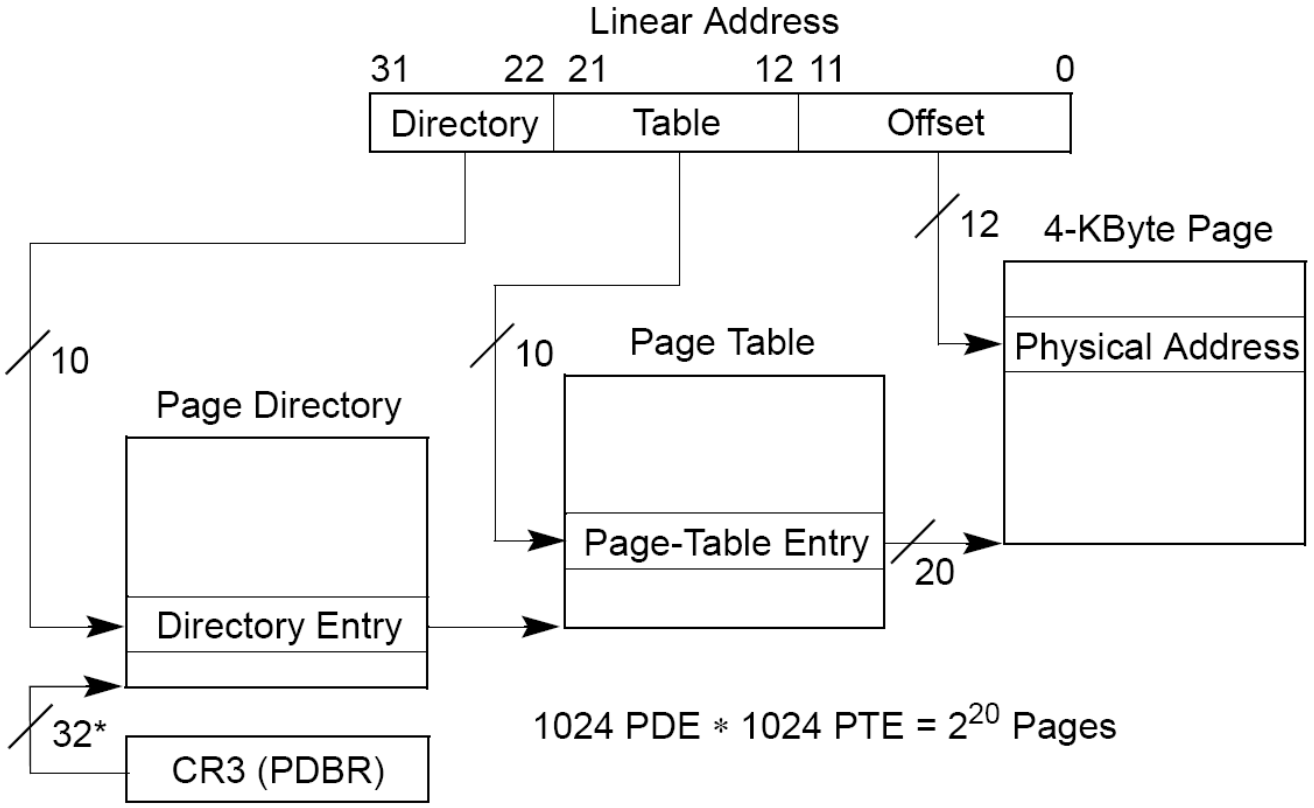
Project 3: Virtual Memory

Prof. Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

Introduction (1)

- Paging in the x86 architecture



Introduction (2)

- **Current Pintos VM implementation**
  - Use paging
  - Page size: 4KB
  - Each process has its own page tables
    - The page directory is allocated when the process is created (`pagedir_create()` @ `userprog/pagedir.c`)
    - `(struct thread *) t->pagedir` points to the page directory (`load()` @ `userprog/process.c`)
    - The (secondary) page tables are dynamically created if necessary (`lookup_page()` @ `userprog/pagedir.c`)
    - For kernel region, processes have the same mapping (`PHYS_BASE` ~ `0xffffffff`)

Introduction (3)

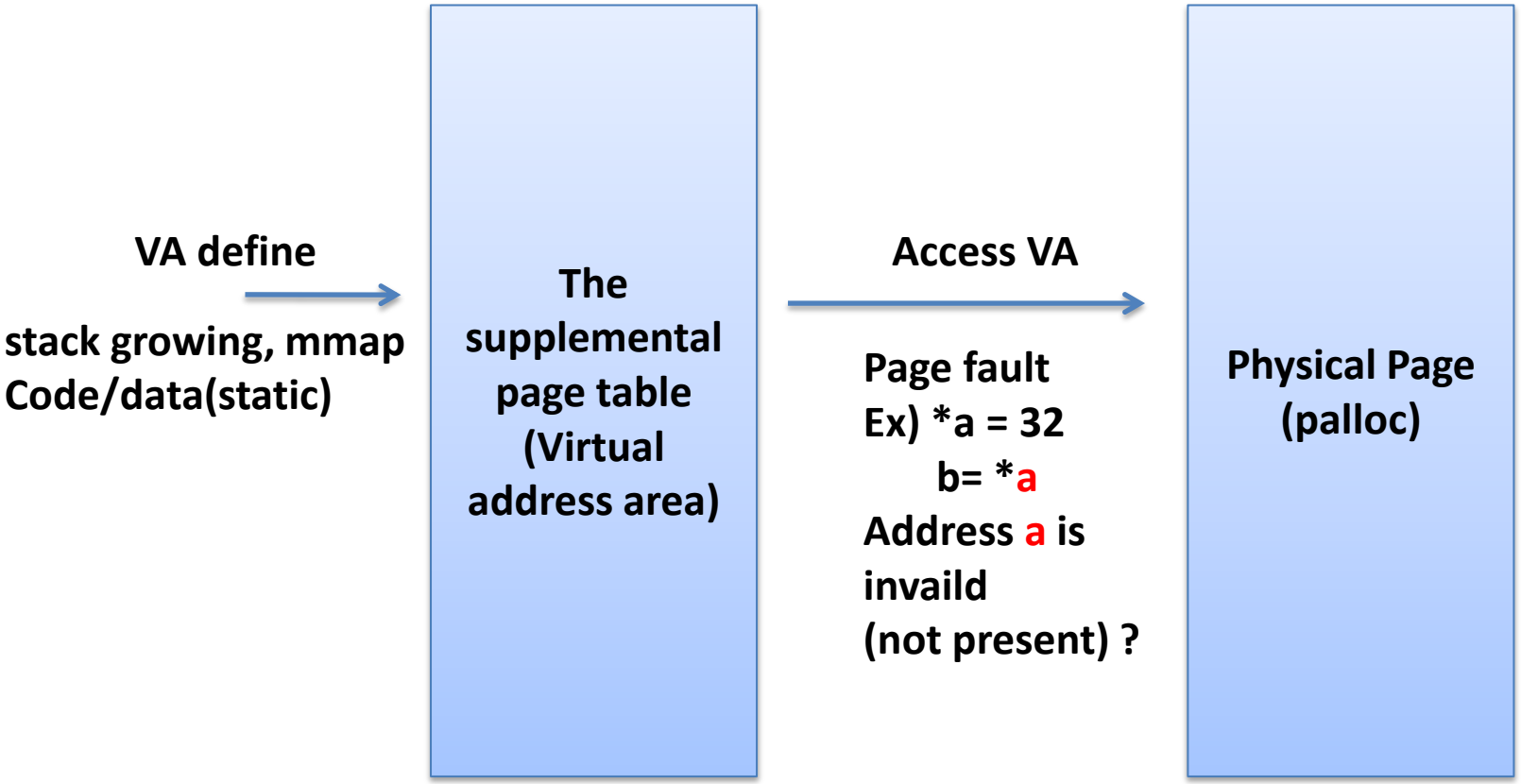
- **Current Pintos VM implementation (cont'd)**
  - No demand paging
    - When a process is created, all the contents of code and data segments are read into the physical memory (`load_segment() @ userprog/process.c`)
  - Fixed stack size
    - Only one stack page is allocated to each process (`setup_stack() @ userprog/process.c`)

Project 3 Overview

- **Requirements -> Paging**
  - Lazy loading (or demand paging)
  - Swapping in/out pages from/to swap disk
  - Dynamic stack growth (project 3-1)
  - Memory mapped files

Project 3 Overview

- Paging



Stack Growth (1)

- **Growing the stack segment**
  - Allocate additional pages as necessary
  - Devise a algorithm that attempts to distinguish stack accesses from other accesses
    - Bug if a program writes to the stack below the stack pointer
    - However, in x86, it is possible to fault 4 ~ 32 bytes below the stack pointer
  - You may impose some absolute limit on stack size
  - The first stack page need not be allocated lazily
    - The page is initialized with the command line arguments
  - All stack pages should be candidates for eviction
    - An evicted stack page should be written to swap

Stack Growth (2)

- **How to obtain the user stack pointer?**
  - You need the current value of the user program's stack pointer on page fault
    - Compare it with the faulted address
  - When the page fault occurred in the user mode
    - Use `(struct intr_frame *) f->esp`
  - When the page fault occurred in the kernel mode
    - `struct intr_frame` is not saved by the processor
    - `(struct intr_frame *) f->esp` yields an undefined value
    - Save `esp` into `struct thread` on the initial transition from user to kernel mode

Memory Mapped Files (1)

- **Example**
  - Writes the contents of a file to the console

```
#include <stdio.h>
#include <syscall.h>
int main (int argc, char *argv[])
{
    void *data = (void *) 0x10000000;

    int fd = open (argv[1]);
    mapid_t map = mmap (fd, data);
    write (1, data, filesize(fd));
    munmap (map);
    return 0;
}
```

Memory Mapped Files (1)

- **Example**
  - Writes the contents of a file to the console

```
#include <stdio.h>
#include <syscall.h>
int main (int argc, char *argv[])
{
    void *data = (void *) 0x10000000;

    int fd = open (argv[1]);
    mapid_t map = mmap (fd, data);
    write (1, data, filesize(fd));
    munmap (map);
    return 0;
}
```

Define virtual address

Page fault

Memory Mapped Files (2)

- **System calls to implement**

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

  - mmap() fails if
    - fd is 0 or 1
    - The file has a length of zero bytes
    - addr is 0
    - addr is not page-aligned (4096)
    - The range of pages mapped overlaps any existing set of mapped pages
  - All mappings are implicitly unmapped when a process exits

Memory Mapped Files (3)

- **Managing mapped files**
  - Lazily load pages in mmap regions
    - For the final mapped page, set the bytes beyond the end of the file to zero
  - Use the mmap'd file itself as backing store for mapping
    - All pages written to by the process are written back to the file
  - Closing or removing a file does not unmap any of its mappings
    - Once created, a mapping is valid until `munmap()` is called or the process exits

Test case

- **Almost all tests**
  - except page-xxxx, mmap-over-code, mmap-over-data
- **Project 2**
  - 30%
  - Minus one point per one FAIL
- **Project 3-1**
  - 70%

Submission

- **Due**
  - Nov 23, 11:59PM
  - Only submit a source code with comments
  - [group_number]_project3-1.tar.gz
  - Tar and gzip your Pintos source codes

```
$ cd pintos
$ (cd src/userprog; make clean)
$ tar cvzf 1_project3-1.tar.gz src
```

  - Upload it at sys.skku.edu
  - threads, userprog directories