Final Exam

CPS 210: Operating Systems
Spring 2013

This exam has four P* problems. Answers are graded on substance, not style. Please try to confine your answers to the space provided: you may attach extra sheets as necessary (please put your name on them). 300 points total.

P1. A Series of Tubes

Unix/Linux has a standard program called cat that runs as a single-threaded process: it reads bytes from stdin and writes them to stdout, in a loop, until stdin returns end-of-file (EOF), and then cat exits.

Suppose I type the following command to your Lab 2 Devil Shell:

dsh> cat | cat | cat

This will result in a flurry of activity, and then all processes block (sleep). Then, as I type characters at my keyboard, the processes work together to process my input. Then, when I type EOF (control-d), the pipeline shuts down.

This question has five parts that ask you to explain in some detail what happens. The parts are equally weighted. This question is worth 150 points (30 points each).

(It starts on the next page.)
(a) Draw the process tree at the point when the pipeline is set up and all processes are sleeping. List the sequence of system calls issued in/by each process up to that point, in order. For each sleeping process, list the events that could cause it to wake up. Extra credit: include arguments for the system calls.
(b) Deep inside the kernel each pipe is a **producer/consumer bounded buffer**: “it’s like a soda machine but with bytes instead of bottles”. Each pipe has one producer process (the writer) and one consumer process (the reader). The producer writes bytes into the pipe, blocking if the pipe is full. The consumer reads bytes from the pipe, blocking if the pipe is empty. The read calls return bytes from the pipe in the order written.

Implement pipe **write** and **read** using monitors (locks/mutexes and condition variables) only. Any kind of pseudocode is fine as long as its meaning is clear. Don’t worry about the data structure or data movement for the buffer itself: just the synchronization.
(c) Implement pipe write and read using semaphores only. Any kind of pseudocode is fine as long as its meaning is clear. Don’t worry about the data structure or data movement for the buffer itself: just the synchronization.
(d) Suppose now that I type a long sequence of characters at the keyboard (e.g., a draft of the 1000-page textbook I am planning to write). The bytes flow in one cat and out the other. This will cause the processes to transition between the ready, running, and blocked states and also between kernel mode and user mode. Illustrate an execution schedule by drawing a gantt chart (or any picture you prefer) showing how the processes change states and modes through time as the bytes flow through them. Don't worry about what happens at the end. (That's next.)

You may assume a single core. Could these processes ever run in parallel on multiple cores, i.e., could the pipeline run faster on a multi-core machine?
(e) Finally, when I am done composing my new textbook, I enter an end-of-file (control-d). My cats have consumed all of my text, written it back out, and then forgotten it. Now what happens?

Draw another gantt chart (or any picture you prefer) showing an execution schedule as the cats exit and the pipeline shuts down. (Don’t forget to show the actions of the shell process itself.)

As the pipeline shuts down, the kernel reclaims resources used. In particular, it destroys the pipes and reclaims the kernel memory allocated for their buffers. When in the schedule does the kernel destroy the pipes? How does it know that it is safe to do so?
P2. Charades! Show You Know the Lingo

Each of these prompts can be answered with a term that is highlighted in red on my slides. Name that term to get the points. (50 points)

(a) A named/typed event directed at or received by an Android component

(b) An Android component type with a get/put service interface allowing other components to access structured content

(c) A digitally signed statement endorsing a principal’s public key

(d) A grant (promise) from a server to a client for exclusive access to a resource (e.g., a lock or a region of a file) for a period of time

(e) A language that defines the interface for a Remote Procedure Call (RPC) service

(f) A scheme for distributing a sequence of storage blocks across multiple disks/servers for higher read/write throughput

(g) A redundancy scheme for storage to protect and recover data in case one disk has an unrecoverable failure in a RAID storage system

(h) Special hardware instructions needed to implement safe mutual exclusion (e.g., spinlocks) on multicore systems

(i) A practical eviction/replacement policy for a block cache or page cache

(j) A malware program that an attacker tries to trick you into running by wrapping it with some functionality you want (or think you might want)
P3. Elastic Scaling in Cartoons

Consider a service handling a stream of incoming requests from clients. The service runs on a cluster of $N$ virtual machines (VMs) of identical speed, obtained from a cloud hosting provider. The service is incrementally scalable: it automatically distributes its incoming requests evenly across all servers (VMs) in the cluster. It selects a server for each incoming request, and then executes the request entirely on that one server.

This part asks you to illustrate some performance and scaling behaviors within the service. Each question asks you to draw a “back of napkin” (rough, qualitative) graph showing the usual measures as a function of offered load. You may assume that the per-request cost (service demand $D$) and inter-arrival times are “well-behaved”: all standard assumptions apply, as discussed in class. (50 points)

(a) Draw a graph showing idealized throughput, utilization, and mean response time as a function of offered load. Label any important features of the graph.
(b) Suppose now that the service has **elastic scaling**: e.g., as offered load increases along the x-axis, the system acquires more servers as needed to handle the load. Draw a graph similar to part (a), but showing the throughput, utilization, and mean response time of an **individual server** and how they change as the service scales up by adding more servers.

(c) Suppose now that one of the servers is slower than the others, but continues to receive requests at the same rate as the other servers. Draw a graph comparing the **mean response time** of this slow server to the others, as a function of offered load (for a fixed cluster size).
(d) What impact does the “one slow server” assumption of (c) have on the response time distribution of the service as a whole? Sketch a response time distribution as a cumulative distribution function (CDF) for some fixed cluster size and load level, with and without the slow server.
P4. Concept Checker

Take your best shot at defining the following terms with a phrase or sentence. If you want to add another, or maybe two, or a figure, or examples, that’s OK, if it helps. (50 points)

(a) atomic update (modifying a structure in memory)

(b) atomic commit (modifying a structure on “disk”)

(c) atomic transaction

(d) context

(e) fault

(f) upcall
(g) thrashing

(h) network partition

(i) stub (for an RPC or system call)

(j) isolation boundary, or protection domain

Extra credit. Help our students who will take the course next semester! Draw a flowchart illustrating how a team member works (or should work) to complete a CPS 210 lab project. (on the extra page)

Thank you: have a great summer etc.!