CS 416 - Operating Systems Design

Homework 3
Virtual Memory in xv6

April 17, 2015
Due TBA at 11:59 PM

1 Introduction

In this homework, you will explore the xv6 virtual memory systems, as well as expand the virtual memory-based features of the operating system. In particular, you will implement memory protection, copy-on-write, and a limited type of demand-based memory allocation.

2 Implementing Memory Protection

2.1 Requirements

Implement an **mprotect**() system call in xv6 similar to the one available on Linux. The method signature should be as follows (*note that **addr** *must* be aligned to a page boundary*):

```
    int mprotect(void *addr, int len, int prot)
```

You should implement support for the following protection levels (there is no need to implement **PROT_EXEC**).

- **PROT_NONE** - The memory cannot be accessed at all.
- **PROT_READ** - The memory can be read.
- **PROT_WRITE** - The memory can be modified.

See the Linux man page for the details of how **mprotect()** should behave.
When a process violates memory protection a signal called **SIGSEGV** should occur, and a structure **siginfo_t** containing information about the nature of the protection fault (the address and type of exception) should be passed to the signal handler as a second argument.
The **siginfo_t** should be defined as follows:

```
typedef struct
{
    uint addr; // Should be an address.
    uint type; // Should be a protection level.
} siginfo_t;
```

You should update the signal handling code so that the signal handlers must adhere to the following prototype:

```
    void handler(int signum, siginfo_t info);
```

For the preexisting exception (**SIGFPE**), *info* can simply contain zeroes for both values, as we can consider the values in *info* undefined for that signal.

2.1.1 Signal Starting Point

The starting branch will contain a fully functional implemenation of the signal facility from Homework 1, in order to let you focus on the memory protection aspects, rather than the signal facility.

2.2 Testing

The starting branch will contain a test program called **test_mprotect** that will test your new memory protection facility.
The program **test_mprotect** will do the following:

- Allocate a page.
- Mark a page read only using **mprotect()**
- Attempt to write to the page.
- Handle **SIGSEGV**.
- Use **mprotect()** inside the handler to mark the page read-write.
- Print all addresses and the status at each step.

3 Implementing Copy On Write

3.1 Requirements

Implement copy-on-write for process images. When **fork()** occurs in the normal case, the process image is copied to the new child process before the child starts executing. You should implement a new system call **cowfork()** that functions identically to the normal **fork()**, except that instead of copying the process image, the mappings should instead be duplicated and marked read-only in both processes. To keep track of this, the kernel should be able to track the number of these read-only mappings to each page. When a write is attempted into any read-only page, the page should be copied into a new read-write page, the mapping updated, and the counter decremented on the original page. You will need to be very careful on **exec()** and **exit()**, when all of the mappings are destroyed and the and counters decremented.

3.2 Testing

The starting branch will contain a test program called **test_cow** to test your copy on write facility.
The program **test_cow** will do the following:

- Make a large number of calls to **fork()** and calculate the average time for each call.
- Make the same number of calls to **cowfork()**, and calculate the average time for each call.
- Cleans up the child processes after the test.

4 Implementing Demand Heap Allocation

Implement on-demand memory allocation for the heap. Currently, heap allocations are done immediately when the allocation is requested (with **malloc()**). Internally, **malloc()** operates by calling the system call **sbrk()**. You should modify the operating system to instead map and return an address to a page in response to an allocation request, but only actually allocate the memory for the page upon the first usage of that memory. You should implement this functionality as a new system call **dsbrk()**.

*This part is **required** for CS518, **extra credit** for CS416.*

4.1 Testing

The starting branch will contain a test program **test_demandalloc** that does the following:

- Perform a number of allocations using **sbrk()** and write 4096 bytes into each page.
- Perform the same number of allocations with **dsbrk()** and write 4096 bytes into each page.
- Time how long the allocation phase vs. the writing phase takes for each of these cases.

*The **test_demandalloc** program will be commented out of the Makefile by default in the starting branch. If you do this section, you **must** uncomment this line in UPROGS.*

5 Source Control

To start on this project, use the following commands in your xv6 git repository (assuming you have checked it out as instructed in the document uploaded to Sakai):

```
git fetch origin
git checkout sp15-hw3-start
```

After checking out, use the following commands to create your working branch:

```
git branch sp15-hw3
git checkout sp15-hw3
```

6 Submission

To submit your work, please follow the following instructions:
Type the following (for each netids field, use both partner's netids, hyphenated like **NETID1-NETID2**):

```
git checkout sp15-hw3
make submit netids=<NETIDS> name=hw3 base=sp15-hw3-start
```

After this is complete, you should have a .tar.gz file containing individual patch file(s) for each commit. Ensure that the tar file actually contains the patches, then submit this tarball on Sakai.

Only ONE student from each group should commit, but you NEED to make sure you give the netids of both partners.

7 Overall Requirements

- The code has to be written in **C** language. You should discuss on Piazza if you think inline **asm** is necessary to accomplish something.
- Do not copy the solution from other students. Use Piazza to discuss ideas of how to implement it. Do not post your solution there. Use Sakai to submit it.
- Commit in your local git for each step of your solution to make sure you can isolate each step later in case you need to refer back.
- Submit a report on **Sakai** named **report.pdf**, detailing what you accomplished for this project, including issues you encountered. This report *must* be named **report.pdf** and *must* be in PDF format.