Lab 2, CS 202, Winter 2016

Ahmad Darki

adark001@ucr.edu

Kittipat Apicharttrisorn

kapic001@ucr.edu

February 23, 2016

The Null Pointer

The objective of this project is to learn more about Virtual Memory management in `xv6` as well as understanding the null pointer and dereferencing it.

Null Pointer Dereference

The definition of a null pointer is a pointer which points to a memory that does not exist, and dereferencing means using this null pointer to access a certain address that is not valid. This problem raises exceptions once it gets to the statement of *null pointer dereferencing*.

Having said this, the project description claims that the current `xv6` implementation available at https://github.com/guilleiguaran does not through any exception and the program runs normally. In order to see this we wrote a simple program called **`testnull.c`** which does the following:

```
#include "types.h"
#include "stat.h"
#include "user.h"
#define NULL ((void *)0)

int
main(int argc, char *argv[])
{
  //example taken from http://stackoverflow.com/questions/4007268/what-exactly-is-meant-by-de-
    referencing-a-null-pointer
  int a, b;
  int *pi;
  a = 5;
  pi = &a;
  a = *pi;
  pi = NULL;
  b = *pi;

  printf(1, "Null Pointer value: %p\n", *pi);
  exit();
}
```

testnull.c

And the output of this command is as followed:

```
cpu1: starting
cpu0: starting
init: starting sh
$ testnull
Null Pointer value: 83E58955
```

Null Pointer

As it appears from the program, there was no exception thrown and there has been an actual address returned.

Now we wish to see what is the layout of the filesystem in `xv6`. In order to do so, we used `superblock` in the `fs.c` file and printed out the layout of the filesystem. In this file there is a **`struct`** called `superblock`, as it has been commented in the `fs.c` file we get the following:

```
104   // The inodes are laid out sequentially on disk immediately after
105   // the superblock. Each inode has a number, indicating its
106   // position on the disk.
```

fs.c

By adding the following to the to the `fs.c` we can see the layout:

```
struct superblock sb;
void
iinit(void)
{
  initlock(&icache.lock, "icache");
```

```
  readsb(ROOTDEV, &sb);
  cprintf("superblock layout: size %d nblocks %d ninodes %d nlog %d\n",
                              sb.size, sb.nblocks, sb.ninodes, sb.nlog);
}
```

fs.c

However the output was not quite what we were expecting:

```
xv6...
cpu0: panic: iderw: ide disk 1 not present
 80102854 801001d9 80101301 801015e5 801034aa 0 0 0 0 0
```

Panic! output

As you can see there was a `panic` in the execution. As we dug deeper we learnt that in the file `ide.c` the panic was raised to the issue that the disk was not defined or we tried to get the output before it was constructed:

```
void
iderw(struct buf *b)
{
  ...
  if(b->dev != 0 && !havedisk1)
    panic("iderw: ide disk 1 not present");
  ...
}
```

ide.c

Now that we learnt what the issue was, we used the `dev` device that is being used as the main file system as the input of the function `void iinit(void)` in file `fs.c`:

```
...
struct superblock sb;
void
iinit(int dev)
{
  initlock(&icache.lock, "icache");
  readsb(dev, &sb);
  cprintf("superblock layout: size %d nblocks %d ninodes %d nlog %d\n",
                              sb.size, sb.nblocks, sb.ninodes, sb.nlog);
}
...
```

fs.c

Now that we made a change in this file, we shall continue changing the definition files and the other files that refer to the same function which are `defs.h` and `proc.c`:

```
...
void            iinit(int dev);
...
```

defs.h

```
...
void
forkret(void)
{
  ...
  if (first) {
    ...
    iinit(ROOTDEV);
    ...
  }
  ...
}
...
```

proc.c

Now that we made changes on `iinit`, we need to make the proper changes for the `log.c` functions too, and of course we need to make the proper changes on the definition files too.

After applying the proper changes, now we are getting the following output as the `FS` layout:

```
xv6...
cpu1: starting
cpu0: starting
superblock layout: size 1024 nblocks 985 ninodes 200 nlog 10
init: starting sh
$
```

File System Layout

Now we have the whole idea of how `xv6` defines its `file system` layout. It is now the time to handle the null pointer dereferencing problem.

Fixing The Problem

The solution comes from the idea of loading the program into the memory not from the address `0` but from the next page which is in fact address `4096` that is `0x1000`. Therefore the paging is now shifted by one. In this section we continue elaborating on the implementation.

In order to understand the paging of the system and what are the important numbers for addresses and memory layout, we opened the file `mmu.h` which is a library that defines the memory management unit for the `xv6`. We get the following from this file:

```
// Page directory and page table constants.
#define NPDENTRIES      1024    // # directory entries per page directory
#define NPTENTRIES      1024    // # PTEs per page table
#define PGSIZE          4096    // bytes mapped by a page
```

mmu.h memory layout

As it indicates above the size for each page is `4096 B`. Now we are ready to shift the memory mapping by one page. The first file that needs to be changed is the `Makefile`. In this file we can apply the following change:

```
...
138   $(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o $@ $^
...
```

Makefile changed

This change basically forces the program to load into the memory not from the address `0` but from the address `0x1000` which is essentially the size of a page. Now that we have this new layout, it is time to see what to do with the processes and loading them into the memory. In the file `exec.c` we have the following part:

```

exec(char *path, char **argv)
{
  ...
  // Load program into memory.
  sz = 0;
  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;
    ...
 }
```

exec.c

As it shows in the function `exec` we have a variable `sz` which basically indicates the beginning of the memory for the program. We need to change this to the first address of the next **page**. In order to do so we have applied the following change using the definition in the file `mmu.h` for the page size that is `PGSIZE`:

```

exec(char *path, char **argv)
{
  ...
  // Load program into memory.
  sz = PGSIZE;
  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;
    ...
 }
```

exec.c changed to load the memory after one page

Now we apply the same change to the function `copyuvm` which basically creates a copy of the parent for a child process:

```
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
  ...
319   for(i = PGSIZE; i < sz; i += PGSIZE){
  ...
}
```

vm.c

After applying these changes it is time to see what the original `testnull.c` program outputs:

```
xv6...
cpu1: starting
cpu0: starting
superblock layout: size 1024 nblocks 985 ninodes 200 nlog 10
init: starting sh
$ testnull
pid 3 testnull: trap 14 err 4 on cpu 1 eip 0x102f addr 0x0--kill proc
$
```

testnull output

As it shows above, the program did not execute and basically an exception was thrown since the address was not valid. This concludes our solution to this null pointer dereferencing issue.

However we continued this project and extended our solution in the execution of a process. In this solution we wanted to make sure that the process will not violate its address space, therefore int the file `syscall.c` we added a new condition that makes sure the address was not violated:

```
...
// Fetch the int at addr from the current process.
int
fetchint(uint addr, int *ip)
{
  if(addr >= proc->sz || addr+4 > proc->sz)
    return -1;
  if (addr < PGSIZE) {
    return -1;
  }
  *ip = *(int*)(addr);
  return 0;
}

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

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

syscall.c