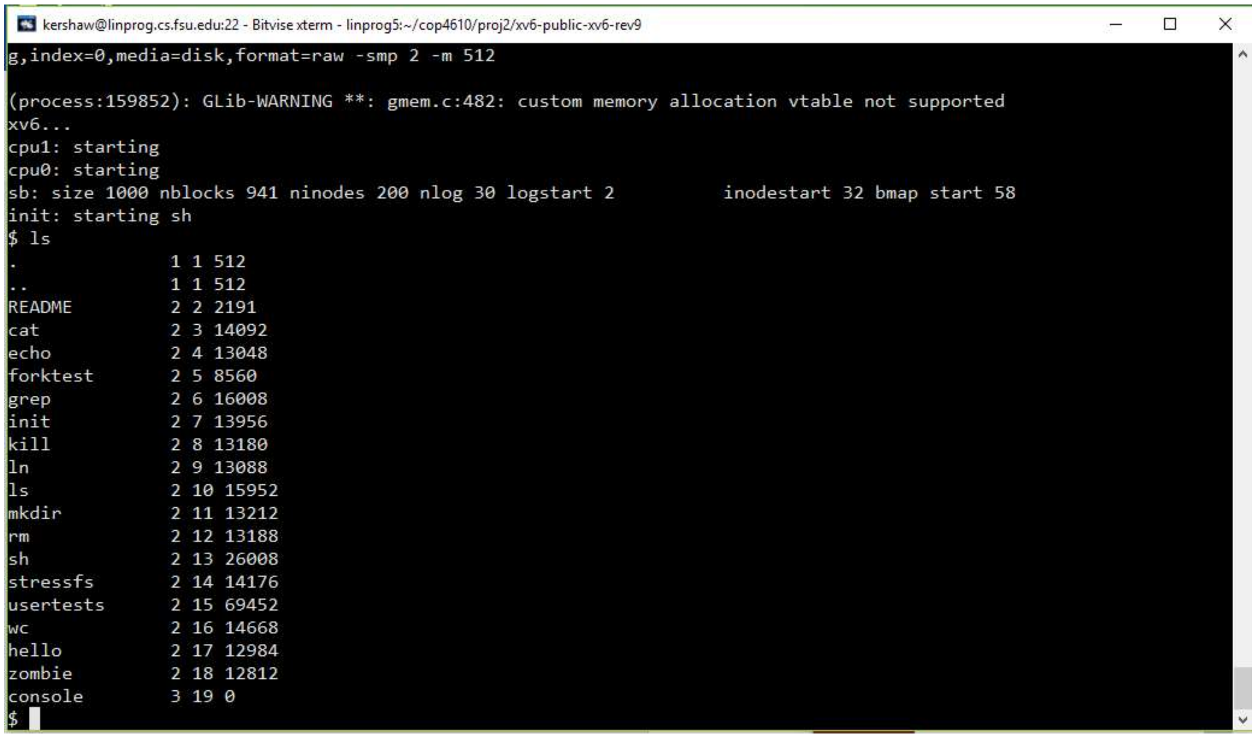
Blake Kershaw
2/7/2018
FSU ID: bk15b
Assignment 2

Part 1: Displaying xv6 running on linprog server with command "ls".



kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog5:~/cop4610/proj2/xv6-public-xv6-rev9

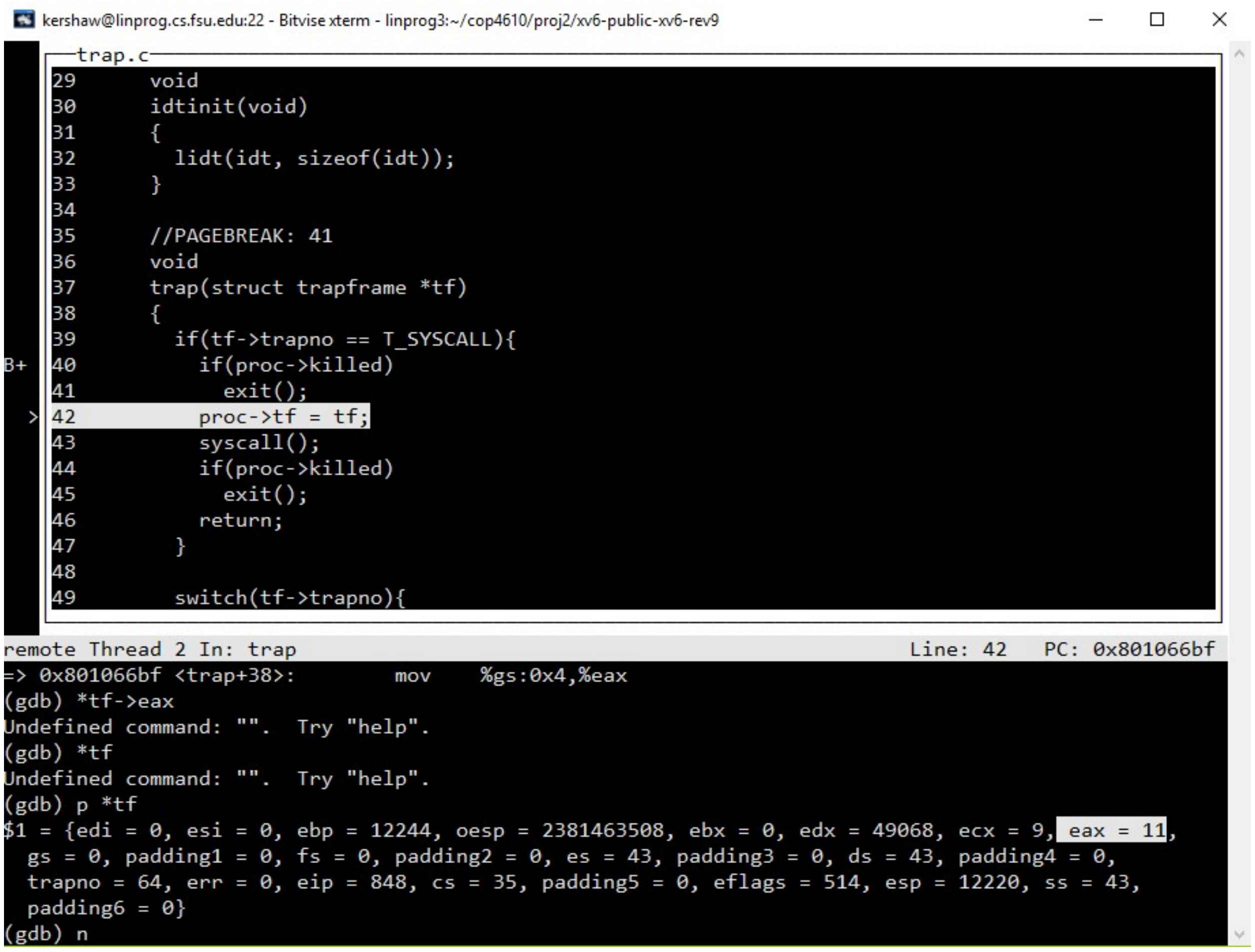
```
g,index=0,media=disk,format=raw -smp 2 -m 512

(process:159852): GLib-WARNING **: gmem.c:482: custom memory allocation vtable not supported
xv6...
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2          inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2191
cat            2 3 14092
echo           2 4 13048
forktest       2 5 8560
grep           2 6 16008
init           2 7 13956
kill           2 8 13180
ln             2 9 13088
ls             2 10 15952
mkdir          2 11 13212
rm             2 12 13188
sh             2 13 26008
stressfs       2 14 14176
usertests      2 15 69452
wc             2 16 14668
hello          2 17 12984
zombie         2 18 12812
console        3 19 0
$ 
```

Part2:

I created a test file hello.c that performs the getpid() call and assigns the value to pid and prints the process id

```
int pid;

int main(int argc, char *argv[])

{   if(argc <= 1){

printf(1,"Who are you?\n");

exit();

}

pid = getpid();

printf(1,"Hello Xv6, %s PID:%d\n", argv[1] ,pid);

exit(); }
```

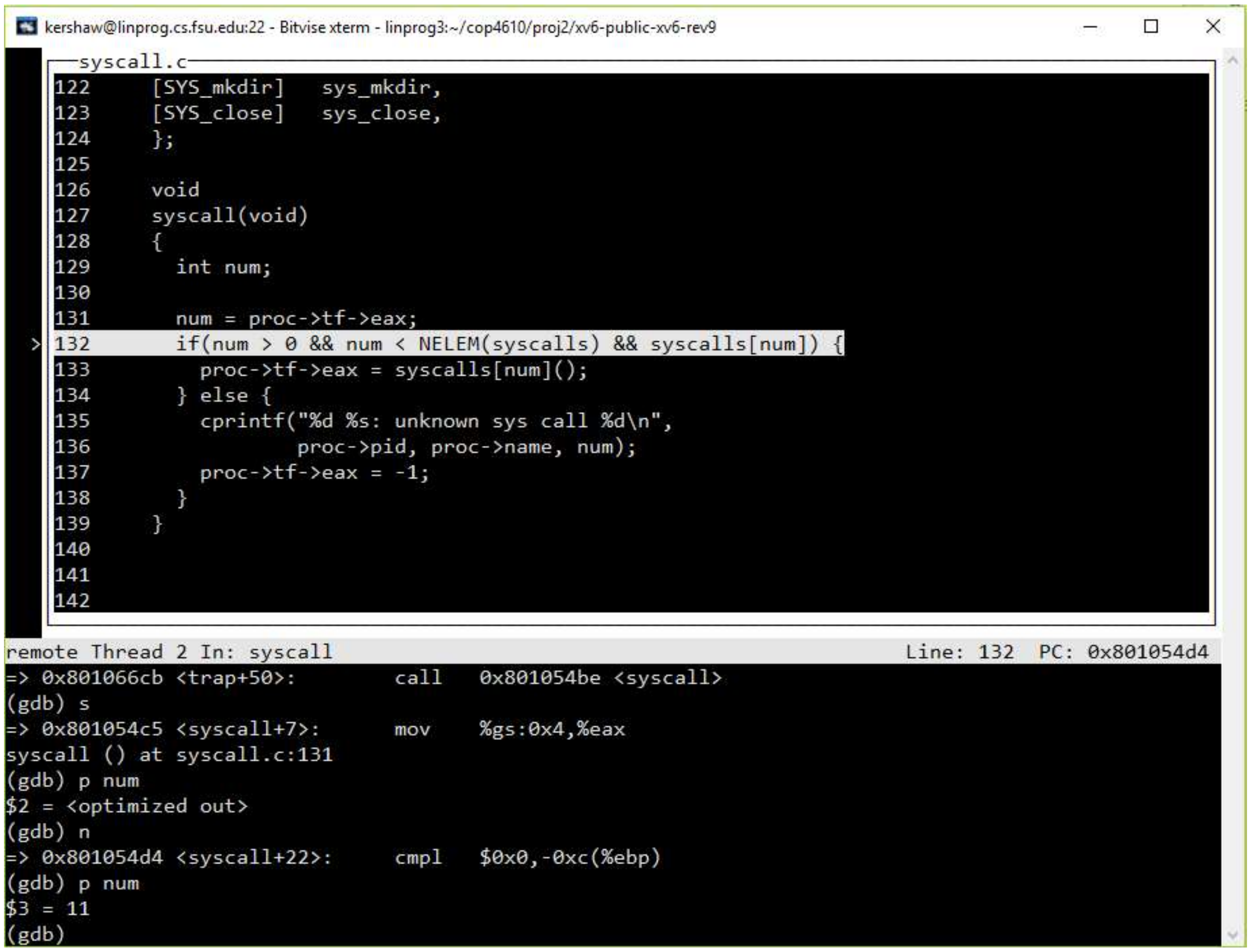
I than began the gdb session with the break at trap, once control was given back to xv6 I called the program hello and set the break point at line 40 of trap.c with the condition of (tf->eax == 11) This is because the syscall id for **getpid is 11** per the syscall.h



```
kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog3:~/cop4610/proj2/xv6-public-xv6-rev9
 ┌─trap.c──────────────────────────────────────────────
 │29      void
 │30      idtinit(void)
 │31      {
 │32        lidt(idt, sizeof(idt));
 │33      }
 │34
 │35      //PAGEBREAK: 41
 │36      void
 │37      trap(struct trapframe *tf)
 │38      {
 │39        if(tf->trapno == T_SYSCALL){
B+ │40          if(proc->killed)
 │41            exit();
 >│42          proc->tf = tf;
 │43          syscall();
 │44          if(proc->killed)
 │45            exit();
 │46          return;
 │47        }
 │48
 │49        switch(tf->trapno){
remote Thread 2 In: trap                                     Line: 42   PC: 0x801066bf
=> 0x801066bf <trap+38>:        mov    %gs:0x4,%eax
(gdb) *tf->eax
Undefined command: "".  Try "help".
(gdb) *tf
Undefined command: "".  Try "help".
(gdb) p *tf
$1 = {edi = 0, esi = 0, ebp = 12244, oesp = 2381463508, ebx = 0, edx = 49068, ecx = 9, eax = 11,
  gs = 0, padding1 = 0, fs = 0, padding2 = 0, es = 43, padding3 = 0, ds = 43, padding4 = 0,
  trapno = 64, err = 0, eip = 848, cs = 35, padding5 = 0, eflags = 514, esp = 12220, ss = 43,
  padding6 = 0}
(gdb) n
```

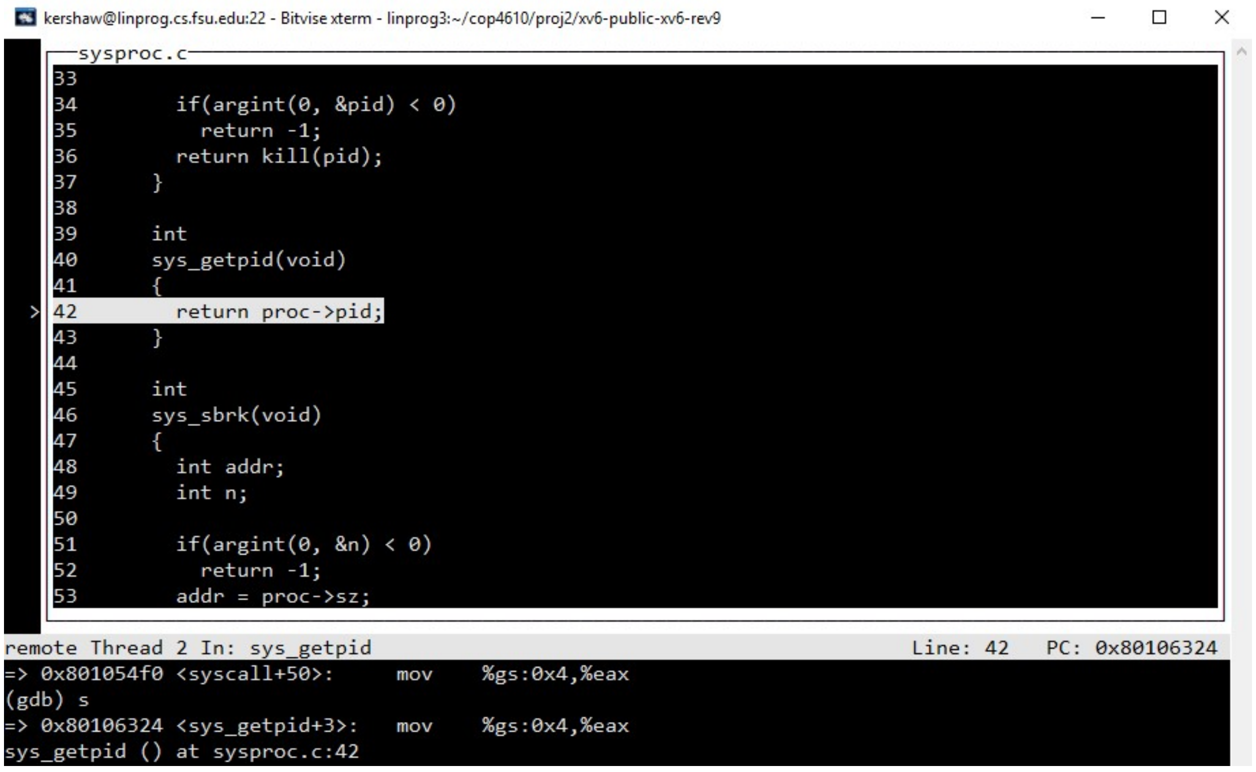
As shown here at line 42, eax = 11 so this is the sycall for getpid().

On line 43 syscall() is called so I step through the syscall function which branches to syscall.c


kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog3:~/cop4610/proj2/xv6-public-xv6-rev9

```
syscall.c
122     [SYS_mkdir]   sys_mkdir,
123     [SYS_close]   sys_close,
124     };
125
126     void
127     syscall(void)
128     {
129       int num;
130
131       num = proc->tf->eax;
> 132     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
133         proc->tf->eax = syscalls[num]();
134       } else {
135         cprintf("%d %s: unknown sys call %d\n",
136                 proc->pid, proc->name, num);
137         proc->tf->eax = -1;
138       }
139     }
140
141
142
remote Thread 2 In: syscall                                Line: 132   PC: 0x801054d4
=> 0x801066cb <trap+50>:        call   0x801054be <syscall>
(gdb) s
=> 0x801054c5 <syscall+7>:      mov    %gs:0x4,%eax
syscall () at syscall.c:131
(gdb) p num
$2 = <optimized out>
(gdb) n
=> 0x801054d4 <syscall+22>:     cmpl   $0x0,-0xc(%ebp)
(gdb) p num
$3 = 11
(gdb)
```

Num is initialized at line 131 with the value in eax(11), this is confirmed on line 132 as p num resulted in $3 = 11. Line 132 if check is a success and continues to line 133 which calls syscalls[num]() or syscalls[11]() and as we step into that function call it opens the function in sysproc.c

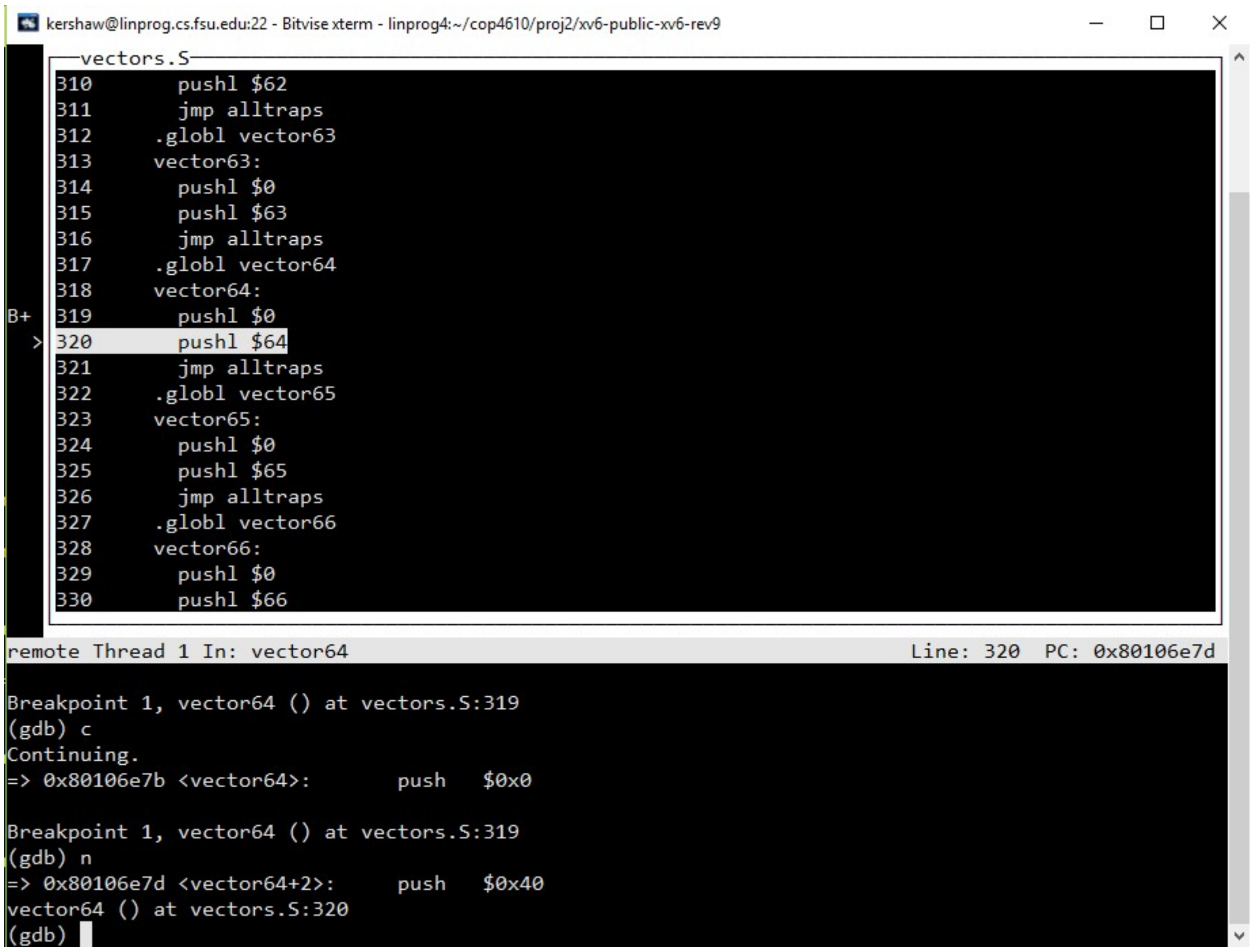


kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog3:~/cop4610/proj2/xv6-public-xv6-rev9

```
sysproc.c

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

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

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

  if(argint(0, &n) < 0)
    return -1;
  addr = proc->sz;
```

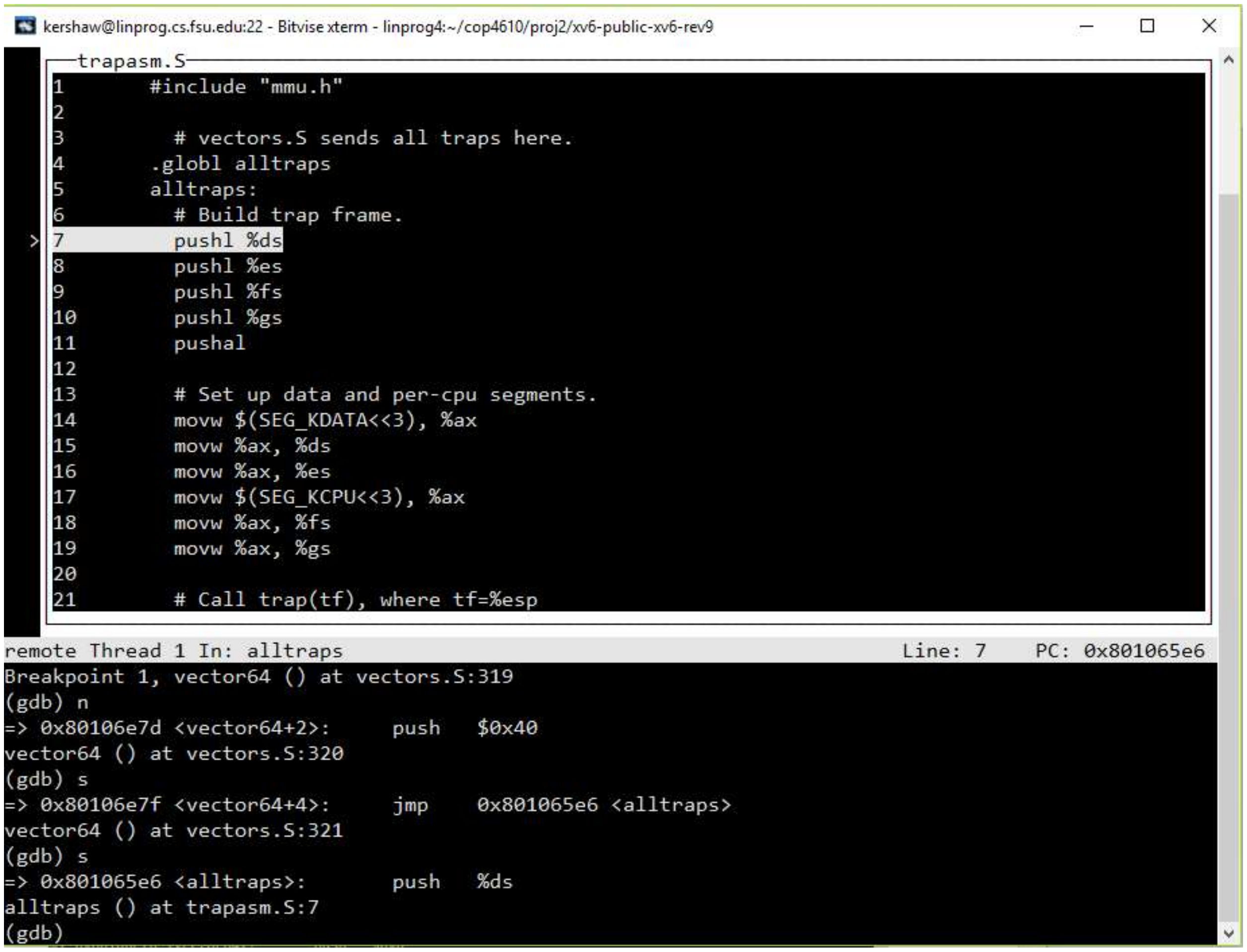
```
remote Thread 2 In: sys_getpid                               Line: 42   PC: 0x80106324
=> 0x801054f0 <syscall+50>:     mov    %gs:0x4,%eax
(gdb) s
=> 0x80106324 <sys_getpid+3>:   mov    %gs:0x4,%eax
sys_getpid () at sysproc.c:42
```

Upon further digging this actually starts back at vector64 inside of vector.S which is responsible for calling all traps found in trapasm.S


kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog4:~/cop4610/proj2/xv6-public-xv6-rev9

```
vectors.S
          pushl $62
          jmp alltraps
        .globl vector63
        vector63:
          pushl $0
          pushl $63
          jmp alltraps
        .globl vector64
        vector64:
B+        pushl $0
  >       pushl $64
          jmp alltraps
        .globl vector65
        vector65:
          pushl $0
          pushl $65
          jmp alltraps
        .globl vector66
        vector66:
          pushl $0
          pushl $66
remote Thread 1 In: vector64                              Line: 320   PC: 0x80106e7d

Breakpoint 1, vector64 () at vectors.S:319
(gdb) c
Continuing.
=> 0x80106e7b <vector64>:       push   $0x0

Breakpoint 1, vector64 () at vectors.S:319
(gdb) n
=> 0x80106e7d <vector64+2>:     push   $0x40
vector64 () at vectors.S:320
(gdb)
```

After jumping to alltraps trapasm.S as shown below, it first builds the trapframe by saving all of the vital information before context switching to the kernel to handle the service call for sys_getpid()

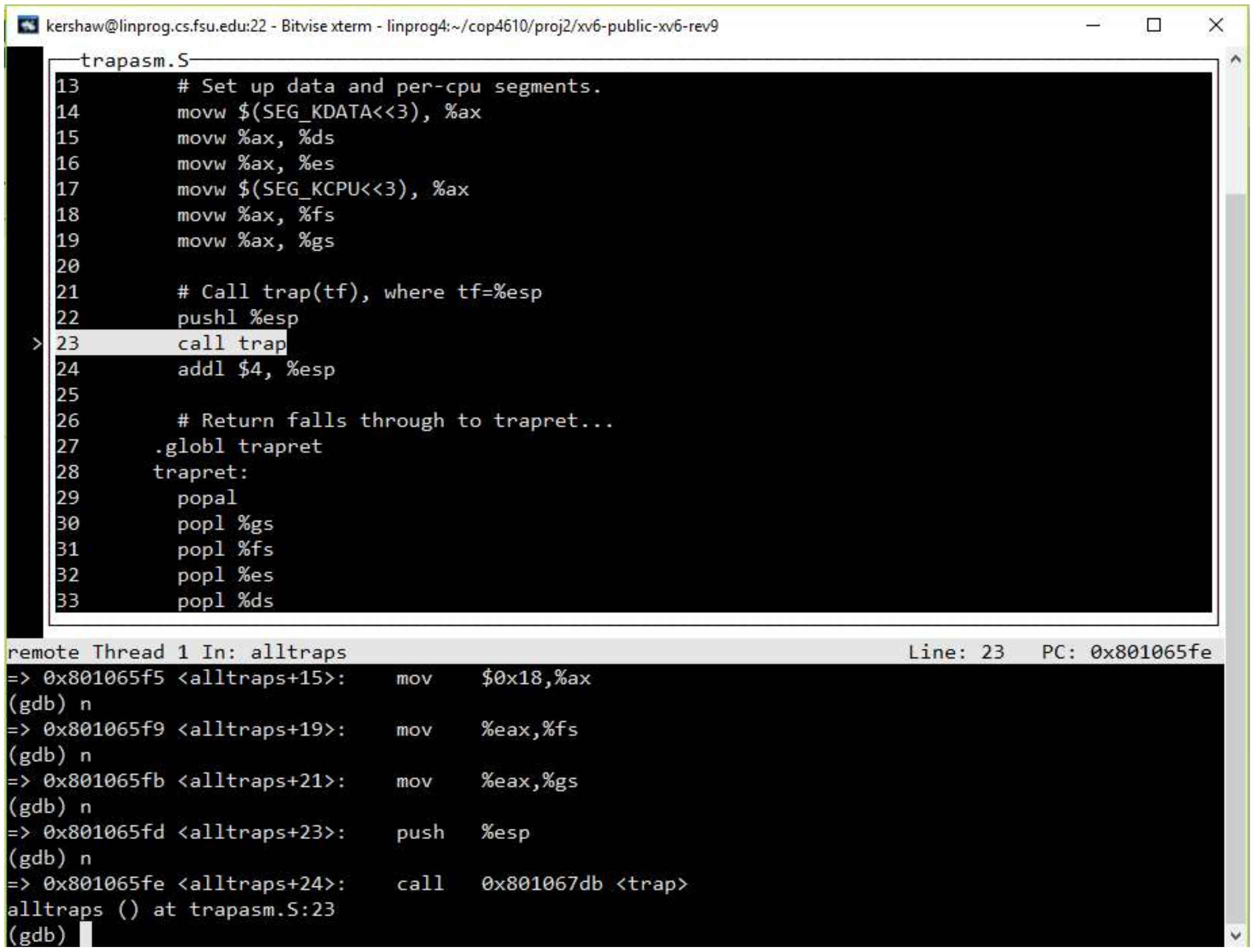


```
kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog4:~/cop4610/proj2/xv6-public-xv6-rev9
```

```
trapasm.S
        #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 and per-cpu segments.
          movw $(SEG_KDATA<<3), %ax
          movw %ax, %ds
          movw %ax, %es
          movw $(SEG_KCPU<<3), %ax
          movw %ax, %fs
          movw %ax, %gs

          # Call trap(tf), where tf=%esp
```

```
remote Thread 1 In: alltraps                                    Line: 7     PC: 0x801065e6
Breakpoint 1, vector64 () at vectors.S:319
(gdb) n
=> 0x80106e7d <vector64+2>:     push   $0x40
vector64 () at vectors.S:320
(gdb) s
=> 0x80106e7f <vector64+4>:     jmp    0x801065e6 <alltraps>
vector64 () at vectors.S:321
(gdb) s
=> 0x801065e6 <alltraps>:       push   %ds
alltraps () at trapasm.S:7
(gdb)
```

After it sets up the trapframe and saves the %esp stack pointer trapasm.S calls the trap function


kershaw@linprog.cs.fsu.edu:22 - Bitvise xterm - linprog4:~/cop4610/proj2/xv6-public-xv6-rev9

```
trapasm.S
      # Set up data and per-cpu segments.
      movw $(SEG_KDATA<<3), %ax
      movw %ax, %ds
      movw %ax, %es
      movw $(SEG_KCPU<<3), %ax
      movw %ax, %fs
      movw %ax, %gs

      # 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
remote Thread 1 In: alltraps                          Line: 23   PC: 0x801065fe
=> 0x801065f5 <alltraps+15>:    mov    $0x18,%ax
(gdb) n
=> 0x801065f9 <alltraps+19>:    mov    %eax,%fs
(gdb) n
=> 0x801065fb <alltraps+21>:    mov    %eax,%gs
(gdb) n
=> 0x801065fd <alltraps+23>:    push   %esp
(gdb) n
=> 0x801065fe <alltraps+24>:    call   0x801067db <trap>
alltraps () at trapasm.S:23
(gdb)
```