
    if(cpuid() == 0){
      acquire(&tickslock);
      ticks++;
      wakeup(&ticks);
      release(&tickslock);
    }
    lapiceoi();
    break;
  case T_IRQ0 + IRQ_IDE:
    ideintr();
    lapiceoi();
    break;

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

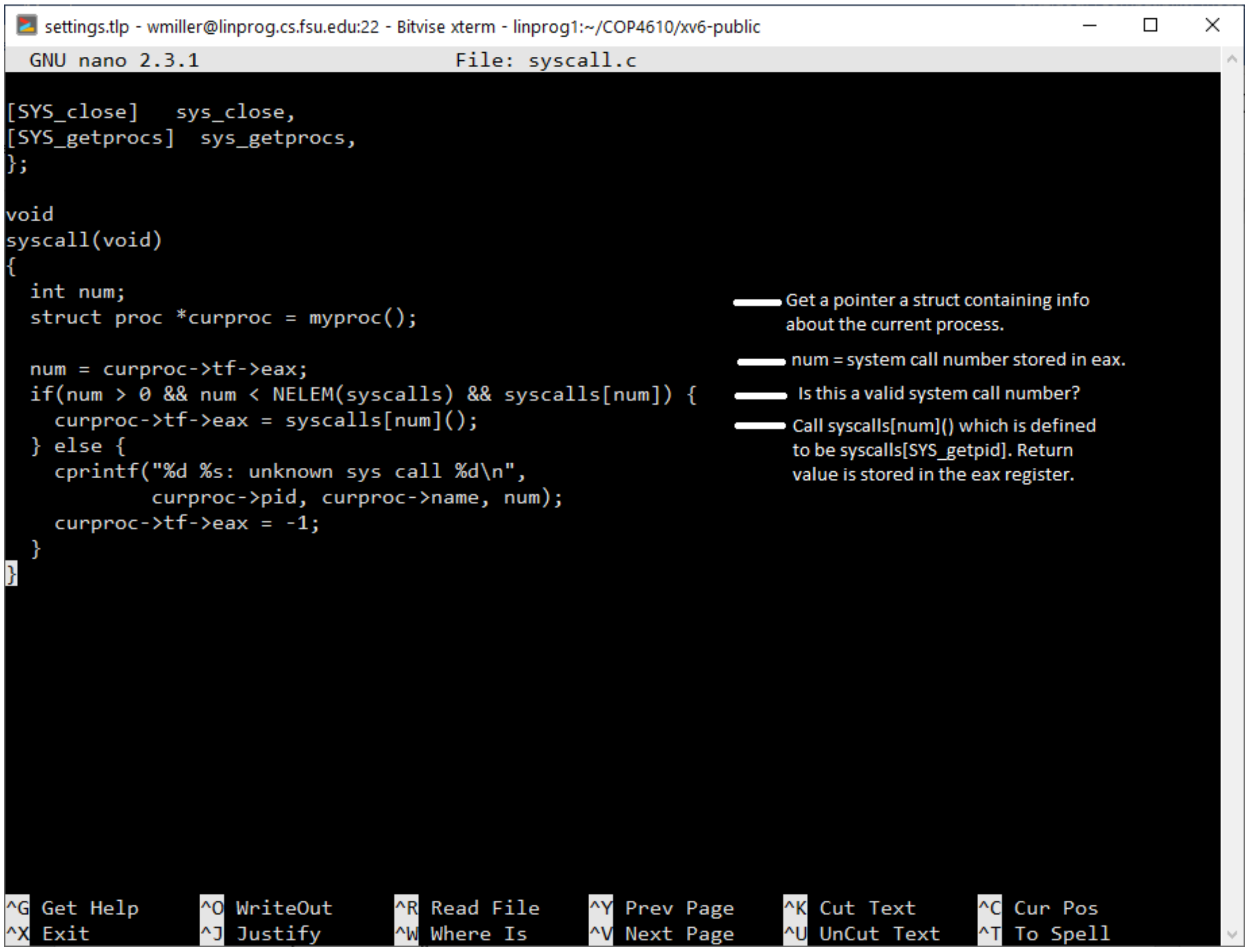
Check if interrupt was caused by a syscall.

Is the process still alive?

Save the current trapframe and call syscall().

Syscall() reads the syscall number in eax (11 for getpid), and calls sys_getpid by:

- Getting a proc pointer to the current process which has the syscall number in the eax register.
    - struct proc *curproc = myproc();
    - num = curproc->tf->eax;
- calls syscalls[SYS_getpid] by invoking syscalls[num]().
- sys_getpid is defined to be syscalls[SYS_getpid] (syscall.c).
- Return value is saved in tf->eax.
    - curproc->tf->eax = syscalls[num]();
- Control is returned to trap.
    - Which then returns to alltraps, restores registers, and returns to userspace.


settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

```
  GNU nano 2.3.1                    File: syscall.c

[SYS_close]   sys_close,
[SYS_getprocs]  sys_getprocs,
};

void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

Get a pointer a struct containing info about the current process.

num = system call number stored in eax.

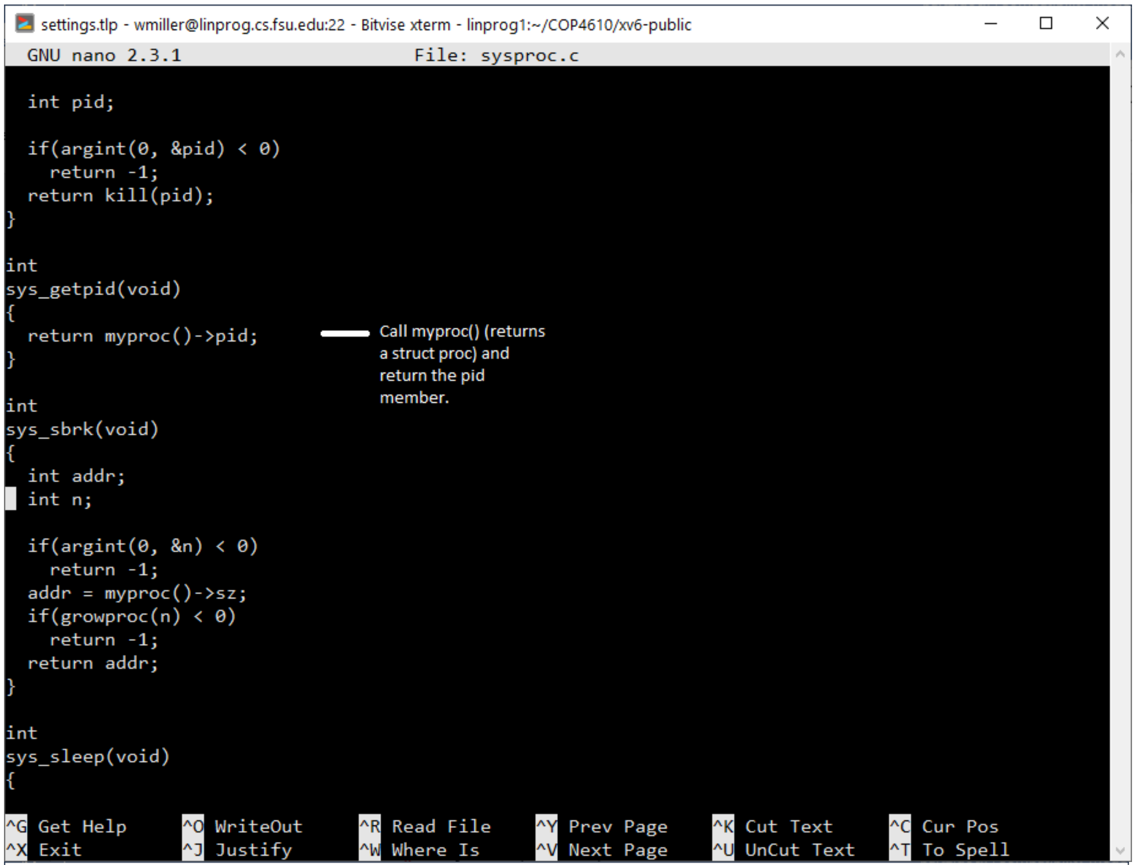
Is this a valid system call number?

Call syscalls[num]() which is defined to be syscalls[SYS_getpid]. Return value is stored in the eax register.

```
^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

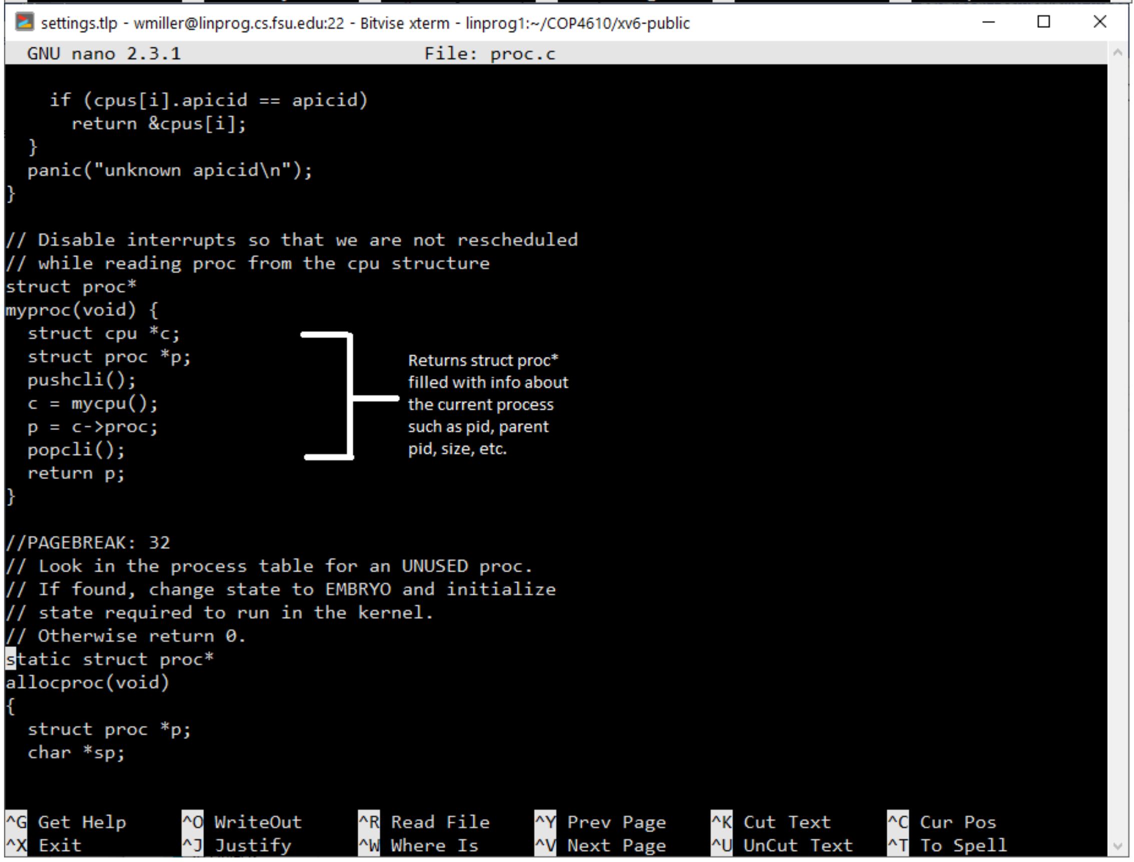
sys_getpid (sysproc.c) returns myproc()->pid.

- myproc() defined in proc.c.
    - returns a struct proc defined in proc.h.
    - calls pushcli() to disable interrupts to make sure process isn't rescheduled while retrieving data. popcli() to return schedule interrupt functionality.
- myproc() creates and returns a struct proc (proc.h) that contains info about the current process running on the cpu.
    - Size of memory, state, pid, parent, etc.
- pid member of the proc struct is returned to syscall().
    - Which is then stored in tf->eax.


settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

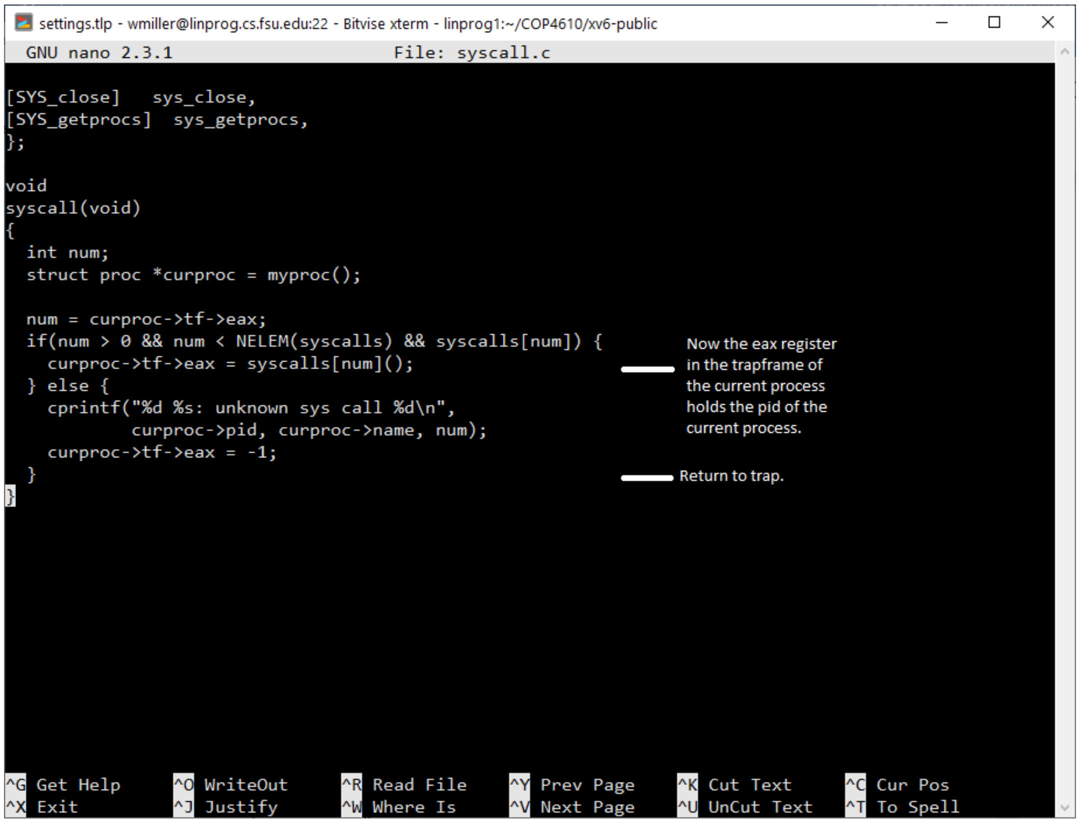
```
  GNU nano 2.3.1                     File: sysproc.c

  int pid;

  if(argint(0, &pid) < 0)
    return -1;
  return kill(pid);
}

int
sys_getpid(void)
{
  return myproc()->pid;
}

int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}

int
sys_sleep(void)
{

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Call myproc() (returns a struct proc) and return the pid member.


settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

```
  GNU nano 2.3.1                     File: proc.c

    if (cpus[i].apicid == apicid)
      return &cpus[i];
  }
  panic("unknown apicid\n");
}

// Disable interrupts so that we are not rescheduled
// while reading proc from the cpu structure
struct proc*
myproc(void) {
  struct cpu *c;
  struct proc *p;
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
  return p;
}

//PAGEBREAK: 32
// Look in the process table for an UNUSED proc.
// If found, change state to EMBRYO and initialize
// state required to run in the kernel.
// Otherwise return 0.
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Returns struct proc* filled with info about the current process such as pid, parent pid, size, etc.

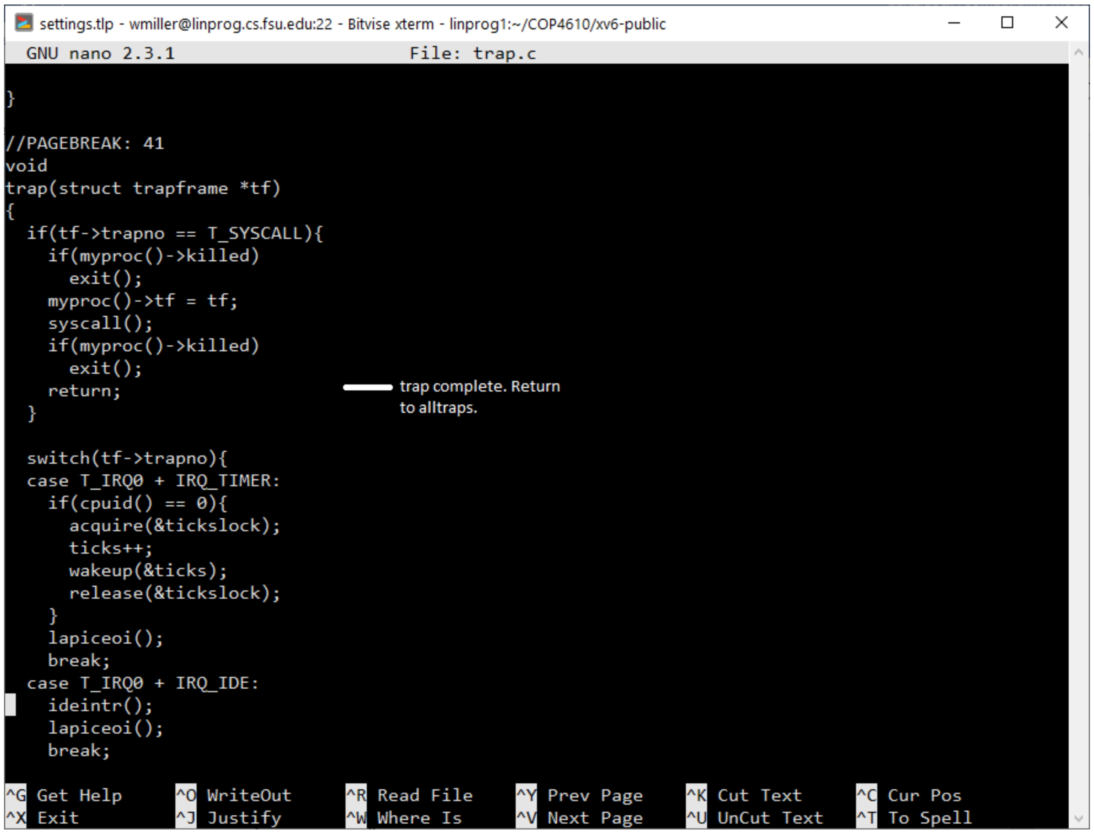


settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

```
  GNU nano 2.3.1                   File: syscall.c

[SYS_close]   sys_close,
[SYS_getprocs]  sys_getprocs,
};

void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

Now the eax register in the trapframe of the current process holds the pid of the current process.

Return to trap.



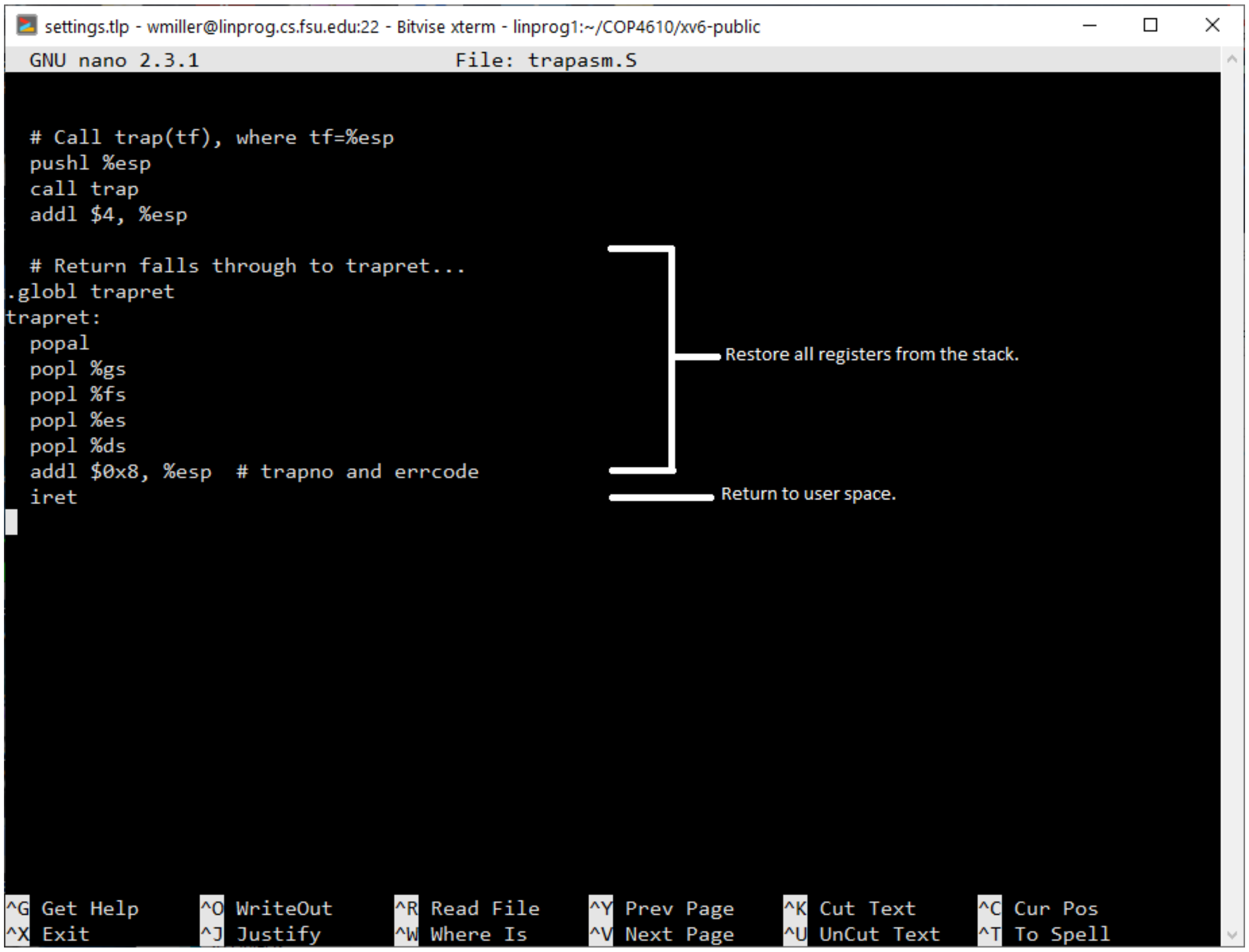
settings.tlp - wmiller@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog1:~/COP4610/xv6-public

```
  GNU nano 2.3.1                   File: trap.c

}

//PAGEBREAK: 41
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
    return;
  }

  switch(tf->trapno){
  case T_IRQ0 + IRQ_TIMER:
    if(cpuid() == 0){
      acquire(&tickslock);
      ticks++;
      wakeup(&ticks);
      release(&tickslock);
    }
    lapiceoi();
    break;
  case T_IRQ0 + IRQ_IDE:
    ideintr();
    lapiceoi();
    break;
```

trap complete. Return to alltraps.

Transition back to user space by restoring the registers in alltraps after calling trap:

- Before jumping back to user space, we restore the user registers from the stack after returning from trap call in alltraps (trapasm.S).
- Returns to userspace after calling iret.



```
  GNU nano 2.3.1                      File: trapasm.S


  # Call trap(tf), where tf=%esp
  pushl %esp
  call trap
  addl $4, %esp

  # Return falls through to trapret...
.globl trapret
trapret:
  popal
  popl %gs
  popl %fs
  popl %es
  popl %ds
  addl $0x8, %esp  # trapno and errcode
  iret
```

Back in userspace:

- Process ID number has been returned to the user space for the calling program.