Quick inventory of Systems concepts and C programming skills

System (OS+Networking+Architecture) concepts

1. Virtual memory
   - virtual pages vs. physical frames
   - page tables & TLBs
   - page replacement (eviction) policies

2. Caches
   - mechanisms, associativity, etc.
   - cache replacement policies
   - cache coherency mechanisms (update vs. invalidate)
   - practical knowledge of L1 & L2 caches and how they attach to cores in multicore architectures

3. SMP vs. NUMA parallel architectures

4. Filesystem & disk basics
   - data structures, inodes, etc.
   - what "block device" means, and how filesystems and block devices interact

5. Network stack basics
   - how data flows from/to an app to/from a NIC through protocol-handling layers in the OS

6. Concurrency & synchronization
   - concurrency vs. parallelism
   - how the OS switches between processes (or kernel threads)
   - how the kernel itself runs (gets control, its stack, etc.)
   - scheduling
   - basic mutual exclusion

7. How a function call/return is implemented in the machine

Diagnostic Homework on Concepts

Answer the following questions (in bullet form, not wordy sentences). Explain with figures wherever appropriate. A picture is worth a thousand words! You can of course consult textbooks to answer the questions. If you find most of the concepts in this diagnostic homework foreign, then perhaps you should consider taking remedial actions before enrolling in the Advanced OS course (CS 6210).

1. Consider a processor that supports virtual memory. It has a virtually indexed physically tagged cache, TLB, and page table in memory. Explain what happens in such a processor

from the time the CPU generates a virtual address to the point where the referenced memory contents are available to the processor.
2. Distinguish between segmentation and paging.
3. Explain all the actions from the time a process incurs a page fault to the time it resumes execution. Assume that this is the only runnable process in the entire system.
4. Explain the following terms: working set of a process, thrashing, paging daemon, swapper, loader, and linker.
5. Explain page coloring and how it may be used in memory management by an operating system.
6. Explain clearly the costs associated with a process context switch.
7. Explain the functionality of the different layers found in the network protocol stack of an operating system such as Linux.

C Programming skills

1. Pointers
- use in general, * and & operators
- how to reason about them ("box & arrow" diagrams)
- struct ptr assignment vs. struct assignment

2. Function pointers

3. Dynamic memory management - general use (malloc & free)

4. Process memory layout
- stack, heap, global data, code, etc.
- differences between pointers to memory in different segments (e.g., ptr to heap mem vs. ptr to stack mem)

5. Data representation - i.e., everything is just bits

6. Casting - especially non-obvious casts, e.g., ptr to/from int or anything to/from void*

7. Compiling & linking
- header files
- libraries (.a)
- declarations vs. definitions

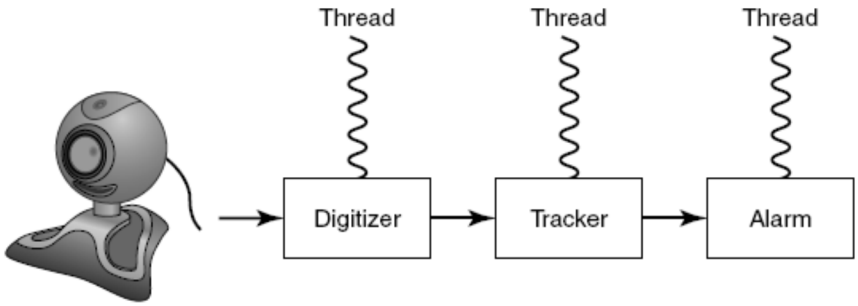
8. Segfaults - why they happen and how to debug them

9. Race conditions & bugs

Example simple programming project you should be able to write without much trouble

- A multithreaded producer-consumer program using the pthreads library

Description:



The figure shows a simple video-based surveillance application. This is a continuous application wherein camera frames are captured and digitized by the digitizer, and analyzed by the tracker, triggering control actions by the alarm module. The digitizer component of the program continually converts the video into a sequence of frames of pixels. The tracker component analyzes each frame for any content that needs flagging. The alarm component takes control action based on the tracking. Write a multi-threaded program for this application using the pthreads library. You can assume the digitizer, tracker, and alarm components are available to you as functions that you can call from the threads that implement each of the three components. The main functionality you have to implement is the producer-consumer functionality between the components.