New York University
CSCI-UA.0202: Operating Systems (Undergrad): Spring 2024
Midterm Exam (Token: V0)

- **Write your name, NetId, and token on this cover sheet and on the cover of your blue book.**
- **Put all of your answers in the blue book; we will grade only the blue book.** Thus, if the blue book answer is blank or incorrect, you will not get credit, regardless of the exam print-out.
- **At the end, turn in both the blue book and the exam print-out.** "Orphaned" blue books with no corresponding exam print-out will not be graded.
- This exam is **75 minutes**. Stop writing when "time" is called. *You must turn in your print-out and blue books; we will not collect them.* Do not get up or pack up in the final ten minutes. The instructor will leave the room 78 minutes after the exam begins and will not accept exams outside the room.
- There are **11** problems in this booklet. Many can be answered quickly. Some may be harder than others, and some earn more points than others. You may want to skim all questions before starting.
- **This exam is closed book and notes. You may not use electronics: phones, tablets, calculators, laptops, etc.** You may refer to ONE two-sided 8.5x11" sheet with 10 point or larger Times New Roman font, 1 inch or larger margins, and a maximum of 55 lines per side.
- Do not waste time on arithmetic. Write answers in powers of 2 if necessary.
- If you find a question unclear or ambiguous, be sure to write any assumptions you make.
- Follow the instructions: if they ask you to justify something, explain your reasoning and any important assumptions. **Write brief, precise answers. Rambling brain dumps will not work and will waste time.** Think before you start writing so that you can answer crisply. Be neat. If we can't understand your answer, we can't give you credit!
- If the questions impose a sentence limit, we will not read past that limit. In addition, *a response that includes the correct answer, along with irrelevant or incorrect content, will lose points.*

Do not write in the boxes below.

I (xx/8)	II (xx/18)	III (xx/28)	IV (xx/10)	V (xx/7)	VI (xx/8)	VII (xx/19)	Fb (x/2)	Total (xx/100)

Name: Solutions

NetId:

I Follow the machine (8 points)

1. [8 points] In the code below, `main()` is written in C. The function `f()` is written in assembly, in an unusual way; for example, it has no prolog or epilog.

What does `main()` print?

Hints:

- Draw the stack as the program executes, and think about what each line is doing
- Recall that the C compiler expects a function's return value to be in `%rax`.
- The arguments to `movq` are in the order `movq SOURCE, DESTINATION`.

```
1     #include <stdio.h>
2
3     extern int f();
4
5     int main()
6     {
7         int a;
8         a = f();
9         printf("%d\n", a);
10        return 0;
11    }
12
13    ------Below here is assembly----
14    .global f
15
16 f:
17        # the next two lines take the address of the code at g, and push
18        # that address on the stack
19        movq $g, %rcx
20        pushq %rcx
21
22        # the next two lines take the address of the code at h, and push
23        # that address on the stack
24        movq $h, %rcx
25        pushq %rcx
26
27        movq $0x4, %rax
28
29        ret
30
31 g:     movq $0x5, %rax
32        ret
33
34 h:     movq $0x6, %rax
35        ret
```

5. This is a (simplified) instance of return-oriented programming, a technique used by attackers, and which we will come back to later in the semester. In this case, at the end of f, the relevant contents of the stack, going downward, are:

return address within main (because f was called)
address of g
address of h

The `ret` in f brings execution to h, the `ret` in h brings execution to g, and the `ret` in g brings control back to `main`. The last function to set `%rax` is g, which moves 5 into `%rax`, thereby returning that value in `a`.

II Lab 2: `ls` (18 points)

2. [18 points] Write a function `print_ownership` in syntactically valid C:

```
static int err_code;

void print_ownership(struct stat* statbuf)
{
    // YOUR CODE HERE




}
```

The specifications are as follows:

- `print_ownership` takes as input an already-filled `struct stat`, passed as a pointer. This argument represents a given file. The function should print the username and group name of the file in the following format:

  ```
  u: <username>
  g: <group name>
  ```

- If there are no errors, then <`username`> and <`group name`> should simply be the username and group name that `ls` would print in a long-listing of the file. If there is an error obtaining either one, then print the string `error` instead. For example, if your code obtains the username as `mwalfish` but fails in getting the group name, then the output should be:

  ```
  u: mwalfish
  g: error
  ```

- As in lab 2, there is a variable `err_code`. If there is an error obtaining the username, set bit 0 of `err_code`; if there is an error obtaining the group name, set bit 1 of `err_code`. You cannot make any assumptions about whether these bits of `err_code` were set previously.
- Assume that username and group name each fit in less than 256 ASCII characters (C type `char`).
- Some helpful definitions are on the next page. Note that there is more information than you need.

```
/* Return information about the file given by pathname.
   Places the returned information in the buffer pointed to by statbuf.
   On success, return 0. On error, return -1. */
int stat(const char* pathname, struct stat *statbuf);

/*
 * This is the same as the macro that you were given in lab2. Take as
 * input 'info' and prints the given 'ch' if the permission 'mask' exists,
 * or "-" otherwise.
 */
#define PRINT_PERM_CHAR(info, mask, ch) printf("%s", (info & mask) ? ch : "-");

/* convert the mode field in a struct stat to a file type, for -l printing */
const char* ftype_to_str(mode_t mode);

/* Tests whether the argument refers to a directory */
bool is_dir(char* pathandname);

/* Get username for uid. Return 1 on failure, 0 otherwise. */
static int uname_for_uid(uid_t uid, char* buf, size_t buflen);

/* Get group name for gid. Return 1 on failure, 0 otherwise. */
static int group_for_gid(gid_t gid, char* buf, size_t buflen);

/* Open a directory */
DIR *opendir(const char *name);

/* Read a directory entry */
struct dirent *readdir(DIR *dirp);

/* A function that you may have filled in when implementing lab 2 */
void list_file(char* pathandname, char* name, bool list_long);

/* A function that you may have filled in when implementing lab 2 */
void list_dir(char* dirname, bool list_long, bool list_all, bool recursive);

struct stat {
   dev_t     st_dev;
   ino_t     st_ino;
   mode_t    st_mode;
   nlink_t   st_nlink;
   uid_t     st_uid;
   gid_t     st_gid;
   dev_t     st_rdev;
   off_t     st_size;
   blksize_t st_blksize;
   blkcnt_t  st_blocks;
}
```

```
extern int err_code;

void print_ownership(struct stat* statbuf)
{
    char username[256];
    char groupname[256];
    int urc, grc;

    urc = uname_for_uid(statbuf->st_uid, username, sizeof(username));
    grc = group_for_gid(statbuf->st_gid, groupname, sizeof(groupname));

    printf("u: ");
    if (!urc) {
        printf("%s\n", username);
    } else {
        printf("error\n");
        err_code |= 0x1;   /* or err = err | 0x1;
                              or err = err | 0b1 */
    }

    printf("g: ");
    if (!grc) {
        printf("%s\n", groupname);
    } else {
        printf("error\n");
        err_code |= 0x2;   /* or err = err | 0x2;
                              or err = err | 0b10; */
    }

}
```

III Lab 3 and Concurrent programming (28 points)

3. [8 points] This question asks you to write `customer` and `supplier`, as in lab 3. We include comments to guide you. There are helper definitions on the next page, but there is more information than you need. You can assume that the queues are already properly synchronized. Use syntactically valid C and C++

```
/* ----------------------------------------------------------------------
 * customer --
 *
 *      The main customer thread. The argument is a pointer to the
 *      shared Simulation object.
 *
 *      Dequeue Tasks from the customer queue and execute them.
 *
 * Results:
 *      Does not return.
 *
 * ----------------------------------------------------------------------
 */
static void*
customer(void* arg)
{
    // TODO: Your code here



}

/* ----------------------------------------------------------------------
 * supplier --
 *
 *      The main supplier thread. The argument is a pointer to the
 *      shared Simulation object.
 *
 *      Dequeue Tasks from the supplier queue and execute them.
 *
 * Results:
 *      Does not return.
 *
 * ----------------------------------------------------------------------
 */
static void*
supplier(void* arg)
{
    // TODO: Your code here.



}
```

```
typedef void (*handler_t) (void *);

struct Task {
    handler_t handler;
    void* arg;
};


class Simulation {
    public:
    TaskQueue supplierTasks;
    TaskQueue customerTasks;
    EStore store;

    int maxTasks;
    int numSuppliers;
    int numCustomers;

    explicit Simulation(bool useFineMode) : store(useFineMode) { }
};

class TaskQueue {

    private:
    // you filled in items here during lab 3 but do not need to do so for this
    // problem.

    public:
    TaskQueue();
    ~TaskQueue();

    void enqueue(Task task);
    Task dequeue();

    private:
    int size();
    bool empty();

};
```

Solution elided; too close to lab code.

4. [20 points] This question considers a simplified EStore, called `EStoreSimple`. This version has no notion of discounts, budgets, or valid items. The store simply tracks a list of item quantities, stored in an array called `inventory`; each item is identified by an integer, and its quantity is stored in the corresponding slot in `inventory`.

This store has two methods:

- `BuyTwoItems(int item1_id, int item2_id)`: This method buys *one* unit of `item1_id` and *one* unit of `item2_id`. If either of the items is not available, the method has to wait.
- `AddStock(int item_id, int count)`: This method adds `count` units of item `item_id`.

Fill in where it says to do so on the next page. There are four places to do so.

Some requirements and non-requirements:

- You may assume that there are threads calling these two methods through a single instance of `EStoreSimple`.
- Follow the concurrency commandments. Follow also the coding pattern from the coarse-grained locking section of lab 3.
- The store begins with 0 of each item.
- If progress is possible, threads should not wait or stay in a blocked state; on the other hand, do not worry about needlessly waking threads.

```
class EStoreSimple {
    private:
        int inventory[INVENTORY_SIZE];
        // (1) FILL IN: MORE REQUIRED HERE


    public:
        void BuyTwoItems(int item1_id, int item2_id);
        void AddStock(int item_id, int count);
};



EStoreSimple::EStoreSimple()
{
    // (2) FILL THIS IN
    //
    // Initialize "inventory" and other state.




}



void
EStoreSimple::BuyTwoItems(int item1_id, int item2_id)
{
    // (3) FILL THIS IN




}


void
EStoreSimple::AddStock(int item_id, int count)
{
    // (4) FILL THIS IN




}
```

```
// (1)
class EStoreSimple {
    private:
        int inventory[INVENTORY_SIZE];
        mutex_t m;
        cond_t c;

    ...
}

// (2)
EStoreSimple::EStoreSimple()
{
    memset(inventory, 0, sizeof(inventory));

    /* alt:
     * for (int i = 0; i < INVENTORY_SIZE; i++) {
     *    inventory[i] = 0;
     * }
     */

    mutex_init(&m);
    cond_init(&c);
}

// (3)
void
EStoresim::buyTwoItems(int item1_id, int item2_id)
{
    m.acquire();

    while (inventory[item1_id] == 0 || inventory[item2_id] == 0)
        cv.wait(&m);

    inventory[item1_id]--;
    inventory[item2_id]--;

    m.release();
}

// (4)
void
EStoresim::addStock(int item_id, int count)
{
    m.acquire();

    inventory[item_id] += count;

    cv.broadcast(&m);
```

```
        m.release()
    }
```

IV Deadlock (10 points)

5. [10 points] Alice and Bob each have an account at a bank, and they are the only two people with accounts at this bank. The bank implements `transfer()` as below:

```
850 // assume all the variables are initialized correctly
851 double balance[2]; // 0 for alice, 1 for bob
852 smutex_t mtx[2];    // 0 for alice, 1 for bob
853
854 bool transfer(int from, int to, double trans) {
855   smutex_lock(&mtx[from]);
856   smutex_lock(&mtx[to]);
857
858   bool result = false;
859   if (balance[from] > trans) {
860     balance[from] = balance[from] - trans;
861     balance[to] = balance[to] + trans;
862     result = true;
863   }
864
865   smutex_unlock(&mtx[to]);
866   smutex_unlock(&mtx[from]);
867   return result;
868 }
```

Write down an interleaving that results in deadlock.

```
T1 calls transfer(0, 1, 100)
T2 calls transfer(1, 0, 100)

T1: lock(&mutex[0])
T2: lock(&mutex[1])
T1: try to lock(&mutex[1])
T2: try to lock(&mutex[0])
```

Keeping the same data structures (the `balance` array and the `mtx` array), rewrite `transfer()` to eliminate the possibility of deadlock.

State which lines you are replacing, and give the replacement, in code.

Rewrite lines 855–856 to be:

```
if (from < to) {
    smutex_lock(&mtx[from]);
    smutex_lock(&mtx[to]);
} else {
    smutex_lock(&mtx[to]);
    smutex_lock(&mutex[from]);
}
```

```
/*
 * continue as before. for good style, the unlock() should be
 * done in the corresponding order, though that is not
 * required.
 */
```

Or, if we know that there are only two mutexes, we could replace lines 855–856 with:

```
smutex_lock(&mtx[0]);
smutex_lock(&mtx[1]);
```

V Therac-25 (7 points)

6. [4 points] When the Therac-25 overdosed patients, the physical position of the _____was inconsistent with the machine's settings for _____.

Fill in the two blanks above.

(a) Turntable position and (b) beam type or energy.

7. [3 points] The inconsistency mentioned in the previous question would have been prevented by hardware that the Therac-20 had but was not present in the Therac-25.

What hardware feature was that?

Hardware interlocks

VI Scheduling (8 points)

8. [8 points] Consider a system with three jobs that arrive in the order A, B, C. Jobs A and B each take 1000 seconds, if given sole access to the CPU. Job C takes 10 seconds, if given sole access to the CPU. A context switch takes 5μs (5 microseconds, or 5×10^{-6} seconds).

Consider two scheduling disciplines: (a) FIFO and (b) Round-robin with a quantum of 1 millisecond.

Which of the two will have greater throughput for the stated workload? If they will result in the same throughput, write "same."

FIFO has greater throughput.

Justify your answer below. Use no more than two sentences.

Round-robin has more context switches, and context switches have a price. So the total time to get everything out of the system is (slightly) higher under RR, and therefore the throughput is (slightly) lower. Some students computed the throughput, but it was not necessary to do so to answer this question.

VII Virtual memory (19 points)

9. [7 points] Assume an architecture with a virtual address of 32 bits and pages of 4 KB.

How many virtual pages are in the system? Show your work.

2^{20} pages. If pages are 4 KB, then there are 12 offset bits. The remaining bits range over all possible pages. There are 32-12=20 such bits, and 2^{20} possible settings. Another way to show work would have been to draw a picture of the address, separate it into two rectangles, one of "length" 12 bits and one of "length" 20 bits.

10. [12 points] Assume our model architecture, x86-64. The operating system wants, for a particular process *p*, to have virtual address `0x403000` map to physical address `0x5000`, and for this virtual address to be accessible in user space, and writable.

To avoid ambiguity, number indexes starting at 0 (index 0, index 1, and so on). Do not use words like "first, second, third" and so on; those words are ambiguous in this context.

To answer the questions below, it may be helpful for you to draw pictures (but you don't have to).

In the L1 page table for process *p*, what is the index of the entry that is relevant to the specified mapping? What does this entry contain?

Index 0 is the relevant one. It contains the physical page number of an L2 page table, and the three bottom permissions bits are set.

In the relevant L2 page table, what is the index of the entry that is relevant to the specified mapping? What does this entry contain?

Index 0 is the relevant one. It contains the physical page number of an L3 page table, and the three bottom permissions bits are set.

In the relevant L3 page table, what is the index of the entry that is relevant to the specified mapping? What does this entry contain?

Index 2 is the relevant one. It contains the physical page number of an L4 page table, and the three bottom permissions bits are set.

In the relevant L4 page table, what is the index of the entry that is relevant to the specified mapping? What does this entry contain?

Index 3 is the relevant one. It contains 0x5 as the physical page number for the mapping, and the three permissions bits are set.

VIII Feedback (2 points)

11. [2 points] This is to gather feedback. Any answer, except a blank one, will get full credit.

Please state the topic or topics in this class that have been least clear to you.

Please state the topic or topics in this class that have been most clear to you.

End of Midterm
Enjoy Spring Break!!