Project 2
CSE 421/521 - Operating Systems
Virtual Memory in Pintos

Deadline: December 4, 2013 @ 11.59pm

Muhammed Fatih Bulut

Preparation

- Chapters 8-9 from Silberschatz.
- Lecture slides on *Memory* and *Virtual Memory*
- Pintos introduction
  - http://www.stanford.edu/~ouster/cgi-bin/cs140-winter13/pintos/pintos_1.html
- Pintos Reference Guides
  - http://www.stanford.edu/~ouster/cgi-bin/cs140-winter13/pintos/pintos_6.html

What is Pintos?

- Pintos is a simple OS for 80x86 architecture developed at Stanford University.
- It supports kernel threads, loading and running user programs, and a file system.
- Some parts left unimplemented for instructional purpose, such as virtual memory.

Programming Task

- Implement the virtual memory component in the Pintos operating system.

Goals!

- You'll have a better understanding of:
  - Paging
  - Page replacement
  - Other virtual memory concepts
  - How an operating system works
  - Yet, lots of C programming. Yay!!!

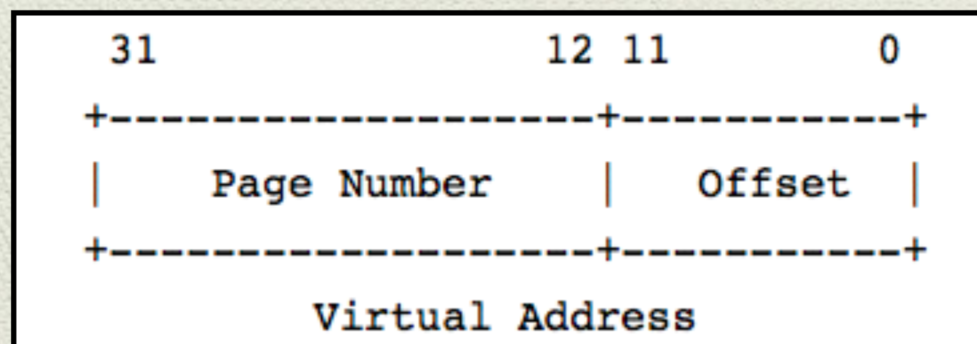
Setting up The Pintos Environment

- Login to timberlake or any other cse servers.
  - ssh [user_name]@timberlake.cse.buffalo.edu
- Copy Pintos source to your home dir.
  - cp /web/faculty/tkosar/cse421-521/projects/project-2/pintos.tar
- Create a dir called pintos and extract to it.
  - tar -xvf pintos.tar
- Grab Bosch simulator and run Pintos on it.
  - cp /web/faculty/tkosar/cse421-521/projects/project-2/boschs-2.6.2.tar

Background

Pages

- A continuous region of virtual memory.
- Page size = 4096 bytes = 12 bits.
- Must be page-aligned.
- In 32-bit virtual address, 20-bit page #, 12-bit page offset

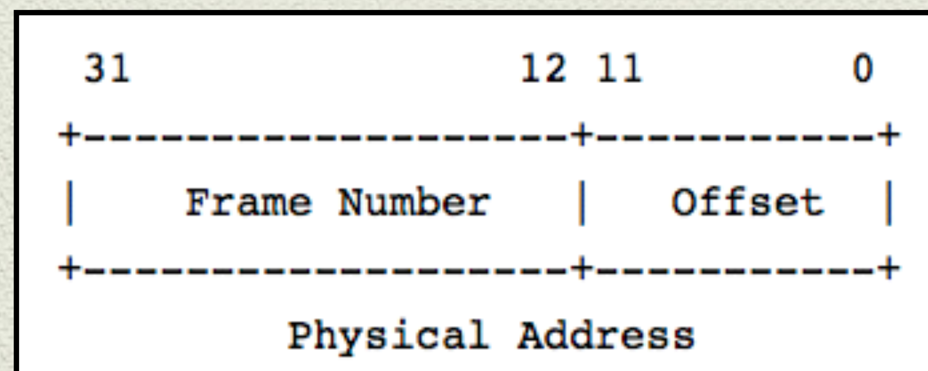


Functions

- Pintos provides various functions to work on virtual addresses.
  - **unsigned pg_ofs (const void *va):** Extracts and returns the page offset in virtual address va.
  - **uintptr_t pg_no (const void *va):** Extracts and returns the page number in virtual address va.
- See Pintos references for more details (A.6 Virtual Addresses).

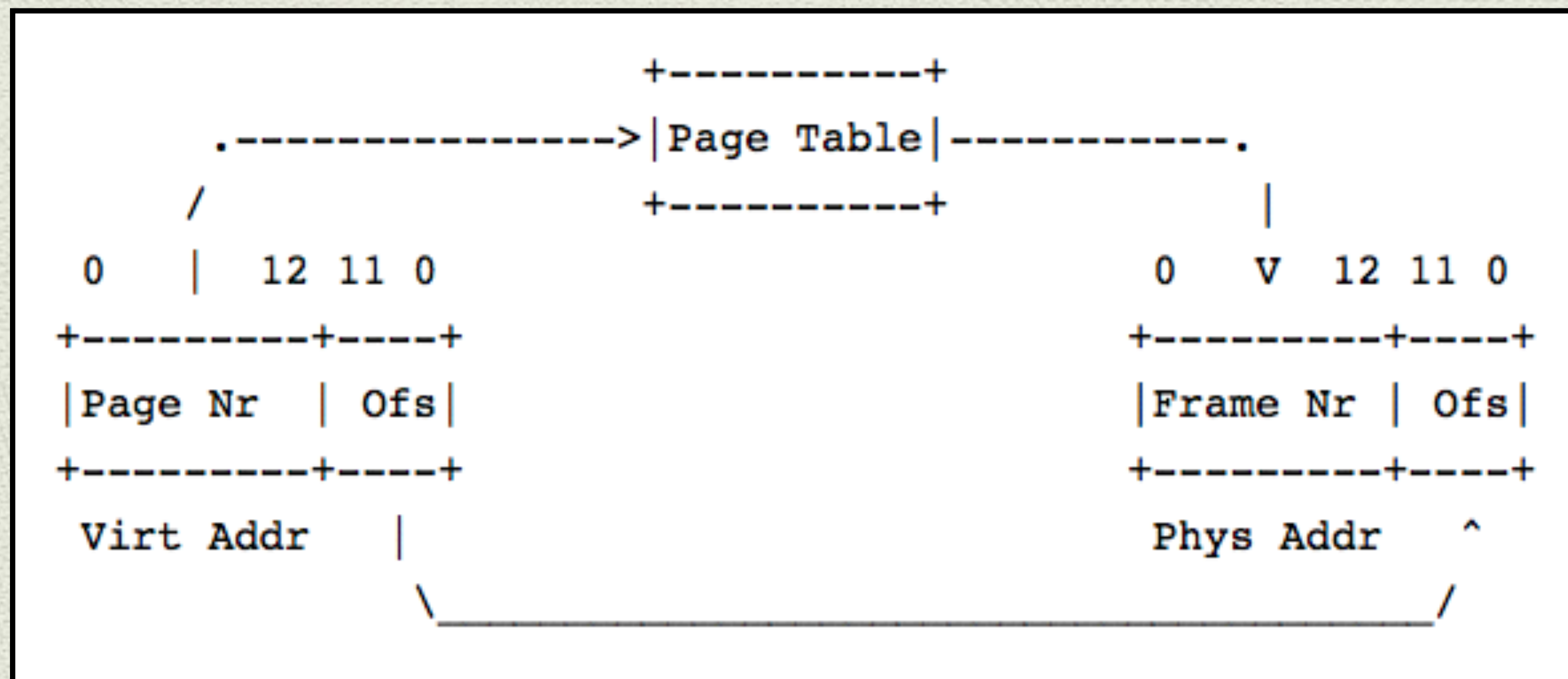
Frames

- Frame is a continuous region of physical memory.
- Must be in same page-size and page-aligned.
- 32-bit physical address = 20-bit frame # + 12-bit frame offset.



Page Tables

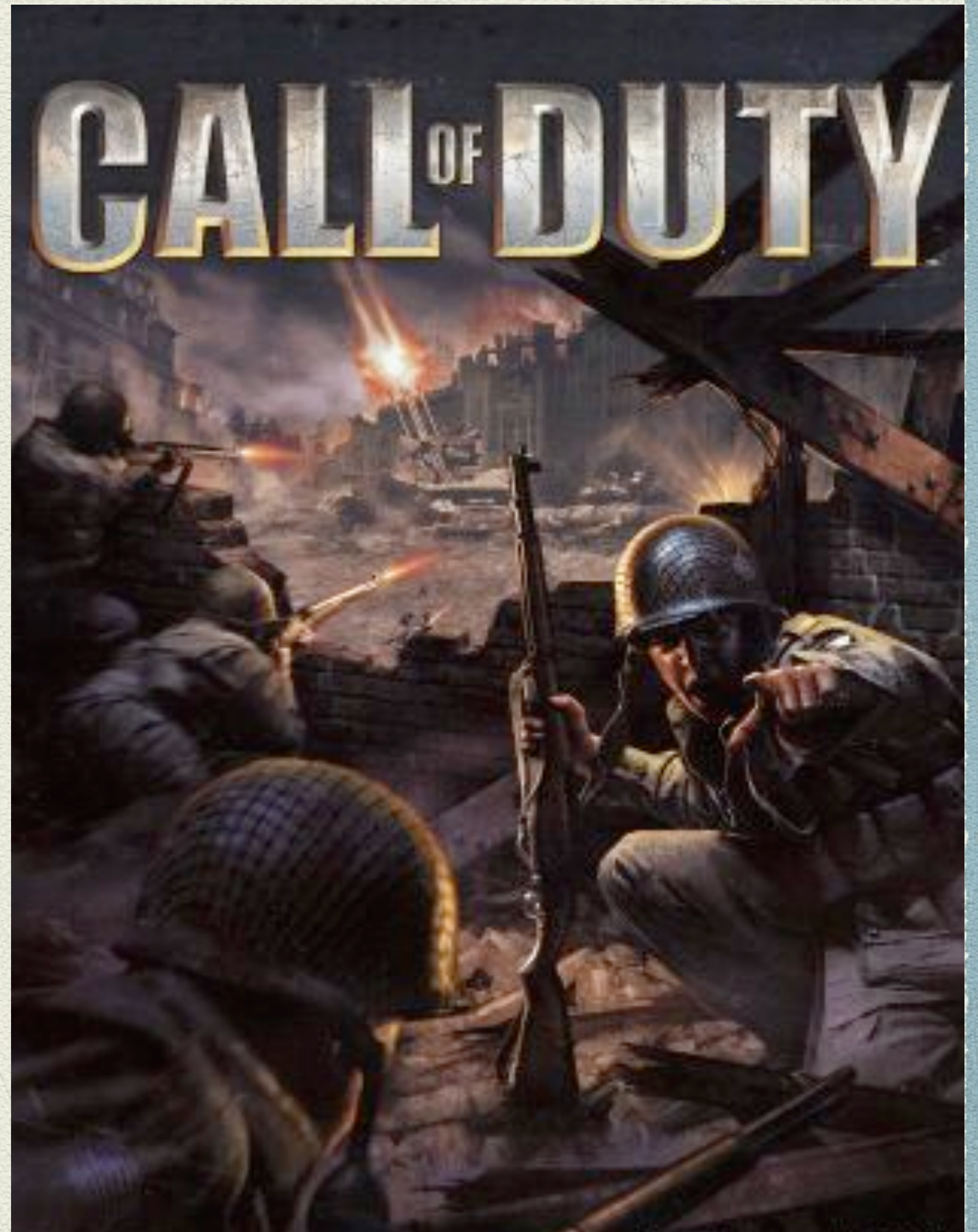
- Page table is a data structure that the CPU uses to translate a virtual address to a physical address (page -> frame).



Swap slots

- Swap slot is a continuous, page-size region of disk space in the swap partition.
- When a frame is evicted from the memory, it is written to swap page.

Your duty!



Tasks

- You will need to design the following data structures
  - Supplemental page table
  - Frame table
  - Swap table
  - Table of file mappings

Managing the Supplemental Page Table (SPT)

- It's supplemental to the page table. It's not the page table itself!
- It's needed because of the limitations imposed by the page table's format (See A.7 Page Table).
- The most important user of SPT is the page fault handler (*see "page_fault()" in "userprog/exception.c"*).

Supplemental Page Table (Cont.)

- Page fault handler should do:
  - Locate the page faulted and find the data that goes in the page (*might be in file system or swap slot or none*).
  - Obtain a frame to store the page.
  - Fetch the data into the frame.
  - Point the page table entry for the faulting virtual address to the physical address (*see userprog/pagedir.c*).

Managing Frame Table

- Frame table maps frames to user pages, and other data of your choice.
- It allows Pintos to implement an *eviction policy*.
- Frames can be obtained from the user pool by calling *palloc_get_page(PAL_USER)* function.
- Be careful about the *user pool* and *kernel pool*.

The process of eviction

- Here are the steps:
    - Choose a frame to evict (LRU, FIFO or …). The "*accessed*" and "*dirty*" bit in the page table will be useful.
    - Remove any reference to the frame from the page table (Be careful on sharing).
    - If necessary write the page to the file system or to swap.

Managing Swap Table

- Swap table tracks in-use and free swap slots.
- You may use BLOCK_SWAP block device for swapping.
- From the vm/build directory, run
  - *pintos-mkdisk swap.dsk —swap-size=n*
  - *swap.dsk* will be automatically attached when you run Pintos.
- Allocate swap slots lazily, i.e. when needed.

Managing Memory Mapped Files

- File system is in general accessed by read/write system calls.
- A secondary interface is to use "*mmap*" system call.
- Program can then use memory directly with virtual addresses.
- You must track what memory is used by memory mapped files to properly handle page faults and to ensure that mapped files do not overlap any other segments.

Suggested order of implementation

- Frame table.
  - Change process.c to use your frame table allocator.
  - Don't implement swapping yet.
  - If you run out of frames: panic the kernel.
- Supplemental page table and page fault handler.
  - Change process.c to record the necessary information to the SPT when loading an executable and setting up its stack.
- Stack growth, mapped files and so on.

Requirements

Paging

- In Pintos, currently executables are loaded in memory by *process_load() in userprog/process.c*.
- The entire executables loaded before run.
- Data which may never be read is loaded.
- Instead load the segments in demand.

Stack growth

- Currently Pintos implements a fixed stack size.
- Implement stack growth. If new space needed, allocate additional pages.
- Impose some limit on stack size, i.e. 8MB.
- First stack page need not be loaded lazily.
- All stack pages are candidates for eviction.

Memory mapped files

- Implement following system calls:
- mapid_t mmap (int fd, void *addr)
  - Maps the file as consecutive virtual pages starting at address *addr.*
- void munmap (mapid_t mapping)
  - Unmap the file from the virtual pages.
  - Modified pages should be written to the disk.
  - Pages should be removed from the process' list of virtual pages.

Accessing User Memory

- Adapt your code to access user memory in while handling a system call.
- More on this in the project page.

What to submit?

- *<lastname_firstname>.tar* package containing all source files.
- Don't forget to include makefile and README.
- Email .tar package: {tkosar, mbulut}@buffalo.edu
- Deadline is December 4th, 2013 @ 11.59pm.
- Also write a design document and submit the hardcopy by the beginning of class December 5th.

Bottom line!

Start working now!

Questions