Pintos Project 3
Virtual Memory

October 26, 2016
Background
Definitions First

**Virtual Address**: “Physical Address” in the eyes of a process

**Page Fault**: Process accessing memory not allocated to it

**Page**: A section of virtual memory

**Page Table**: A table of pages each process has that keeps track of which pages have been allocated to it

**Frame**: A section of physical memory

**Frame Table**: Stores information about the frames you have allocated from the cache.

**Swap**: Writing / reading between the cache and the disk

**Swap Table**: Keeps track of which pages are written in which section of disk.
Overview
What is virtual memory?

Gives the illusion that every process has the entire cache to itself:

Thread 1:
- 0x80: Data 1a
- 0x84: Data 1b

Thread 2:
- 0x80: Data 2a
- 0x84: Data 2b

Physical Memory:

<table>
<thead>
<tr>
<th>Data 1a</th>
<th>Data 1b</th>
<th>Data 2a</th>
<th>Data 2b</th>
</tr>
</thead>
</table>
Overview of Part 1: Stack (memory) Growing

Hey I want this vaddr, but I don’t have the page to store it. Can I have page?

Ok, here you go
Overview of Part 2: Cache Eviction

Hey I want more memory

Ok, here you go (you greedy pig)

Wait I have no more space, let me make some space.

User

Kernel

Page Table
- Page 1
- Page 2
- Page 3
- NULL

Cache
- USED->FREE
- USED
- USED
- USED

Write to disk
Overview of Part 2: Cache Eviction

Where’s my memory?

Kernel

Cache
FREE->USED
USED
USED
USED

Page Table
Page 1
Page 2
Page 3
NULL

User

Here you go

New page
Overview of Part 2: Cache Eviction

I need the vaddr located in page 2

Oh oops, I just evicted that from cache
Let me get that back for you

Page Table

- Page 1
- Page 2
- NULL
- NULL

Cache

- USED
- USED->FREE
- USED
- USED

Write to disk
Overview of Part 2: Cache Eviction

Where’s my data man?

User 2

Kernel

Cache

USED
FREE->User 2’s Page 2
USED
USED

Page Table

Page 1
Page 2
NULL
NULL

Here you go

Read from disk
Overview of Part 3: Memory Mapping

I’m gonna load in this huge-ass file, it’s gonna take like 20 pages of memory

Ok. I’ll give you your memory (I’ll pretend to since you’re a greedy pig, you’ll have “memory” but you won’t get it until you ask for it)
Hey can I get the vaddr of File_page 2?

Sure I have it for you! (wait I actually don’t, but let me go and allocate memory for it now)
Requirements

Implementation
Project 3 will be done in `src/vm/
  This means you will run `make` in `src/vm`
  This means you will run tests in `vm/build`

DesignDoc
Project 3 design document can be found in `doc/vm.tmpl`
Rename the design document as `DESIGNDOC` and place in `src/vm`.

Naming
Please name your project 3 directory “`proj3`” in your repository. Also please name your design document “`DESIGNDOC`”

Due November 11th
Overview

This project aims for you to implement the necessary features for processes’ virtual memory to grow.

Part 1: Growing the Stack
In userprog/exception.c, and userprog/process.c support the feature when a page fault happens, it installs the page if able to be installed.

Part 2: Cache Eviction
In userprog/exception.c, when no more pages are available from the cache, write to the disk with swap-tables

Part 3: Memory Mapping
Support the memory mapping syscalls in userprog/syscall.c
Prerequisites

Setup proj3 Directory
Make a copy of your proj2 directory and name it proj3.

Of the 110 tests for project 3, 76 of those tests are the exact same as the tests in project 2.

Add New Files to Makefile.build
Under the appropriate section add the paths to your new .c files.
Introduction
Virtual Address vs Physical Address

Recall that virtual address is the processes’ addresses that they use. Physical memory is the actual memory in the hardware.

You will have to do all the book-keeping to keep track of which physical memory is mapped to which processes’ virtual memory. This happens when you map a memory from physical to virtual.
When do I map memory for processes?
Page Faults

What’s a Page Fault?

When a process is trying to access virtual memory that does not have a page for (in the page directory), the kernel will give a page fault.

What do I do with a Page Fault?

In `userprog/exception.c`, there is the `page_fault` function. In this function you will look at the pointer that caused the page fault, and resolve it if possible.

For example, if the pointer is in kernel memory, you want to kill the process to protect the OS. If the pointer is not harmful, you will want to allocate that page to the page table.
Part 1
Growing the Stack
Detecting when to Grow a Stack

You will determine whether or not to grow the stack (allocating a new page) in userprog/exception.c

You get to decide in which file you want to handle allocating and installing a new page:
   - You can get a new frame with `palloc_get_page`
   - You install this frame into a page table with `pagedir_set_page`

Remember, every time you allocate a new page, you HAVE TO keep track of this new page with some kind of bookkeeping method

We highly suggest you to create a Supplementary Page Table and a Frame Table
What are Frame & Frame Tables?

A frame is a section of physical memory. The cache is basically a lot of frames

This means you will be mapping a page to a frame
What are Frame & Frame Tables?

You will need a frame table (collection of frames) to keep track of all the frames you have allocated from the cache.

You should keep track of who owns this frame, the pointer to the frame, and which page this frame is mapped to.

You can do this with a global list of frame table entries.

Wait, I need to keep track of pages?

In addition to keeping track of frames, it is also a good idea to keep track of all the pages (virtual memory) you have allocated. This can be done with a Supplementary Page Table.
What are Supplementary Page Tables?

It is a good idea to keep track of all the pages you have allocated.

This means information about the pointer to the virtual address associated with this page, the owner of this page, and the corresponding frame mapped to this page.

You might also want to keep track of its dirty bit, access time, and other information you want to consider.

Each thread should have its own supplementary page table.

You can do this either with a list of supplemental page table entries, or with a hash table (much faster). A hash table works very well because each vaddr entry of the table is a unique for each process.
I’m Confused

It’s ok, you’re not alone. Here’s a summary:

A process requests a vaddr that has not been allocated memory yet (page fault)

If the vaddr is a bad address, kill the process
If the vaddr is a good address, you can grow the stack

1. Get a new frame from the cache \( (\text{palloc\_get\_page}) \)
2. Install this frame into the page table with the requested vaddr \( (\text{pagedir\_set\_page}) \)
3. Bookkeep important information on the newly allocated frame and page
Part 2
Cache Eviction
When to Evict?

If you request a new frame, and there are no more frames left: 
\[ \text{palloc\_get\_page returns a NULL} \]

You need to free memory from the cache. This is eviction.

How do I evict a frame?

Write the information of the page mapped by the frame into the disk. Once you do this, you can clear the frame with \[ \text{pagedir\_clear\_page and realloc\_locate the frame to another page with palloc\_get\_page again.} \]

Make sure you keep track of which pages are written to the disk. This is called a swap table.
Another table…? Yes, the Swap Table.

Yes, in order to keep track of which pages are written to which sections of disk in order to retrieve the information later.

This can be done with a global bitmap.

Swapping can be done with a block device *(devices/block.h)*
Each sector of the disk is 512 bytes

You need to keep track of which sectors are free and which are not *(use a <bitmap.h>)*

If you use a bitmap, you don’t need swap table entires, you can just initialize a swap table and use the bitmap for all your bookkeeping.
I’m confused again

It’s ok, here’s another summary:

Evicting to Disk
During a page fault, if palloc_get_page returns a NULL, it means the cache is full.

Evict a frame from the cache
Reallocate the frame to the requested vaddr.

Reading from Disk
During a page fault, if the page has been allocated but cannot be found, you need to find it in the disk.

Evict a frame (least recently used is best) from the cache
Read the needed frame back into the cache. Free this spot in disk.
Part 3
Memory Mapping
Memory Mapping Syscalls

You need to support two new syscalls:

```c
mapid_t mmap (int fd, void* addr)
void munmap (mapid_t mapping)
```

Your task is to keep track of the memory used by memory mapped files.

If a page fault happens and the vaddr is a memory mapped address, allocate the file memory instead of just allocating a new frame.

You also need to make sure that memory mapped files don’t overlap other segments.

Hint: You need to allocate a lot of supplementary page tables entries.
Questions & Concerns?