

# P2 - Add Guard Page for NULL Dereference to xv6

---

## Summary

You're going to modify how **xv6** lays out process memory! Specifically, you are going to add a guard page in the first 4K of memory (starting at address 0 of process memory). The utility of this is that programs that dereference a null pointer will crash.

I provide the test program for user land. You must add it to your **makefile**.

## Current memory layout

**xv6** currently lays out process memory like this (starting at address 0 at top):

### Contents

---

code

---

stack

---

heap

Your job is to make programs start beyond address zero (one page to be exact) so that a dereference of any address in the first page of memory will result in instadeath.

The new layout should be like this:

Like this:

### Contents

---

guard page

---

code

---

stack

---

heap

where the code starts at the first page boundary, not on the zeroth page.

## Suggested steps

### 1. Make a copy of XV6

Do not defile the virgin.

### 2. Test program

This is the test program called **guard**.

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

int main(int argc, char ** argv) {
    unsigned char * p = (unsigned char *) 1024;
    unsigned char c;
    printf(1, "Reading from address 0x%x value: 0x%x\n", p, (unsigned int)
(c = *p));
    printf(1, "Writing to address 0x%x\n", p);
    *p = c;
    printf(1, "Reading from address 0x%x value: 0x%x\n", p, (unsigned int)
*p);
    exit();
}
```

Add this program to xv6 (via the makefile) and build the system. The program will not crash in an unmodified **xv6**.

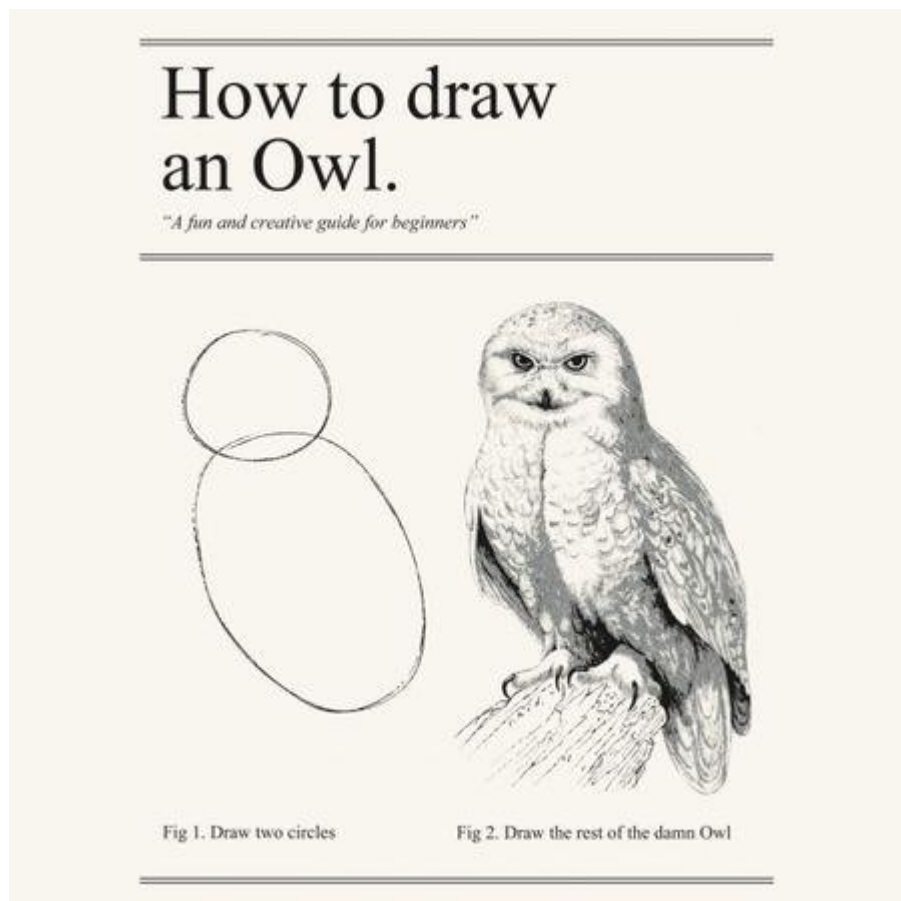
It should crash, producing this:

```
$ guard
pid 3 guard: trap 14 err 4 on cpu 1 eip 0x1013 addr 0x400--kill proc
$
```

Again, if **guard** does NOT crash as above, you're project isn't working.

Notice this user land **printf** is a little different from the one provided in **C** in that the first argument to the **xv6 printf** is the file descriptor to use for the output. The normal **printf** does not have this first parameter.

Draw the rest of the owl



### Hint 1

Look at how `exec()` works to better understand how address spaces get filled with code (loaded from disk) and initialized.

Also look at `fork()`. To create a duplicate address space, it has to copy the old one.

### Hint 2

`makefile`. There is something you need to change in the `makefile` beyond simply adding `guard` to the list of user programs to build.

### Hint 3

If you change or add more than 30-ish bytes in the WHOLE project (other than adding the name of your partner), you're doing it wrong. That's right. This project is just a few bytes. The trick is knowing which few bytes.

### Fun

As soon as you begin making changes, attempting to test can result in an OS crash.

There is a *limited* form of `printf` available to help debugging from inside `xv6`. It is called `cprintf`.

### What to hand in

Zip your cleaned XV6 tree and submit the zip file.

Only one partner submits code. The other partner submits the same text file added to XV6.

## Partner rules

Work in pairs.