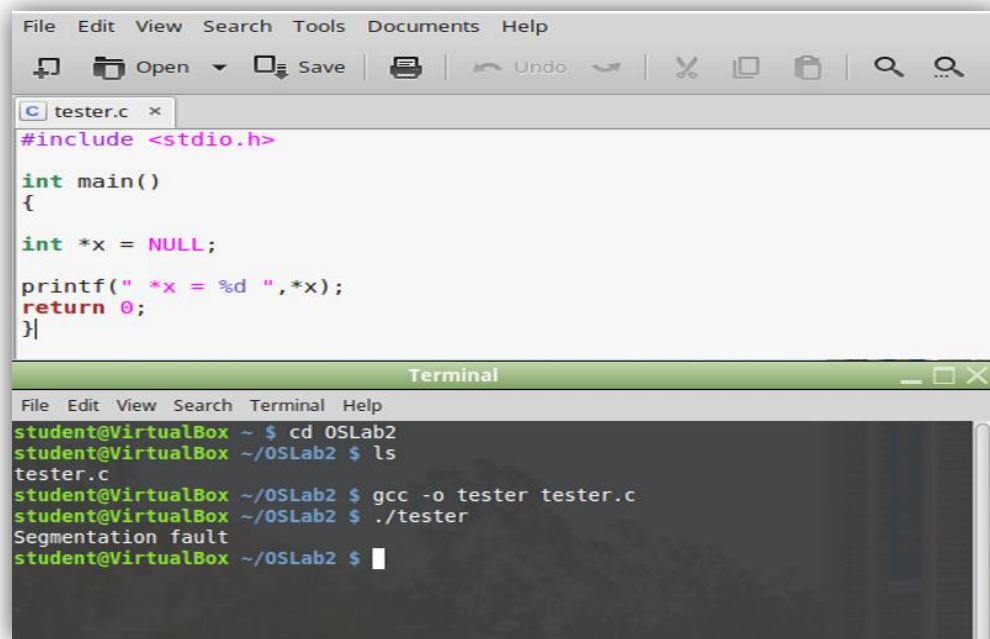


Lab - II

"The Null Pointer"

*Kiran Kumar Gangadevi**ID : 861242727*

Initially, to start off with the given task I've written a program "**tester.c**" which dereferences a Null pointer. I've then tried to run the program in Linux operation system. While running the program, the program was resulting in a Segmentation fault.



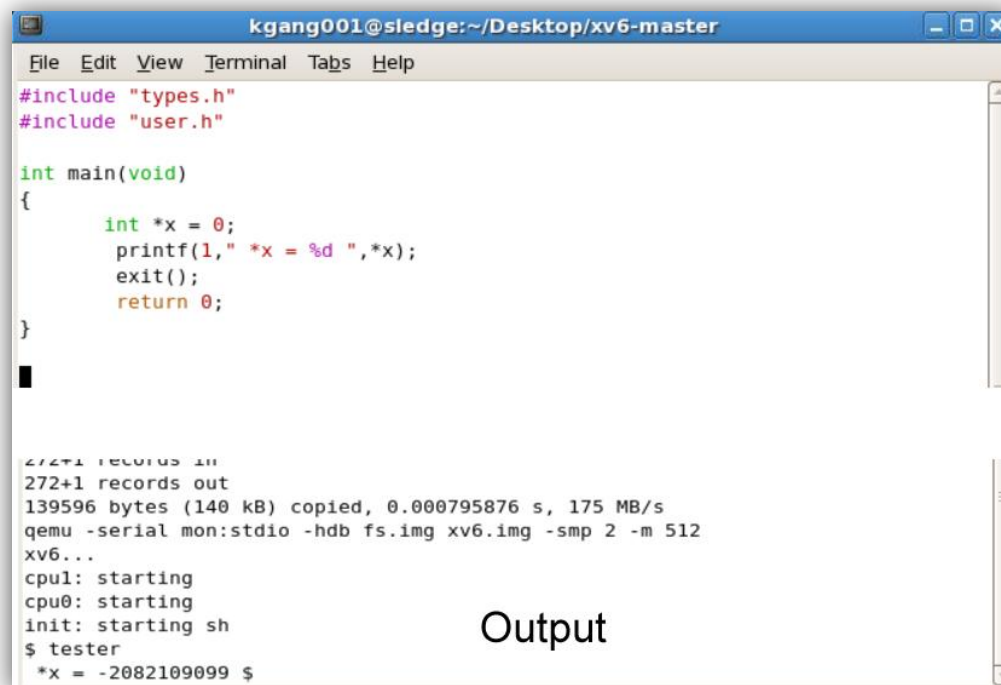
```
File Edit View Search Tools Documents Help
tester.c x
#include <stdio.h>

int main()
{
    int *x = NULL;
    printf(" *x = %d ", *x);
    return 0;
}

Terminal
File Edit View Search Terminal Help
student@VirtualBox ~ $ cd OSLab2
student@VirtualBox ~/OSLab2 $ ls
tester.c
student@VirtualBox ~/OSLab2 $ gcc -o tester tester.c
student@VirtualBox ~/OSLab2 $ ./tester
Segmentation fault
student@VirtualBox ~/OSLab2 $
```

A **segmentation fault**, or **segfault**, is a memory error in which a program tries to access a memory address that does not exist or the program does not have the rights to access. The reason for the error is because, in modern OSes, the page tables are usually set up to make the address 0 an invalid virtual address. Therefore dereferencing a Null pointer gives rise to a segmentation fault. Thus, the Linux operating system kernel responds by killing the program with SIGSEGV.

Later on, when the same program was run on XV6 operating system, we do not encounter a segmentation fault. This is because the XV6 operating system loads a program at address 0 unlike Linux.



```

kgang001@sledge:~/Desktop/xv6-master
File Edit View Terminal Tabs Help

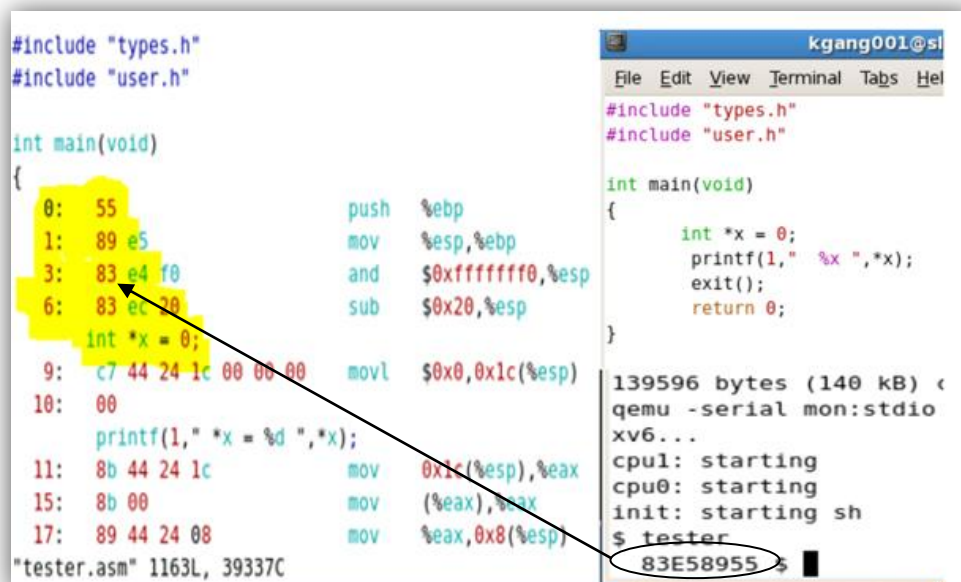
#include "types.h"
#include "user.h"

int main(void)
{
    int *x = 0;
    printf(1, " *x = %d ", *x);
    exit();
    return 0;
}

272+1 records in
272+1 records out
139596 bytes (140 kB) copied, 0.000795876 s, 175 MB/s
qemu -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ tester
*x = -2082109099 $
  
```

Output

As we can see below, when we examine the assembly code of the program (tester.asm), we can see the address at which the code is stored.



```

#include "types.h"
#include "user.h"

int main(void)
{
    0: 55          push    %ebp
    1: 89 e5      mov     %esp,%ebp
    3: 83 e4 f0    and     $0xfffffff0,%esp
    6: 83 ec 20    sub     $0x20,%esp
    int *x = 0;
    9: c7 44 24 1c 00 00 00 movl    $0x0,0x1c(%esp)
    10: 00
    printf(1, " *x = %d ", *x);
    11: 8b 44 24 1c  mov     0x1c(%esp),%eax
    15: 8b 00      mov     (%eax),%eax
    17: 89 44 24 08  mov     %eax,0x8(%esp)
    "tester.asm" 1163L, 39337C
}

139596 bytes (140 kB) c
qemu -serial mon:stdio
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ tester
83E58955 $
  
```

Hence, it is clear that the XV6 operating system loads the program starting at address 0.

The goal of the Lab 2 is to make sure that when a program is loaded, it is not to be loaded at address 0 and further more we need to make sure that if someone tries to access address 0, that's not in the page table so that you would get a page fault.

To achieve the goal, the following files have to be changed.

1. Makefile
2. exec.c
3. vm.c
4. syscall.c
5. tester.c (program to test the implementation)

Firstly, Makefile file has been changed in such a way that the user programs to start execution at page one rather than page zero.

```
ULIB = ulib.o usys.o printf.o umalloc.o

_%: %.o $(ULIB)
$(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o $@ $^
$(OBJDUMP) -S $@ > $.asm
$(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > $.sym
```

Secondly, the vm.c file has been changed to have the function copyuvm() copy 1 page beyond the start of a process page table in order to make sure that it does not have multiple pages of 0s.

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

Thirdly, the exec.c file has been changed to leave the page zero of memory and start loading the program at page one.

```
// Load program into memory.  
//sz = 0;  
// To Make Null Dereference Fail, We Leave the First Page Inaccessible  
sz = PGSIZE-1;
```

Finally, the syscall.c file has been changed to throw an error for null pointers being requested from the kernel.

```
int  
fetchstr(uint addr, char **pp)  
{  
    char *s, *ep;  
  
    if(addr >= proc->sz || addr == 0)  
        return -1;  
    *pp = (char*)addr;  
    ep = (char*)proc->sz;  
    for(s = *pp; s < ep; s++)  
  
int  
fetchint(uint addr, int *ip)  
{  
    if(addr >= proc->sz || addr+4 > proc->sz || addr == 0)  
        return -1;  
    *ip = *(int*)(addr);  
    return 0;  
}  
  
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 || (uint)i == 0)  
        return -1;  
    *pp = (char*)i;  
    return 0;  
}
```

I've written a test program **tester.c** to verify the correctness of the modifications made to the Virtual Memory features to xv6. Upon testing the program, the output is displayed below.

Output:

```
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ tester
pid 3 tester: trap 14 err 4 on cpu 1 eip 0x1015 addr 0x0--kill proc
$
```

When the program which dereferences a Null pointer was run, the xv6 operating system was successfully able to trap and kill the process.

The "**pid 3 tester: trap...**" message is from the kernel trap handler in trap.c. It has caught a page fault (trap 14, or T_PGFLT), which the xv6 kernel does not know how to handle. The "**addr 0x0**" indicates that the virtual address that caused the page fault is 0x0.

The output before the modification:

```
xv6...
cpu1: starting
cpu0: starting
Start Address =0 End Address =2364
init: starting sh
Start Address =0 End Address =5680
$
```

The output after the modification:

```
xv6...
cpu1: starting
cpu0: starting
Start Address =4095 End Address =6460
init: starting sh
Start Address =4095 End Address =9776
$ tester
Start Address =4095 End Address =6152
pid 3 tester: trap 14 err 4 on cpu 1 eip 0x1015 addr 0x0--kill proc
$
```

Clearly, we see that the program was prior being loaded at address 0 in xv6 operating system. After the modification was performed, we see that the starting address of the programs being loaded has been changed. To be more precise, the program is being loaded in page 1 instead of page 0. i.e 1 Page = 4096B

Thus, as required for Lab 2, when a user tries to access a null pointer in xv6 OS, the OS is able to trap and kill the process that tried to access the null pointer. Also, the xv6 virtual memory feature is modified such that the any program is now loaded at address 1 while it was supposed to be loading at address 0. I was able to successfully demonstrate it with the test program **“tester.c”**.