

CS 111 Midterm Examination

Spring Quarter, 2023

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; 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.

Note: 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	12	10	40	78

Problem 2 (6 points)

No existing approach to memory allocation and freeing is perfect. Briefly describe one disadvantage of each of the following techniques:

(a) Slab-based storage allocation

(b) Reference counting for storage reclamation

(c) Garbage collection for storage reclamation

Problem 3 (12 points)

- (a) In the code below, do all of the references to `x` refer to the same variable? If so, explain why; if not, explain how many different variables there are, and why.

```
int x = 100;

void func1(...)
{
    ...
    x += 100;
    ...
}

void func2(...)
{
    x--;
    std::thread t([](){x = 300;});
    t.join();
    std::cout << "x = " << x << std::endl;
    ...
}
```

- (b) In the code below, do all of the references to `x` refer to the same variable? If so, explain why; if not, explain how many different variables there are, and why.

```
int x = 100;

void func(char **argv)
{
    x += 100;

    int childPid = fork();
    if (childPid == 0) {
        // This code runs in the child process.
        x += 100;
        execvp(argv[0], argv);
        ...
    }

    // This code runs in the parent process.
    waitpid(childPid, NULL, 0);
    std::cout << "x = " << x << std::endl;
}
```

Problem 3, cont'd

- (c) In the code below, do all of the references to `x` refer to the same variable? If so, explain why; if not, explain how many different variables there are, and why.

```
void func1(int x)
{
    int sum = x;

    for (int i = 0; i < 10; i++) {
        int x = i*i;
        sum += x;
    }
    sum += x;
    std::cout << "x = " << x << std::endl;
}
```

- (d) In the code below, do all of the references to `x` refer to the same variable? If so, explain why; if not, explain how many different variables there are, and why.

```
void func1(void)
{
    int x = 0;

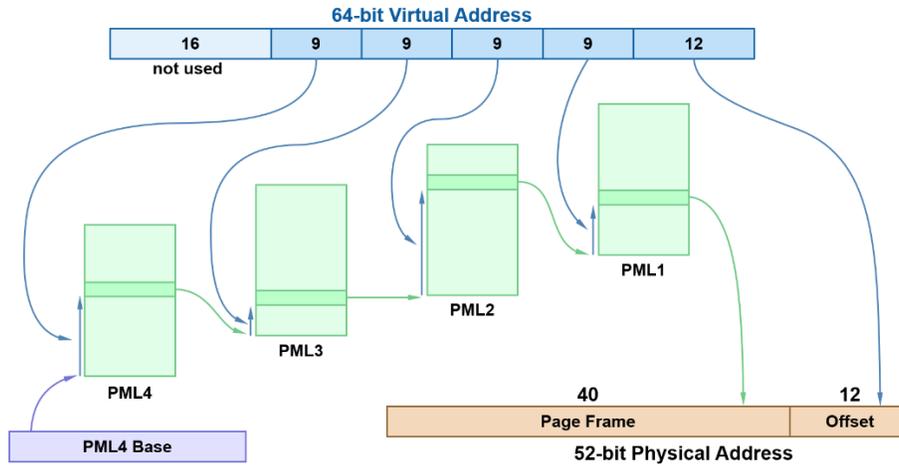
    for (int i = 0; i < 10; i++) {
        x += i*i;
    }
    std::cout << "x = " << x << std::endl;
}

void func2(void)
{
    int x = 0;

    for (int i = 0; i < 10; i++) {
        x += i+20;
    }
    std::cout << "x = " << x << std::endl;
}
```

Problem 4 (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 48 bits of virtual address to 52-bit physical addresses.



- (a) (5 points) Consider a process that needs only one page each for code, data, and stack (assume it has only one thread), with virtual memory layout as discussed in lecture. How many PML2 page maps will the operating system have to allocate for this process? Explain your answer briefly.
- (b) (5 points) Now suppose that the process' thread makes deeply nested recursive method calls, which cause the stack to grow, and that the operating system allocates additional pages for the stack as needed. How many pages must the stack consume before the operating system will have to allocate another PML2 page map? You can express your answer as a power of 2 if that is more convenient. Explain your answer briefly.

Problem 5 (40 points)

In this problem you must implement an airplane boarding process similar to that of Southwest Airlines: each boarding pass contains an integer boarding position (no groups A, B, and C) and passengers board in order of their positions (lowest boarding position first). Specifically, you must define a C++ class `Flight` with a constructor, destructor, and two methods:

```
Flight(size_t max_passengers);
~Flight();
void wait_to_board(size_t position);
bool board_next();
```

The `max_passengers` argument to the constructor indicates the maximum number of passengers that will board the flight. When a passenger arrives at the gate, it invokes `wait_to_board`; the `position` argument indicates the position from the passenger's boarding pass: you can assume that it is unique for this `Flight` and lies between 0 and `max_passengers-1`, inclusive. The `wait_to_board` method must not return until it is this passenger's turn to board.

The gate agent will call `board_next` to allow the first waiting passenger to begin boarding; once that passenger has passed the boarding pass scanner, the gate agent will invoke `board_next` to board the next passenger, and so on until `board_next` returns false. The `board_next` method should return false if there are no available passengers; otherwise it should wake up the passenger with the lowest remaining boarding position and return true. If a passenger is not present when their boarding position is reached, they will be skipped and the next higher position will board; if such a passenger eventually arrives, then they must board before other passengers with higher positions.

You do not need to worry about what passengers do after `wait_to_board` returns or how the gate agent determines that it should make each call to `board_next`.

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 must not use busy-waiting.
- For full credit, you must not use `notify_all`, and each call to `board_next` should invoke `notify_one` at most once.
- 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.

Use the next page(s) for your answer.

Implement the Flight class here:

```
class Flight {
public:
    Flight(size_t max_passengers);
    ~Flight();
    void wait_to_board(size_t position);
    bool board_next();
private: /* You must fill in the part below. */
```

```
}
```

```
// Implement the methods here.
```

Additional working space for any problem, if needed

Additional working space for any problem, if needed