Stanford University
Computer Science Department
CS 140 Final
Dawson Engler
Winter 2000

This is an open-book exam. You have 180(!) minutes to answer as many questions as possible. The number in parenthesis at the beginning of each question indicates the number of points given to the question. Write all of your answers directly on the paper. Make your answers as concise as possible. Sentence fragments ok.

Question	Points	Score
1	45	
2	12	
3	28	
4	15	
5	20	
total	120	
bonus	3	

Stanford University Honor Code

In accordance with both the letter and the spirit of the Honor Code, I did not cheat on this exam nor will I assist someone else cheating.

Name:

Signature:

1. Short-attention-span questions (45 points)
Answer each of the following questions and, in a sentence or two, say *why* your answer holds. (5 points each).

1. (5 points) Assume we eliminate the use of inodes by putting the information they contain in their associated directory entry. Ignoring the issue of hard links, what is a performance drawback of doing so? (And please give an example Unix command that illustrates this problem.)

2. (5 points) After a long period of no dating, your cs140 partner gets really excited about "signal" and "wait" and wants to use them throughout your nachos code, despite it not being implemented as a monitor (i.e., multiple threads can run "related" code concurrently). What general problem will you probably run into?

3. (5 points) On hardware that uses a TLB and a fixed page size, what feature of the TLB will place a hard upper bound on the amount of physical memory your system can use?

4. (5 points) To impress your friend XY you change his file system to always prefetch the next 64K of disk memory on every disk access. He's enthusiastic about the speedup and throws a party. You then try the same trick for your friend XX. She's unimpressed by the negligible improvement and is irritated that you've wasted her afternoon. Who would you expect to be using the more stupid file system? (And why?)

5. (5 points) Ethernets use statistical multiplexing to increase link utilization. Please give an example of where we've used similar ideas to increase resource utilization, and give a reasonable, non-statistical multiplexing alternative.

6. (5 points) You are using a reliable byte stream protocol (such as TCP) that acknowledges the number of bytes it has received so far (e.g., "the last byte received was byte 1023"). For the case where a sender has multiple, unacknowledged packets outstanding what might be a better acknowledgement message for the receiver to send?

7. (5 points) You and I want to agree on a time to eat lunch. I send you an email, saying "let's meet at 1pm." You reply "ok, we'll meet at 1pm." Do you know for sure that we have agreed upon when to go to lunch? If so, say why; if not give a concrete example of failure that could mess things up. (You may assume our mail readers attempt to handle packet loss using techniques discussed in class.)

8. (5 points) Roughly order stack, code, and heap segments as to how well you would expect them to perform under a paged VM system that uses LRU page replacement. (And give the intuition(s) for your ordering.)

9. (5 points) What virtual memory paging and eviction strategy should you use for virtual address range that exhibits good spatial locality but poor temporal locality? Additionally, please give an realistic example of when such a situation can arise.

2. One name. (12 points)

Disgruntled with overwork, your 140 partner decides to simplify your nachos implementation by having all processes live in a single address space. Now, instead of having a page table per process, there is one for the entire system, and each process has a protection table that tracks the virtual pages it is allowed to access.

1. (6 points) At a high level sketch how to do linking/loading on this system.

2. (6 points) What correctness challenge does the use of a single address space create for a Unix fork implementation? (I.e., where a new process is created by cloning an existing one.)

3. Sort-of RAID (28 points)

To help pay Stanford tuition, you buy a cut rate disk for your computer. Unfortunately, you notice that over the course of use one data block per file goes bad.

1. (10 points) Ignoring system crashes, explain how to compensate for a bad data block using error correction ideas from RAID. (Note: you only have a single disk and you should only need a single extra "parity" block per file.) At a high level please state (1) what you must do on a disk write and (2) what you must do when a block goes bad. (You may assume that the disk controller will give you a "bad block" error when you try to read or write a bad block.)

2. (4 points) When modifying cached blocks, you can either recompute the parity block when you evict a block, or when you modify it (and thus still have the unmodified version of the block around). What is the main (significant) advantage of recomputing it on every modification?

3. (10 points) Assume we do not ignore crashes, but do ignore the problem of blocks going bad during the period of crash to the end of recovery. (I.e., the set of bad blocks right before a crash is exactly equal to the set of bad blocks after recovery.) What order should you write out disk blocks to eliminate the chance that your file system will be in an unrecoverable state? (And why?) What does an "fsck" file system recovery program now have to do on after a crash?

4. (4 points) Assume blocks can go bad during the period of crash to recovery: what problem does this pose?

4. Stacks-R-Fun (15 points)

(10 points) Assume you are building a user-level (*not* kernel-level) threads system that manages multiple threads in the same address space. Each thread has their own stack. For ease of use, you want your system to transparently grow thread stacks dynamically. Please give a high level sketch of what virtual memory support the OS should provide so that you can (1) detect when a thread's stack has been exceeded and (2) grow the thread's stack. (And, of course, how you'd use this functionality to do these two acts.) Additionally, please briefly discuss the tradeoffs around where you decide to place thread stacks in virtual memory.

(5 points) What do you have to be prepared to do to dynamically grow a thread's stack? What complication does taking the address of a stack variable create?

5. Weird concurrency (20 points)

Given a list of runnable threads, the sort-of TERA machine executes them round-robin by running one statement in the first thread, always immediately context switching to the next thread, running one statement in it, etc.

You have been presented with the following two "push" implementations that operate on a single shared stack. The code was intended to run on a TERA machine that supports a maximum of two threads: state whether it is free of race conditions (and provide a succinct intuition for why) or is broken (and if so, give a specific execution sequence that will cause it to fail).

Note, the lack of a "pop" implementation is not accidental: you may ignore it and any possible interference with it. Additionally, to simplify reasoning, each statement in the implementation has a number preceding it: *the machine will run this statement in its entirety, then always(!) context switch to the next thread, run a statement in it, etc.*

1. **(10 points)** The first (partial) stack implementation is linked list based.

```
struct elem {
        struct elem *next;
        /* ..stuff.. */
};

/* stack head */
struct elem *head;

void push(struct elem *e) {
          struct elem *old;

/* 1 */   old = head;
/* 2 */   e->next = head;
/* 3 */   head = e;
/* 4 */   if(head != e) {
/* 5 */        head = e;
/* 6 */        e->next = old;
          }
}
```

2. **(10 points)** The second partial stack implementation holds elements in an "infinite" array.

```
elem stack[]; /* an "infinite" stack */
int n = 0;    /* position of last stack element */

void push(struct elem *e) {
          int i;

/* 1 */    i = n;
/* 2 */    n = n + 1;
/* 3 */    stack[i] = e;
/* 4 */    if(stack[i] != e)
/* 5 */         stack[i+1] = e;
}
```