CS 111 Midterm Examination
Spring Quarter, 2024
Solutions

You have 90 minutes for this examination; the number of points for each question indicates roughly how many minutes you should spend on that question. Make sure you print your name and sign the Honor Code statement below. During this exam you are allowed to consult two double-sided pages of notes that you have prepared ahead of time and your solutions for the Caltrain and Party problems; other than that, you may not consult books, notes, your laptop or cell phone, or any other materials. 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 two double-sided pages of prepared notes.

_____
(Signature)

_____
(Print your name, legibly!)

_____
(SUNet email id)

Problem	#1	#2	#3	#4	#5	Total
Score						
Max	10	6	10	15	40	81

Problem 1 (10 points)

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

(a) No scheduling algorithm can produce average response times better than STCF (Shortest Time to Completion First).

Answer: true. By always running the thread that will complete soonest, STCF produces optimal average response time.

(b) Dynamic linking tends to reduce memory usage.

Answer: true. Dynamic linking enables shared libraries, where a single copy of a library package can be shared by many processes. Without dynamic linking, there would be a separate copy of the library in each process that uses it.

(c) When a linker creates an executable file, it arranges for the sections coming from a particular object file to be adjacent to each other in the program's virtual address space.

Answer: false. The linker arranges like sections from each file next to each other (e.g. all code sections). As a result, the sections from a single file will almost certainly be separated from each other in virtual address space.

(d) Dangling pointers cannot occur with a memory allocator that uses reference counting for reclamation.

Answer: true. Since reference counting counts the pointers to an object, an object cannot be deleted until the last pointer is deleted (the problem with reference counting is that it may never reclaim groups of objects with circular references).

(e) Virtual addresses must be the same size as physical addresses.

Answer: false. Virtual addresses in a system can be either larger or smaller than physical addresses. For example, the Intel x86-64 architecture discussed in lecture has 48-bit virtual addresses and 52-bit physical addresses.

Problem 2 (6 points)

For each of the scheduling algorithms listed below, indicate whether or not the algorithm is preemptive. If it is preemptive, describe briefly the conditions under which preemption occurs.

(a) First come first served

Answer: non-preemptive.

(b) Round robin

Answer: preemptive. Preemption occurs when a time slice ends.

(c) Shortest time to completion first

Answer: preemptive. Preemption occurs when a thread becomes ready and its completion time is shorter than that of a currently running thread.

Problem 3 (10 points)

(a) (4 points) An operating system needs to trust that a given process will not access any memory except that explicitly made available to it by the operating system (such as its code, data, and stack segments). One of the ways it does this is with the "present" bits in page tables. Which form of trust does this represent (assumption, inference, or substitution) and why?

Answer: this is trust by substitution. The OS isn't 100% sure that a process will access only virtual addresses in the defined segments, so it clears the "present" bits in all other pages as a backup precaution. If the application errs and makes a stray memory reference, the cleared "present" bits will result in a trap into the OS.

(b) (2 points) Under what conditions does round robin scheduling behave identically to FIFO?

Answer: if the completion time for all threads is shorter than the time slice.

(c) (4 points) In multi-core processors, there is typically a separate ready queue for each core. Describe one advantage of this approach, in comparison to a single shared ready queue, and one disadvantage.

Advantage of separate ready queues: eliminating lock contention related to the ready queue that could occur if a single ready queue is shared by many cores.

Disadvantage of separate ready queues: it could result in uneven loading of the cores. In the worst case, some cores might sit idle (because their ready queues are empty) while other cores have many threads waiting in their ready queues.

Problem 4 (15 points)

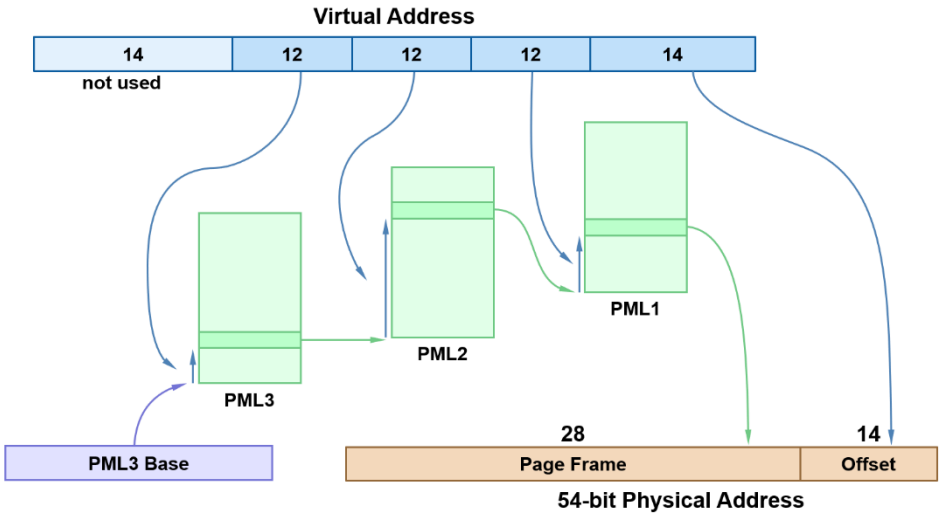
You have been hired to design a page-based virtual memory architecture for a system with 16 KB pages (2^{14} bytes) and 42-bit physical addresses. Virtual addresses will be stored in 64 bits, but only the lower 50 bits will be used (the upper 14 bits will always be zeroes).

(a) (5 points) Draw a diagram of a single page map entry for this system, showing all of the fields it must contain, along with their sizes in bits.

Answer: each page map entry must contain at least the following information:

- **Physical page number: 28 bits**
- **Read-only: 1 bit**
- **Present: 1 bit**
- **It's also fine, but not required, to mention additional bits that weren't discussed before the exam cutoff, such as "kernel-only", "referenced", or "dirty"**

(b) (10 points) Draw a diagram of the virtual address translation mechanism (in the same form as the diagram shown in class for the Intel x86-64 translation mechanism) showing how virtual addresses are translated to physical addresses. It must be clear how many levels of page map there are and which virtual address bits are used at each point in the translation. You may assume each page map entry occupies 4 bytes.



Explanation: since page map entries require 4 bytes and pages are 2^{14} bytes long, each page map holds 2^{12} page map entries (4096). Since virtual page numbers are 36 bits long (50 – 14), there will need to be 3 levels of page map, each of which maps 12 bits of virtual address.

Problem 5 (40 points)

Amazon has hired you to implement an inventory management system for their newest automated warehouse. The warehouse contains a collection of stock items (distinct items that customers might purchase), and you must keep track of how many of each item are currently available for purchase. To do this, define a class `Inventory` with the following methods:

```
Inventory(int max_stock_id);
~Inventoryh();
void add(int stock_id, int count);
void reserve(std::vector<int> &stock_ids, std::vector<int> &counts);
```

Each stock item has a unique identifier (its "stock identifier"). The constructor argument gives the largest stock number that will ever be passed to either the `add` or `reserve` method, so stock numbers will range from 0..`max_stock_id`, inclusive. The warehouse starts out with no inventory. When new inventory arrives at the warehouse, the `add` method is invoked; its arguments indicate which stock number was replenished and how many new items were added to inventory.

When a new order arrives at the warehouse, the `reserve` method will be invoked. Its `stock_ids` argument lists all of the stock numbers requested in that order, and the `counts` argument indicates how many items of each stock number have been purchased (`counts[i]` corresponds to `stock_ids[i]`). The `reserve` method must not return until there is sufficient inventory to satisfy the order. Once `reserve` returns, automated picker machinery will fetch the ordered items and ship them to the customer. You do not need to worry about the picking and shipping functions, but you must make sure that there will always be sufficient inventory to fill an order when `reserve` returns.

The `reserve` method must not reserve any inventory until it can complete the order. This means that while `reserve` is waiting for one stock item to be replenished, other orders may consume all of the inventory of other items needed for the waiting order.

Additional information:

- You must write your solution in C++ using `std::mutex` and `std::condition_variable`.
- Use the monitor style for your code, as discussed in class and required for Assignment 2.
- Your solution does not need to be optimal in terms of efficiency (focus on simplicity), but it must not use busy-waiting.
- Your solution need not be fair (orders need not be filled in the exact order of arrival), but if there is enough inventory to fill one of the orders currently in `reserve` then that order must be allowed to proceed.
- Try to make your solution simple and obvious; we will reduce your score by up to 20% if your solution is overly complicated.
- If you can't remember exact names and syntax for C++ library methods you can either ask the staff or just make your best guess. As long as your intent is clear and precise we aren't going to quibble about syntax.

Implement the Inventory class here:

```
class Inventory {
public:
    Inventory(int max_stock_id);
    ~Inventory();
    void reserve(std::vector<int> &stock_ids, std::vector<int> &counts);
    void add(int stock_id, int count);

private:
    std::mutex mutex;
    std::condition_variable new_inventory;
    std::vector<int> inventory;
}

Inventory::Inventory(int max_stock_id)
    : mutex()
    , new_inventory()
    , inventory(max_stock_id+1, 0)
{}

void Inventory::add(int stock_id, int count)
{
    std::unique_lock<std::mutex> lock(mutex);
    inventory[stock_id] += count;
    new_inventory.notify_all();
}


void Inventory::reserve(std::vector<int> &stock_ids, std::vector<int> &counts)
{
    bool order_ok = false;
    std::unique_lock<std::mutex> lock(mutex);

    while (!order_ok) {
        order_ok = true;
        for (size_t i = 0; i < stock_ids.size(); i++) {
            if (inventory[stock_ids[i]] < counts[i]) {
                new_inventory.wait(lock);
                order_ok = false;
                break;
            }
        }
    }

    for (size_t i = 0; i < stock_ids.size(); i++) {
        inventory[stock_ids[i]] -= counts[i];
    }
}
```

Additional working space for Problem 5

Additional working space for any problem, if needed