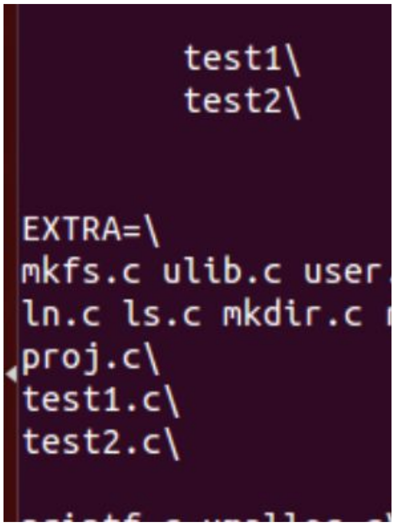
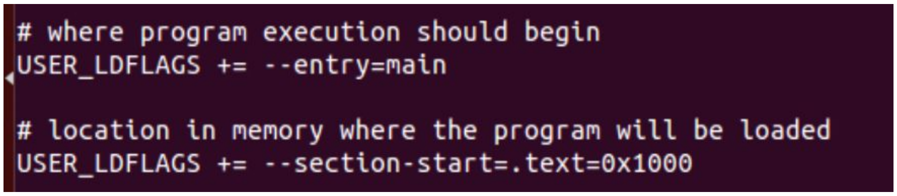
File	Image #	Code	Reason
user/makefile.mk	1	Test1\ Test2\ Test3\	Enables tests to work with qemu
user/makefile.mk	2	USER_LDFLAGS += --section-start=.text = 0x1000	Change location where virtual memory starts
user/test1.c	3	Look at Image 3	Implement test1.c
user/test2.c	4	Look at Image 4	Implement test2.c
user/test3.c	5	Look at Image 5	Implement test3.c
kernel/exec.c	6	Sz = 4096	Change sz to 4096 to leave null page inaccessible
kernel/vm.c	7	for(i = 4096; i < sz; i += PGSIZE)	Change for loop initial value to 4096 that way it doesn’t start at 0
kernel/syscall.c	8	If (proc->pid > 1 && addr < PGSIZE)	Checks if int does not extend outside of stack
kernel/syscall.c	9	If (proc->pid > 1 && addr < PGSIZE)	Checks that the int does not touch code of program
kernel/syscall.c	10	If ((uint)i >= proc->sz || (uint)i + size > proc->sz)	Checks that pointers remain in stack
qemu	11	Test1 and test2 working	Look at Image 11



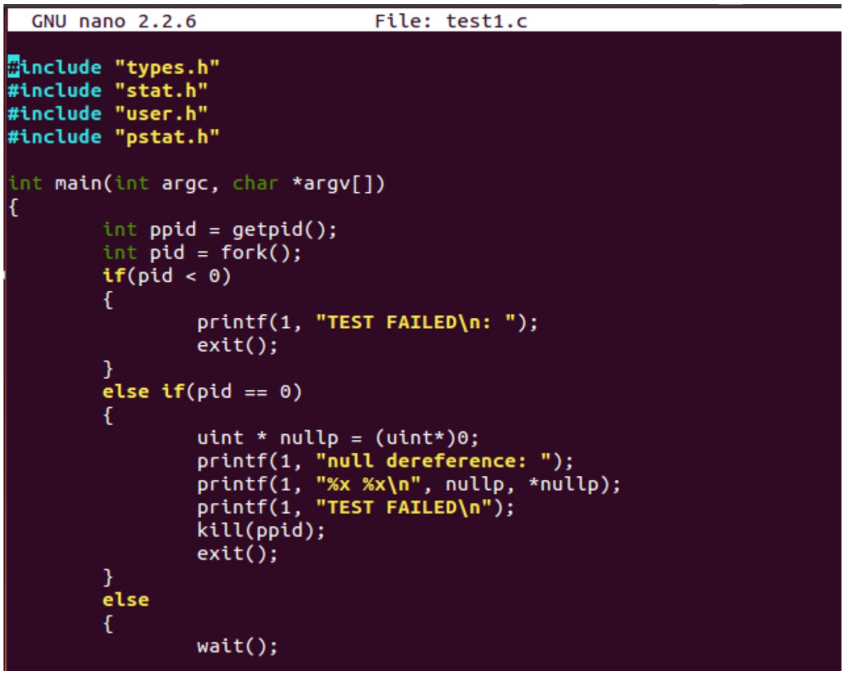
```
        test1\
        test2\


EXTRA=\
mkfs.c ulib.c user.
ln.c ls.c mkdir.c r
proj.c\
test1.c\
test2.c\
```

//Image 1



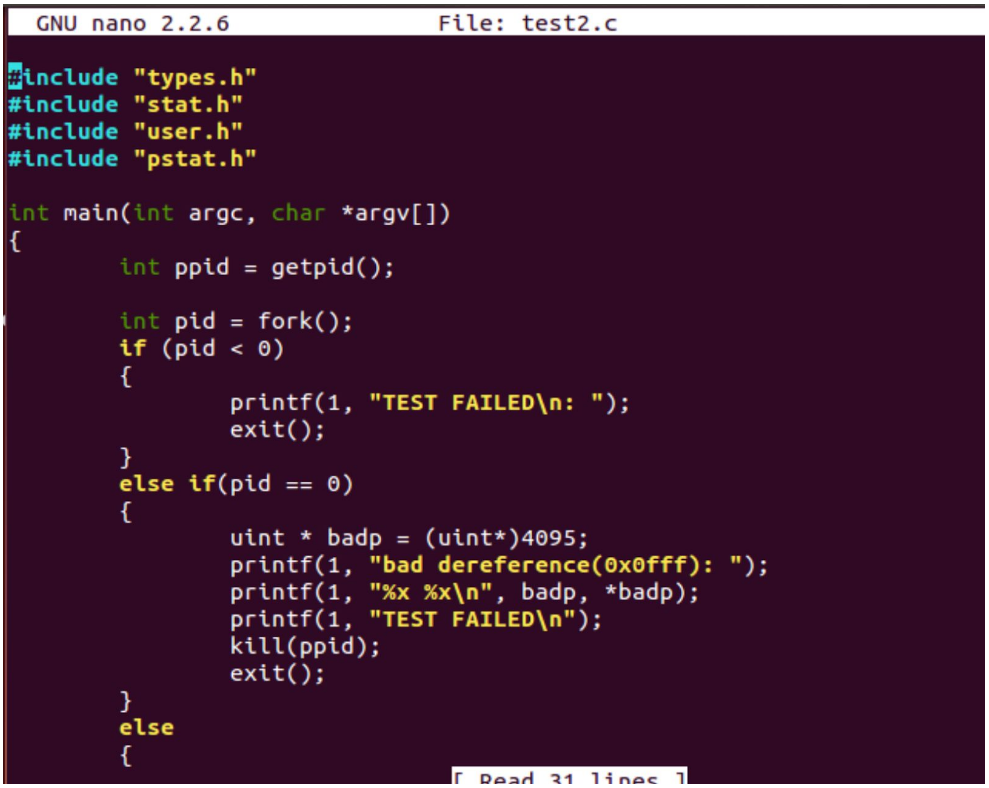
```
# where program execution should begin
USER_LDFLAGS += --entry=main

# location in memory where the program will be loaded
USER_LDFLAGS += --section-start=.text=0x1000
```

//Image 2



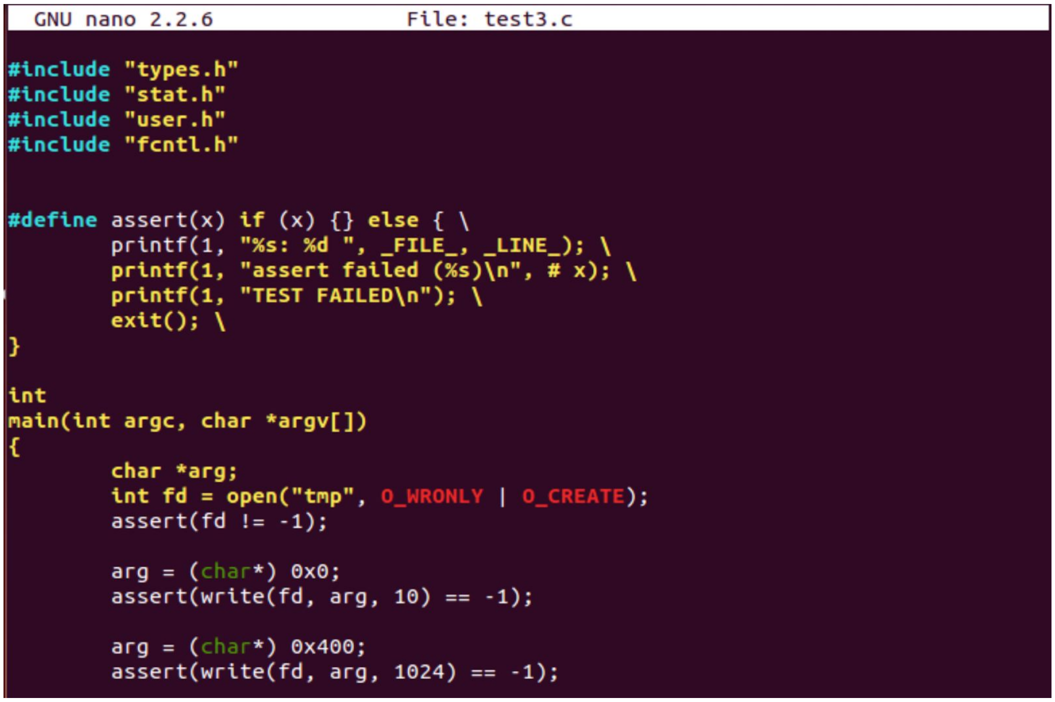
```
  GNU nano 2.2.6                File: test1.c

#include "types.h"
#include "stat.h"
#include "user.h"
#include "pstat.h"

int main(int argc, char *argv[])
{
        int ppid = getpid();
        int pid = fork();
        if(pid < 0)
        {
                printf(1, "TEST FAILED\n: ");
                exit();
        }
        else if(pid == 0)
        {
                uint * nullp = (uint*)0;
                printf(1, "null dereference: ");
                printf(1, "%x %x\n", nullp, *nullp);
                printf(1, "TEST FAILED\n");
                kill(ppid);
                exit();
        }
        else
        {
                wait();
```

//Image 3



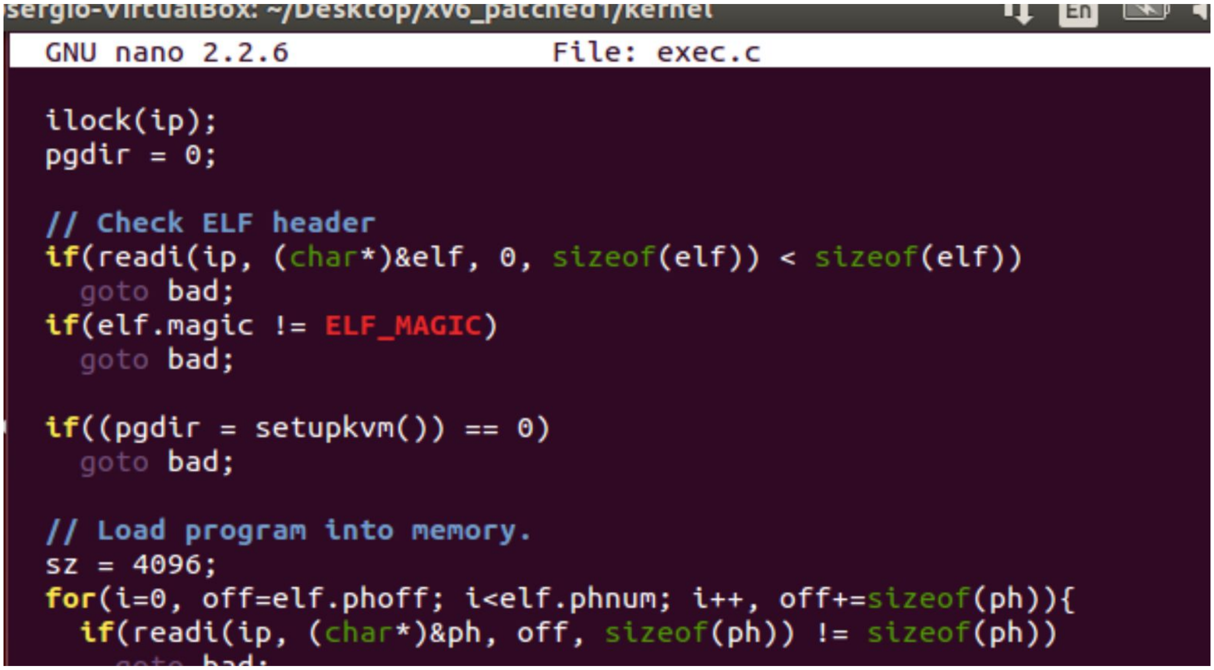
```
  GNU nano 2.2.6                File: test2.c

#include "types.h"
#include "stat.h"
#include "user.h"
#include "pstat.h"

int main(int argc, char *argv[])
{
        int ppid = getpid();

        int pid = fork();
        if (pid < 0)
        {
                printf(1, "TEST FAILED\n: ");
                exit();
        }
        else if(pid == 0)
        {
                uint * badp = (uint*)4095;
                printf(1, "bad dereference(0x0fff): ");
                printf(1, "%x %x\n", badp, *badp);
                printf(1, "TEST FAILED\n");
                kill(ppid);
                exit();
        }
        else
        {
                                [ Read 31 lines ]
```

//Image 4



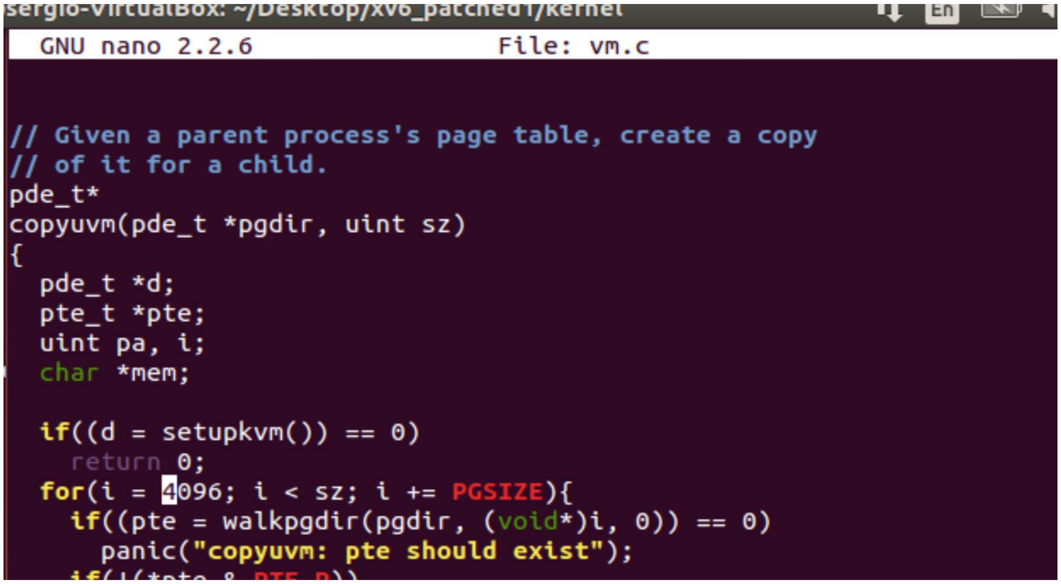
```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"


#define assert(x) if (x) {} else { \
        printf(1, "%s: %d ", _FILE_, _LINE_); \
        printf(1, "assert failed (%s)\n", # x); \
        printf(1, "TEST FAILED\n"); \
        exit(); \
}

int
main(int argc, char *argv[])
{
        char *arg;
        int fd = open("tmp", O_WRONLY | O_CREATE);
        assert(fd != -1);

        arg = (char*) 0x0;
        assert(write(fd, arg, 10) == -1);

        arg = (char*) 0x400;
        assert(write(fd, arg, 1024) == -1);
```

//Image 5



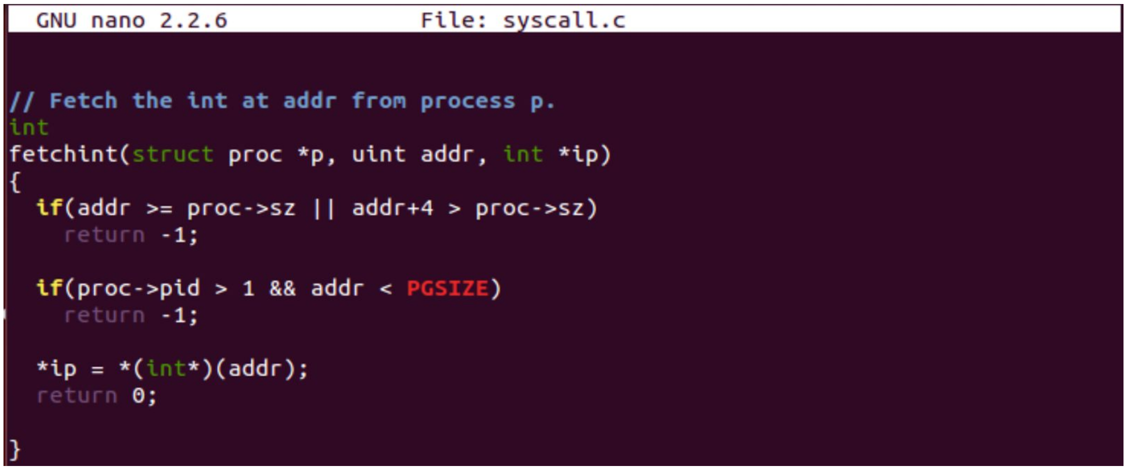
```
  ilock(ip);
  pgdir = 0;

  // Check ELF header
  if(readi(ip, (char*)&elf, 0, sizeof(elf)) < sizeof(elf))
    goto bad;
  if(elf.magic != ELF_MAGIC)
    goto bad;

  if((pgdir = setupkvm()) == 0)
    goto bad;

  // Load program into memory.
  sz = 4096;
  for(i=0, off=elf.phoff; i<elf.phnum; i++, off+=sizeof(ph)){
    if(readi(ip, (char*)&ph, off, sizeof(ph)) != sizeof(ph))
      goto bad;
```

//Image 6



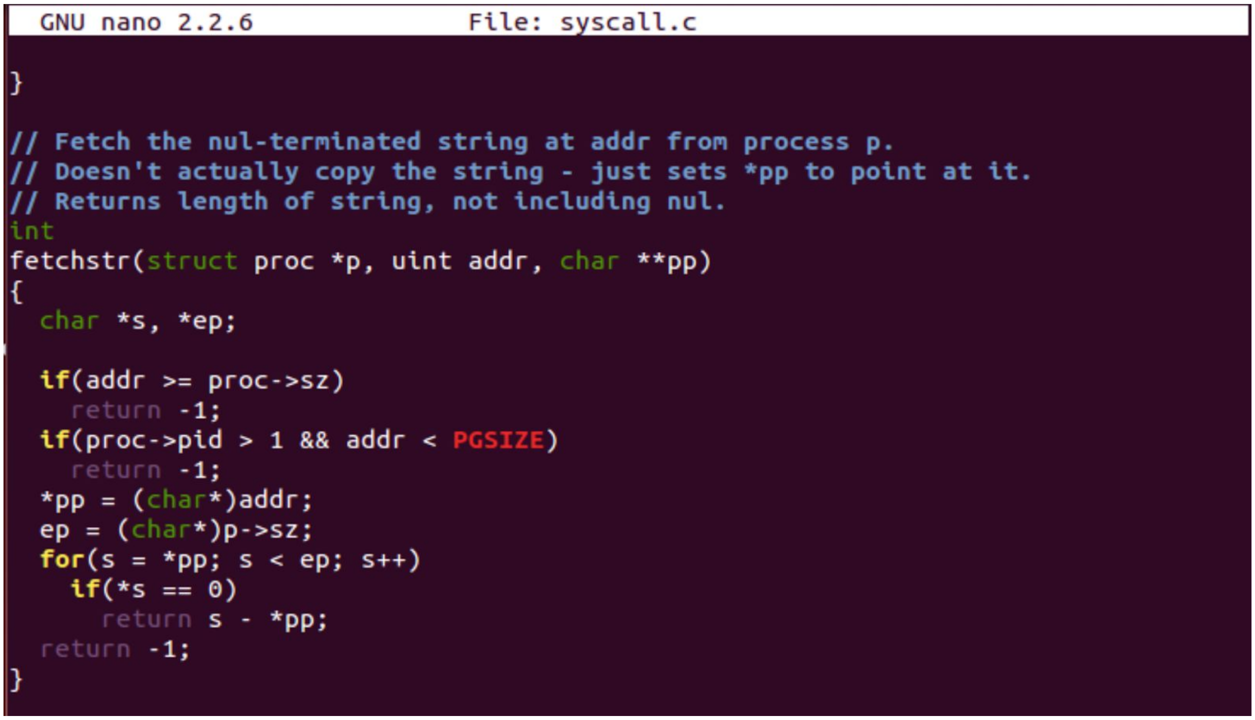
```
  GNU nano 2.2.6                  File: vm.c


// Given a parent process's page table, create a copy
// of it for a child.
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
  pde_t *d;
  pte_t *pte;
  uint pa, i;
  char *mem;

  if((d = setupkvm()) == 0)
    return 0;
  for(i = 4096; i < sz; i += PGSIZE){
    if((pte = walkpgdir(pgdir, (void*)i, 0)) == 0)
      panic("copyuvm: pte should exist");
    if(!(*pte & PTE_P))
```

//Image 7



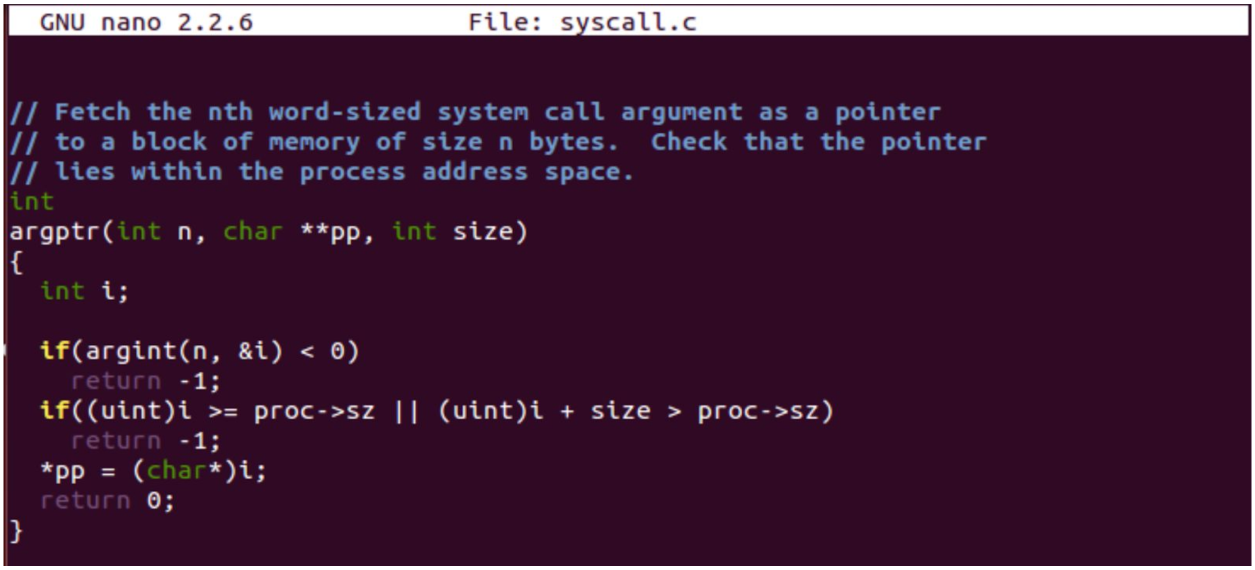
```
 GNU nano 2.2.6               File: syscall.c


// Fetch the int at addr from process p.
int
fetchint(struct proc *p, uint addr, int *ip)
{
  if(addr >= proc->sz || addr+4 > proc->sz)
    return -1;

  if(proc->pid > 1 && addr < PGSIZE)
    return -1;

  *ip = *(int*)(addr);
  return 0;

}
```

//Image 8



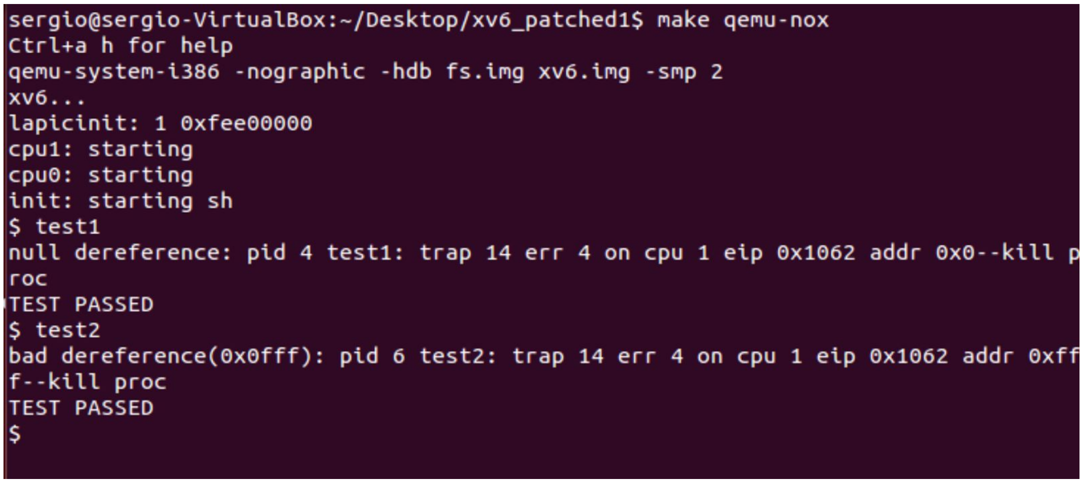
```
}

// Fetch the nul-terminated string at addr from process p.
// Doesn't actually copy the string - just sets *pp to point at it.
// Returns length of string, not including nul.
int
fetchstr(struct proc *p, uint addr, char **pp)
{
  char *s, *ep;

  if(addr >= proc->sz)
    return -1;
  if(proc->pid > 1 && addr < PGSIZE)
    return -1;
  *pp = (char*)addr;
  ep = (char*)p->sz;
  for(s = *pp; s < ep; s++)
    if(*s == 0)
      return s - *pp;
  return -1;
}
```

//Image 9



```
// Fetch the nth word-sized system call argument as a pointer
// to a block of memory of size n bytes.  Check that the pointer
// lies within the process address space.
int
argptr(int n, char **pp, int size)
{
  int i;

  if(argint(n, &i) < 0)
    return -1;
  if((uint)i >= proc->sz || (uint)i + size > proc->sz)
    return -1;
  *pp = (char*)i;
  return 0;
}
```

//Image 10



```
sergio@sergio-VirtualBox:~/Desktop/xv6_patched1$ make qemu-nox
Ctrl+a h for help
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2
xv6...
lapicinit: 1 0xfee00000
cpu1: starting
cpu0: starting
init: starting sh
$ test1
null dereference: pid 4 test1: trap 14 err 4 on cpu 1 eip 0x1062 addr 0x0--kill p
roc
TEST PASSED
$ test2
bad dereference(0x0fff): pid 6 test2: trap 14 err 4 on cpu 1 eip 0x1062 addr 0xff
f--kill proc
TEST PASSED
$
```

//Image 11