Mini Project 3: Memory - Report for Rainbow Chickens

Members: Nathan Cader, Benjamin Voor, Aidan Guenther, Khalee Johnson

COP4610.03I&T

December 1, 2025

Instructor: Dr. Luis G. Jaimes

COP4610 Operating Systems Concepts
Fall 2025

Introduction

This project extends the xv6 operating system to correctly handle null pointer dereferences.

In the original xv6, the first page of memory is mapped and accessible, which means address 0x0 does not trap as expected. Modern operating systems always mark the first page invalid to prevent this behavior.

The objective of this project is to modify xv6 so:

- The null page (0x0000–0x0FFF) is always unmapped.
- Any attempt to dereference a null pointer results in a trap and process termination.
- System calls properly validate user pointers before accessing them.

These changes are verified using the provided test program test1-test2-test3.c

Files Modified

kernel/exec.c:

- line 35: Changed `sz = 0;` to `sz = PGSIZE;`

kernel/makefile.mk:

- line 126: Changed `...=text=0x0` to `...text=0x1000`

kernel/proc.c:

- line 87: Changed `p->sz = PGSIZE;` to `p-sz = 2*PGSIZE;`
- line 94: Changed `p->tf->esp = PGSIZE;` to `p->tf->esp = 2*PGSIZE;`
- line 95: Changed `p->tf->eip = 0;` to `p->tf->eip = PGSIZE;`

kernel/vm.c:

- line 198: Changed `mappages(pgdir, 0...` to `mappages(pgdir, (void*)PGSIZE...`
- line 309: Changed `i = 0` to `i = PGSIZE`

user/makefile.mk:

- line 77: Changed memory location to 0x1000 (4096)

user/test1-test2-test3.c:

- Uncommented test3 to get it ready for test case 3.

Test case 3

kernel/syscall.c:

- Line 22: Added *if(p->pid > 1 && addr < 4096)*
- Line 23: Added return -1;
- Line 39: Added *if(p->pid > 1 && addr < 4096)*
- Line 40: Added return -1;
- Line 70: Added *if((unit)i < 4096 || (uint)i >=USERTOP)*
- Line 71: Added return -1;

user/test1-test2-test3.c:

- Uncommented bounds()

Implementation Details

kernel/exec.c:

- By starting at PGSIZE (0x1000) instead of 0x0, the first page is left unmapped. This makes a protected zone, where any access to the address 0-4095 will cause a page fault and kill the process, which catches null pointer bugs immediately.

kernel/makefile.mk:

- The linker needs to know where the code will run so it can generate the correct addresses for jumps and function calls. Since initcode is loaded at 0x1000, the linker needs to compile it to run at 0x1000. Otherwise, wrong addresses result in a crash.

kernel/proc.c:

- This tells the kernel that the address space ranges from 0 to 2*PGSIZE. The address space still spans 0-0x2000 even though the null page is unmapped.

- ESP is the stack pointer and the stack grows downwards from higher to lower addresses. Thus, the stack needs to start at the top of the non-null page.

- EIP is the instruction pointer and tells the CPU where to begin executing code. Since the initcode starts at 0x1000 now, it must point to this.

kernel/vm.c:

- The second parameter (the one that was changed) is the virtual address being mapped. By using PGSIZE instead of 0, we map the second page to physical RAM, leaving the null page unmapped. This creates the protection, as no mapping results in a page fault on access.

- When fork() creates a child process, copyuvm copies all the parent's memory pages. Since the null page doesn't exist (it's unmapped), there is nothing to copy. Starting at 0 would access the null page and cause a panic, while starting at PGSIZE skips the null page and copies only the mapped pages.

user/makefile.mk:

- The linker must know where the code will run. Since exec.c loads user programs at 0x1000, the linker must compile them for 0x1000.

user/test1-test2-test3.c:

- Temporarily commented bounds() to ensure the first two tests are successful.

Part 3

kernel/syscall.c:

- On lines 22 and 39 in fetchint & fetchstr, the problem was that the addresses inside the first 4 KB (the null page) are never valid user data. Real programs don't keep data there; the OS intentionally marks it invalid. The program fixes this by stopping bad pointers (cases a & b) before xv6 ever dereferences them.

- On line 70 in argptr USERTOP marks the boundary between legal user memory and the code/kernel region. If a pointer touches that boundary, it means: it points into the code segment, or its size might spill into it. The third bad pointer starts inside valid stack

memory, but its length goes past the legal range. The program fixes this by checking blocks the entire scenario.

Testing results

```
$ test1-test2-test3
null dereference: pid 5 test1-test2-tes: trap 14 err 4 on cpu 1 eip 0x10a6 addr 0x0--kill proc
TEST PASSED!
bad dereference (0x0fff): pid 6 test1-test2-tes: trap 14 err 4 on cpu 1 eip 0x114b addr 0xfff--kill proc
TEST PASSED!
TEST PASSED!
```

Conclusion

This project required modifying xv6 to invalidate the null page, update memory layout loading logic, and strengthen pointer validation inside system calls.

With these changes, xv6 works like a modern OS when there's null pointer dereferences and incorrect syscall pointers.

All three test cases from the provided test program were successfully passed. Screenshots confirming the results are included in the testing section.