CS 241 Midterm study guide

04 March 2012

Contents

1 POSIX **1**
1.1 Examples . . . 1

2 System Calls **1**

3 Libraries **1**
3.1 strlen vs sizeof . . . 1

4 Pointers **1**
4.1 Pointer Operations . . . 1
4.1.1 Pointer Arithmetic . . . 1
4.2 NULL . . . 1
4.3 Function Pointers . . . 1
4.4 Strings . . . 1
4.4.1 What's the difference between char $c[80]$ and char* c . 1
4.4.2 What's the difference between a string and a string literal? . . . 2
4.4.3 How do strcpy, strcat, and strncat work? . . . 2

5 Memory **2**
5.1 Memory Fragmentation . . . 2
5.1.1 Internal . . . 2
5.1.2 External . . . 2
5.2 Cache/Page Table replacment policies . . . 2
5.2.1 OPT . . . 2
5.2.2 LRU . . . 2
5.2.3 Working Set . . . 2
5.2.4 Thrashing . . . 2

5.2.5 Belady's anomaly 2
5.2.6 When is a process swapped out to disk 2
5.3 Stack 2
5.3.1 When is the stack full? 2
5.4 Heap 2
5.4.1 Page faults 2
5.4.2 How does malloc and free work? 2
5.5 Virtual Memory and Paging/Segmentation 3
5.5.1 Virtual vs Physical memory 3
5.5.2 Paging vs Segmentation 3
5.5.3 Virtual to Physical address translation in multi-level page tables 3
5.5.4 Advantages of multi-level page tables 4
5.5.5 Determining optimal page size 4
5.5.6 Calculating the number of pages per page table 4

6 Threads and Processes **4**
6.1 Process 4
6.1.1 Creating a process using fork() 4
6.1.2 exec() 4
6.1.3 Orphans and Zombies 4
6.2 Threads 4
6.2.1 Shared Resources 4
6.2.2 Creating a thread using $\text{pthread}_{\text{create}}$() 4
6.2.3 $\text{pthread}_{\text{detach}}$() and $\text{pthread}_{\text{join}}$ 4
6.2.4 Exiting a thread with out a thread library exit call . . 4
6.2.5 What are the maximum number of threads that can be run concurrently? How is this number determined? 5
6.3 Context Switching 5
6.3.1 In Processes 5
6.3.2 In Threads 5
6.3.3 Kernel-Space vs User-Space thread managment 5
6.4 Memory Consistency 5
6.4.1 Shared memory 5
6.4.2 Locking, Blocking, and Semaphores 5
6.4.3 POSIX wait() 6

7 Scheduling **6**
7.1 Five state model: started ,running, ready, blocked, terminated 6
7.2 Scheduling schemes 6

7.2.1 Wait Time 6
7.2.2 Turnaround time 6
7.2.3 Response time 6
7.2.4 Preempting 6
7.2.5 Quanta 6
7.2.6 Fairness, progress guarentees, and interactive systems 6
7.2.7 Schemes 6
7.3 Execution Order 7
7.4 Starvation 7
7.5 Blocking 7
7.6 Signals and Interrupts 7
7.6.1 Explain how re-entrant functions are used in C. 7
7.7 Convoy Effect 7

1 POSIX

1.1 Examples

2 System Calls

3 Libraries

3.1 strlen vs sizeof

4 Pointers

4.1 Pointer Operations

4.1.1 Pointer Arithmetic

4.2 NULL

4.3 Function Pointers

4.4 Strings

4.4.1 What's the difference between char $c[80]$ and char* c

- What about when they're used in sizeof()?

4.4.2 What's the difference between a string and a string literal?

4.4.3 How do strcpy, strcat, and strncat work?

5 Memory

5.1 Memory Fragmentation

5.1.1 Internal

5.1.2 External

5.2 Cache/Page Table replacment policies

5.2.1 OPT

5.2.2 LRU

5.2.3 Working Set

- Locality

5.2.4 Thrashing

5.2.5 Belady's anomaly

5.2.6 When is a process swapped out to disk

5.3 Stack

5.3.1 When is the stack full?

5.4 Heap

5.4.1 Page faults

- SEGFAULT

5.4.2 How does malloc and free work?

- Memory Selection Algorithms

    - Implicit Free Lists

  - Explicit Free Lists
  - Segregated Free Lists
  - Buddy System

5.5 Virtual Memory and Paging/Segmentation

5.5.1 Virtual vs Physical memory

- Advantages of virtual memory

5.5.2 Paging vs Segmentation

5.5.3 Virtual to Physical address translation in multi-level page tables

- MMU
  - Example: How does the virtual memory subsystem know the exact location where a particular page is stored on disk, if it is swapped out of memory?
- TLB
- Algorithm for address translation goes here
  - Example: Assuming a 32-bit address space and 4 KB pages, what is the virtual page # and offset for virtual address 0xd34f6a5?
  - Example: Suppose we have a 64-bit address space and 16 KB pages. How big is the page table of a single process? What if it was multi-level?

5.5.4 Advantages of multi-level page tables

5.5.5 Determining optimal page size

5.5.6 Calculating the number of pages per page table

6 Threads and Processes

6.1 Process

6.1.1 Creating a process using fork()

- Starts new process with an incremented PC count

6.1.2 exec()

- Example: Explain how a shell process can execute a different program.

6.1.3 Orphans and Zombies

6.2 Threads

6.2.1 Shared Resources

6.2.2 Creating a thread using pthread$_{\text{create}}$()

6.2.3 pthread$_{\text{detach}}$() and pthread$_{\text{join}}$

- Example: Explain how one process can wait on the return value of another process.

6.2.4 Exiting a thread with out a thread library exit call

- How it happens: calling exit(), return, or termination

- Problems

6.2.5 What are the maximum number of threads that can be run concurrently? How is this number determined?

6.3 Context Switching

6.3.1 In Processes

6.3.2 In Threads

6.3.3 Kernel-Space vs User-Space thread managment

6.4 Memory Consistency

6.4.1 Shared memory

- Example: X is a global variable and initially X=0. What are the possible values for X after two threads both try to increment X?

6.4.2 Locking, Blocking, and Semaphores

- Mutual exclusion

- Semaphore and mutex

- Designing a a lock system for concurrent programming

6.4.3 POSIX wait()

7 Scheduling

7.1 Five state model: started ,running, ready, blocked, terminated

7.2 Scheduling schemes

7.2.1 Wait Time

7.2.2 Turnaround time

7.2.3 Response time

7.2.4 Preempting

7.2.5 Quanta

7.2.6 Fairness, progress guarentees, and interactive systems

7.2.7 Schemes

- Round Robin

  - Quanta length vs performance

- First Come First Serve (FCFS)

- Pre-emptive SJF

- Non-preemptive

  - Smallest Initial response time?

  - Smallest Initial wait?

  - Smallest Initial turnaround time?

- smalled average wait time?

- longest average wait time?

7.3 Execution Order

7.4 Starvation

7.5 Blocking

7.6 Signals and Interrupts

7.6.1 Explain how re-entrant functions are used in C.

7.7 Convoy Effect