Exam #2 Review

sseshadr

Agenda

- Administrative things
  - Exam tomorrow, should've been studying
  - Proxy lab out tomorrow
  - You're probably done with malloc. Congratulations!
- Exam Review
- Brief intro to proxy lab

Studying

- What to do
  - Look at the 213 lecture schedule and read lectures
  - Read the book
  - Do the past exams
  - **Understand** the labs
- Lectures?
  - “Memory Hierarchy” to “Dynamic Memory Allocation”
- Book?
  - The readings next to the lectures on the schedule

[Subset of] Exam Topics

- Physical Memory
  - SRAM/DRAM? SSD? Volatile vs. non-volatile? Bus?
  - Disks (calculating capacity, mem. access time)
  - Locality (temporal, spatial)
  - Cache memories
    - Terms: types of misses, write-{through/back/allocate}, blocking, L1, L2, …
    - Given S E B & memory accesses, calculate hits/misses/evictions

[Subset of] Exam Topics

- Linking
  - Types of ELF files (.o, .so, and the “a.out” file)
  - Static libraries (.a “archive files”)
  - What goes in an ELF?
  - Symbol resolution (strong/weak, global/external/local?)
  - static keyword
  - The `ld` command? What is dynamic linking (`dlopen`)?
  - Types of interpositioning

[Subset of] Exam Topics

- ECF and Processes
  - What is ECF and “when can it happen”
  - Kernel code vs. user code, context switching at a high level
  - Synchronous ECF (traps, faults, aborts) vs. asynchronous ECF (interrupts)
  - **<u>SIGNALS</u>** and **<u>HANDLERS</u>** (a lot to know here….)
    - Non-queuing, signal system calls, defaults, deferring
    - Here is some code with signals and handlers and…
      - It’s trying to do X, but it doesn’t. What’s wrong with it?
      - What all could it print output? (Gets worse with sys io)

[Subset of] Exam Topics

- ECF and Processes
  - Types of processes, what is reaping?
  - What is a process group?
  - What is async-signal-safety?
  - fork, exec, wait/waitpid
    - Fork COPIES and ISOLATES memory
    - Exec REPLACES memory
  - Process lab concepts
    - **<u>SYNCHRONIZATION</u>**
  - Non-local jumps
    - How do you use sigsetjmp, siglongjmp? Stack dangers?

[Subset of] Exam Topics

- I/O
  - open/close/read/write, wrappers, RIO, standard IO
    - When do you use what?
  - File descriptor table, initialized with 0, 1, 2
  - File metadata
  - File sharing and redirection
    - Interaction with fork (refcnt, file position)
    - dup and dup2

[Subset of] Exam Topics

- Virtual Memory (a lot to know here…)
  - **<u>TRANSLATIONS</u>**
  - Address anatomy
  - VM system design
  - TLB
  - Page table & PTEs
  - mmap
  - Page faults
  - Special registers
  - COW and Demand Paging (ZFOD)?

[Subset of] Exam Topics

- Virtual Memory
  - Diagrams…

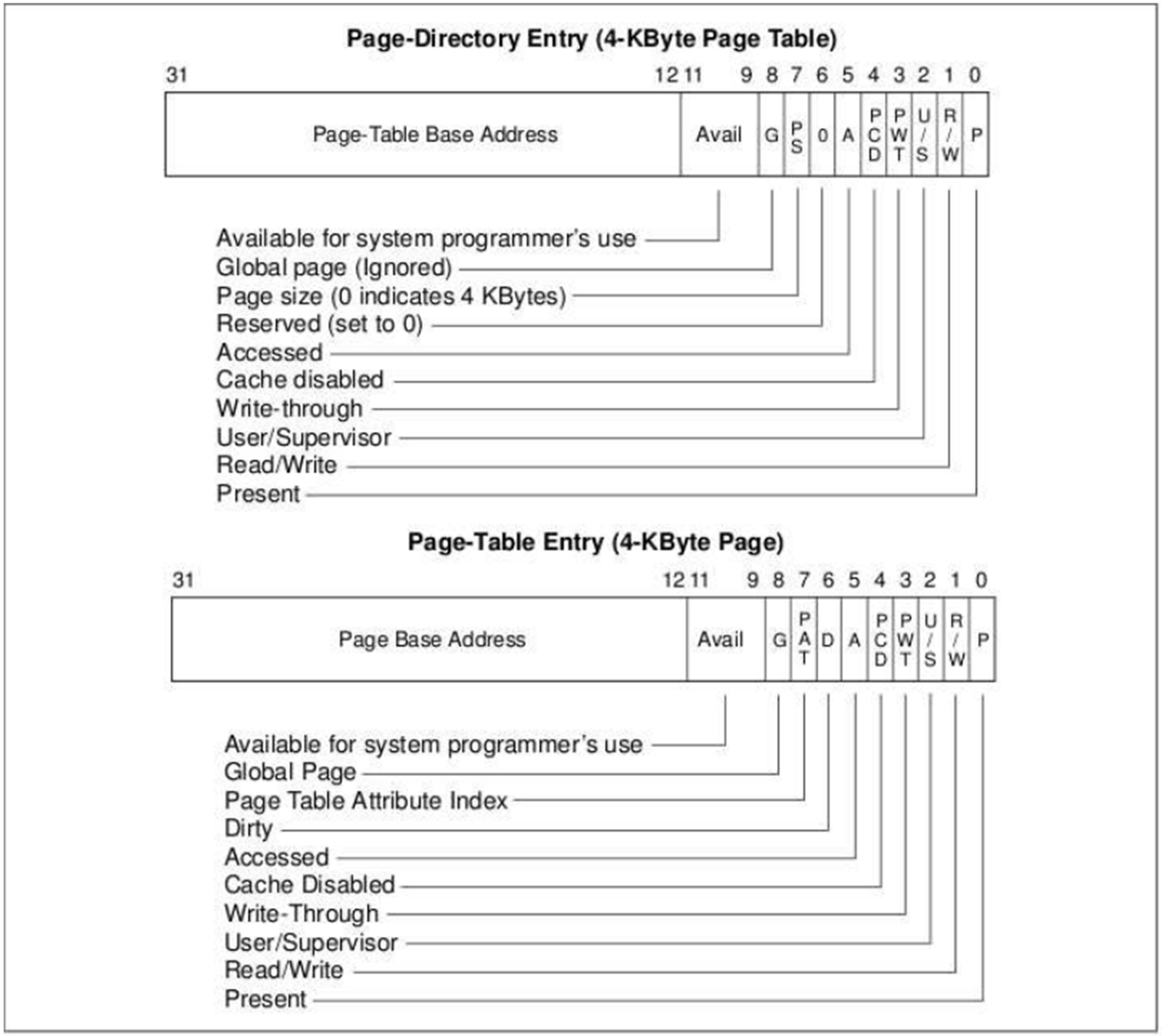

Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages

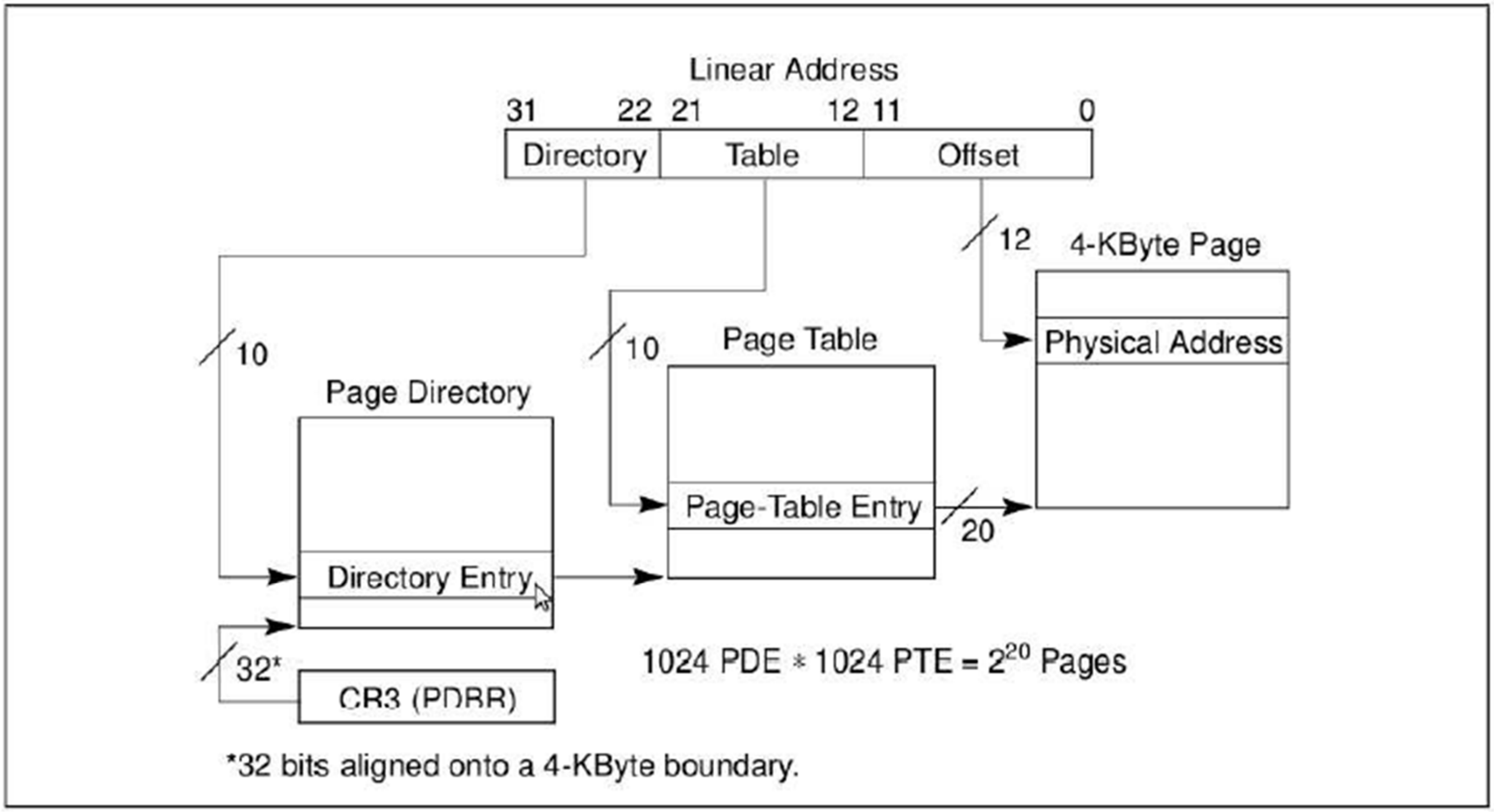

Figure 3-12. Linear Address Translation (4-KByte Pages)

[Subset of] Exam Topics

- Dynamic Memory Allocation
  - malloc/calloc/realloc/free/sbrk
  - Types of fragmentation
  - Ways to coalesce
  - Implementation
    - Types of lists (pros and cons)
    - First-fit/best-fit? Address-ordered vs. LIFO?
  - Garbage collection
  - Identifying memory bugs

[Subset of] Exam Topics

- Lab & Recitation Topics
  - Can you
    - Simulate a cache?
    - Simulate calls to malloc/free?
    - Draw process trees?
  - Macros
  - Pointer declarations

one.txt	`abc`
two.txt	`nidoking`
three.txt	`conflageration`

You are also presented with the `main()` function of three small programs (header includes omitted), each of which uses simple and familiar functions that perform file i/o operations. For each program, determine what will be printed on `stdout` based on the code and the contents of the file. Assume that calls to `open()` succeed, and that each program is run from the directory containing the above files. (The program execution order does not matter; the programs are independent.)

Program 1:

```
void main() {
  char c0 = 'x', c1 = 'y', c2 = 'z';
  int r, r2 = open("one.txt", O_RDONLY);

  read(r2, &c0, 1);
  r = dup(r2);
  read(r2, &c1, 1);
  close(r2);
  read(r,  &c2, 1);

  printf("%c%c%c", c0, c1, c2);
}
```

output to `stdout` from Program 1:	abc

one.txt	abc
two.txt	nidoking
three.txt	conflageration

Program 2:

```
void main() {
  char c0 = 'x', c1 = 'y', c2 = 'z';
  char scrap[4];
  int pid, r, r2 = open("two.txt", O_RDONLY);
  r = dup(r2);

  if (!(pid = fork())) {
    read(r, &c0, 1);
    close(r2);
    r2 = open("two.txt", O_RDONLY);
    read(r2, &scrap, 4);
  } else {
    waitpid(pid, NULL, 0);
    read(r,  &c1, 1);
    read(r2, &c2, 1);
  }

  printf("%c%c%c", c0, c1, c2);
}
```

output to `stdout` from Program 2:	nyzxid

one.txt	`abc`
two.txt	`nidoking`
three.txt	`conflageration`

Program 3:

```
void main() {
  char c[3] = {'x', 'y', 'z'};
  int r, r2, r3;

  r  = open("three.txt", O_RDONLY);
  r2 = open("three.txt", O_RDWR);
  dup2(1, r3);
  dup2(r2, 1);

  read(r, &c[0], 1);
  printf("elephant");
  fflush(stdout);
  read(r,   &c[1], 1);
  read(r2,  &c[2], 1);
  write(r3, &c[0], 3);

  printf("%c%c%c", c[0], c[1], c[2]);
}
```

Pretend r3 was initialized to some nice value (even 0 works)

output to `stdout` from Program 3:	clr

Questions?