CS 111 Final Examination
Spring Quarter, 2024

You have 3 hours (180 minutes) for this examination; the number of points for each question indicates roughly how long we think it will take to answer that question. Make sure you print your name and sign the Honor Code below. During this exam you are allowed to consult three double-sided pages of notes that you have prepared ahead of time, as well as any of the .c and .h files from your solution to Assignment 7. Other than these materials, you may not consult any other sources, including books, notes, your laptop, or phones or other personal devices. If there is a trivial detail that you need for one of your answers but cannot recall, you may ask the course staff for help.

Be sure to write only on the front sides of pages. We'll be using Gradescope to grade the exams and may not see information written on the back sides. There are extra pages at the end of the exam, if you need more space for an answer.

I acknowledge and accept the Stanford University Honor Code. I have neither given nor received aid in answering the questions on this examination, and I have not consulted any external information other than three double-sided pages of prepared notes.

_____
(Signature)

_____
(Print your name, legibly!)

_____
(SUNet email id)

Problem	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	Total
Score													
Max	12	8	11	5	6	20	12	20	10	12	14	20	150

Problem 1 (12 points)

Indicate whether each of the statements below is true or false, and explain your answer briefly.

(a) The dispatcher runs as a special high-priority thread.

(b) Reducing the length of time slices could worsen a system's average response time.

(c) There is no fragmentation when a stack is used for storage allocation.

(d) If the "ordered writes" approach is used for file system crash recovery, there is no need to run `fsck` to restore consistency during reboots.

(e) Page offsets in virtual addresses must be the same size as page offsets in physical addresses.

(f) The linker must be aware of the page size.

Problem 2 (8 points)

(a) (4 points) Suppose that the name of each file were stored with the file on disk (for example, in its inode), not in a separate directory file. Would this make file lookups faster or slower, and why?

(b) (2 points) Give an example of how someone could infer trust in a closed-source OS such as macOS or Windows.

(c) (2 points) Give an example of how someone could infer trust in an open-source OS (such as Linux) that would not also apply to a closed-source OS.

Problem 3 (11 points)

(a) (4 points) In the solution discussed in lecture for implementing locks in a multiprocessor operating system, the `lock` and `unlock` methods both (a) disable interrupts and (b) busy-wait on a lock variable. Does the order of these operations matter? If so, indicate what the order must be and explain why. If not, explain why not.

(b) (3 points) What is the difference between internal and external fragmentation?

(c) (4 points) In order to avoid thrashing in a paging system, the kernel must keep in memory the pages that a process is actively using; this is called the "working set" of the process. All pages outside the working set can be moved to backing store. If the page size is doubled, will the size (**in bytes**) of the working set of a process increase, decrease, or stay the same? Explain your answer briefly.

Problem 4 (5 points)

Circle "Yes" or "No" next to each of the statements below to indicate whether that function is performed by a linker.

(a) Combine like sections from different object files.	Yes	No
(b) Fill in the addresses in a jump table.	Yes	No
(c) Modify addresses in code sections.	Yes	No
(d) Create a symbol table.	Yes	No
(e) Compute the virtual address of the first byte of the data segment.	Yes	No

Problem 5 (6 points)

In the original DOS FAT file system, each entry in the FAT (File Allocation Table) was 16 bits in size and stored either a block pointer or a special reserved value indicating "end-of-file" or "block free".

(a) (3 points) If each disk block contains 512 bytes, what is the largest disk size that this version of FAT can support (you can give the answer as a power of two, if that is more convenient)? Explain your answer.

(b) (3 points) If the FAT is loaded entirely into main memory, how much memory will it take? Explain your answer.

Problem 6 (20 points)

A friend of yours wrote the following function: given two accounts, it should transfer a given amount from the larger account to the smaller one, if the larger account has sufficient funds:

```
1  struct account {
2      std::mutex mutex;
3      double balance;
4  }
5
6  void transferFromLarger(struct account *a1, struct account *a2,
7          double amount) {
8      if ((a1->balance > a2->balance) && (a1->balance >= amount)) {
9          a1->mutex.lock();
10         a1->balance -= amount;
11         a1->mutex.unlock();
12         a2->mutex.lock();
13         a2->balance += amount;
14         a2->mutex.unlock();
15     } else if ((a2->balance > a1->balance) && (a2->balance >= amount)) {
16         a2->mutex.lock();
17         a2->balance -= amount;
18         a2->mutex.unlock();
19         a1->mutex.lock();
20         a1->balance += amount;
21         a1->mutex.unlock();
22     }
23 }
```

(a) (5 points) Unfortunately, this code does not always behave correctly when invoked concurrently by different threads. Describe a sequence of events that causes the code to misbehave.

(b) (15 points) Use the next page to rewrite `transferFromLarger` so that it always behaves correctly. Your solution must use the same definition of `struct account` (e.g. there must be a separate `mutex` for each account).

Write your solution to Problem 6(b) here:

```
struct account {
    std::mutex mutex;
    double balance;
}

void transferFromLarger(struct account *a1, struct account *a2, double amount)
{









}
```

Problem 7 (12 points)

The `exchange` method on `std::atomic` types was introduced in class as a primitive to use in implementing higher-level synchronization mechanisms such as locks and condition variables. For example, consider the following code:

```
std::atomic<int> x;
...
int y = x.exchange(12);
```

The `exchange` method will atomically read `x`, overwrite it with `12`, and return the old value of `x`, which is then stored in `y`.

A friend shows you the following code and claims that it will atomically exchange the values of the variables `x` and `y`:

```
std::atomic<int> x = 3;
std::atomic<int> y = 4;
...
y.exchange(x.exchange(y));
```

Suppose that two threads execute the last line of code above at the same time; assume that `x` has the value 3 and `y` has the value 4 initially, and that the variables are shared between the threads. If this code is truly atomic, then once both threads have executed the code `x` will be 3 again and `y` will be 4 (the variables will have been exchanged twice, restoring each variable to its initial value).

(a) (6 points) Is it possible that both variables could end up with the value 4? If so, explain how; if not, explain why not. You may assume that memory reads and writes are atomic.

(b) (6 points) Is it possible that both variables could end up with the value 3? If so, explain how; if not, explain why not.

Problem 8 (20 points)

Suppose that each of the changes below is made to a file system. Consider whether each change affects the performance, consistency, and/or durability of the file system. For each of the three properties indicate whether the change improves that property, degrades it, or leaves it about the same. Explain each answer briefly.

(a) (5 points) Starting from a file system with no cache, a block cache with synchronous writes is added.

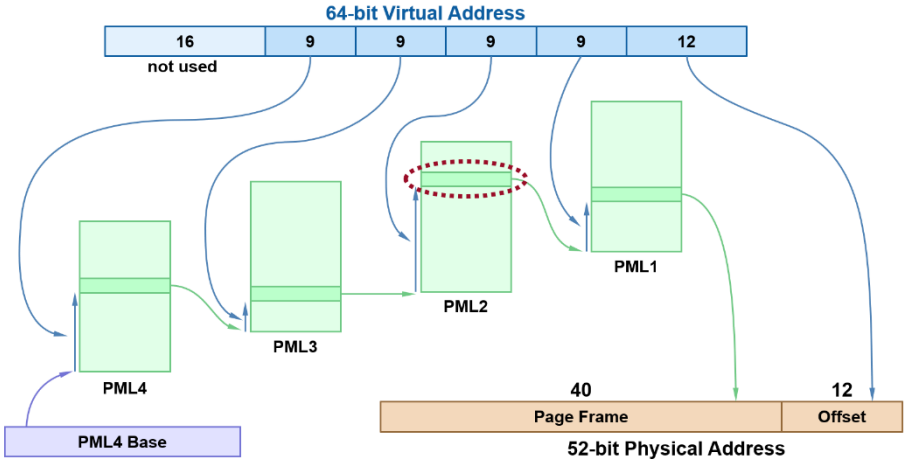
(b) (5 points) Starting from a block cache with synchronous writes, a delayed-write mechanism is added.

(c) (5 points) Starting from a block cache with delayed writes, an ordered-write mechanism is added for both metadata and data; the ordered writes are performed synchronously.

(d) (5 points) Starting from a block cache with delayed writes, a write-ahead log is added. Only metadata changes are written to the log, and the log entry for an operation is flushed to disk before the operation modifies any blocks in the cache.

Problem 9 (10 Points)

The figure below shows the address translation mechanism used for 64-bit mode in Intel x86 processors, which was discussed in lecture. It uses 4 levels of page map to translate virtual addresses to physical addresses.



As part of translating a virtual address, one entry will be read from a PML2 page map (dotted oval above). This page map entry resides in physical memory, so the physical address for that entry will need to be generated in order to read it. The empty box below represents the **physical address of the PML2 entry**. Mark up the box to indicate where every bit of the physical address comes from (if a bit is computed, explain the computation). Be as precise as possible: for example, give the size in bits of each field.

52-bit physical address

Problem 10 (12 points)

Describe a program that calls `malloc` and `free` (or C++ `new` and `delete`) in a way that requires at least 1200 total bytes of heap space, even though the program never has more than 1000 bytes of memory allocated at once. You don't need to write actual code, but your description must be clear and precise. Your program must work for any (non-garbage-collecting) implementation of `malloc` and `free`; it can test the pointers returned by `malloc` if that helps.

Problem 11 (14 points)

Consider an operating system that uses a FIFO thread scheduler and supports demand paging. The OS is running on a system with a single-core CPU and two disks; one disk is used exclusively as the backing store for demand paging, while the other disk holds files. When a particular workload is running on the system, the measured utilizations of the elements (% of time each unit is doing useful work) are:

CPU	20%
Paging disk	99.7%
Disk for files	5%

For each of the following actions, indicate whether it is likely to increase the CPU utilization and explain your answer briefly.

(a) Get a faster processor.

(b) Get a paging disk with more capacity.

(c) Start more processes executing.

(d) Kill some existing processes.

(e) Replace the paging disk with a flash drive that has the same capacity.

(f) Switch from a FIFO scheduler to the 4.4 BSD scheduler.

(g) Install more memory.

Problem 12 (20 points)

Extend your solution for Assignment 7 (Reading Unix V6 Filesystems) by writing a C function with the following signature:

```
int inode_upgrade(const struct unixfilesystem *fs, struct inode *inp);
```

The `inp` argument points to an inode for a "small" file; the function must make whatever changes are needed to convert this file to the "large" file format in V6 Unix while retaining the file's contents. It should return 0 if it is successful and -1 if an error occurred. Your solution may use any of the code in your solution for Assignment 7, and it may also assume the existence of the following C functions:

```
int diskimg_writesector(int fd, int sectorNum, void *buf);
int block_alloc(const struct unixfilesystem *fs);
```

`diskimg_writesector` is similar to the `diskimg_readsector` function you used in Assignment 7 except that it writes to disk instead of reading. The `block_alloc` function will allocate a free block on the disk and return its block number (if there is an error it will return -1).

The `inode_upgrade` function need not write the inode back to disk (the caller will handle this), but it must write back any other changes by calling `diskimg_writesector`. You need not worry about consistency issues that could occur after an error or crash.

```
int inode_upgrade(const struct unixfilesystem *fs, struct inode *inp)
{
```

Use this page if you need extra space for any problem on the exam

Use this page if you need extra space for any problem on the exam