CS 3214 Fall 2023 Midterm

October 31, 2023

- This is a closed-book, closed-internet, closed-cell/smart phone or watch and closed-computer exam. However, you may refer to your sheet of prepared notes.
- Your exam should have 12 pages with 4 questions totaling 100 points. You have 75 minutes. Please write your answers in the space provided on the exam paper.
- If you finish the exam early you are expected to leave the room very quietly. If you finish within 15 minutes or less before the end of the allotted time, please stay in your seat until the end.
- Answers will be graded on correctness and clarity. The space in which to write answers to the questions is kept purposefully tight, requiring you to be concise. You may lose points if your solution is more complicated than necessary or if you provide extraneous information along with a correct solution.

Name (printed) ______________________ PID ______________________`@vt.edu`

Section: ☐ Dr. Back ☐ Dr. Butt ☐ Dr. Williams

I accept the letter and the spirit of the Virginia Tech undergraduate honor code — I have not given or received aid on this exam.

(signed) ____

You are expected to keep the content of this exam secret until told otherwise by your instructor. Please do not start until instructed to do so.

#	Problem	Score
1	Operating System Concepts (20 pts)	
2	Unix Processes and IPC (23 pts)	
3	Multithreading (38 pts)	
4	Development and Linking (19 pts)	
	Total	

1 Operating System Concepts (20 pts)

1.1 Basic OS Ideas (10 pts)

Check if the following statements are true or false.

a) One of the major tasks of an operating system is the protection and isolation of different processes from each other and from the system's kernel.

☐ true / ☐ false

b) Stable OS interfaces allow us to write code that once compiled, runs on different machines with the same or a compatible instruction set architecture (ISA) and can interface with its environment.

☐ true / ☐ false

c) Dual-mode operation refers to the permission system resulting from having unprivileged users (e.g. your account on rlogin) and privileged admin users, i.e., admin mode vs user mode.

☐ true / ☐ false

d) It is possible for two processes to use the same addresses for their variables without conflicting with each other.

☐ true / ☐ false

e) Because the OS kernel is usually a program systems designers trust to be correct, defending against vulnerabilities in kernel code is substantially harder than defending against vulnerabilities in user mode.

☐ true / ☐ false

1.2 On Process States and Scheduling (10 pts)

In class, we discussed how to model the execution of processes using a simplified process state model. Determine whether the following statements are true or false.

a) To help prevent attacks, on most OS, the number of processes that are currently in the `RUNNING` state is a closely guarded secret accessible only to system administrators.

☐ true / ☐ false

b) System calls such as `read()` frequently transition a process into the `BLOCKED` state.

☐ true / ☐ false

c) OS are designed to avoid situations in there are some `READY` processes assigned to a CPU, but no `RUNNING` process.

☐ true / ☐ false

d) On a well-balanced machine with n CPUs (or cores) a user controlling the machine would typically find n processes in the `READY` and n processes in the `RUNNING` state.

☐ true / ☐ false

e) A laptop's battery life is heavily related to how much blocking is encountered on the machine, that is, the time-averaged number of `BLOCKED` processes in the system.

☐ true / ☐ false

2 Unix Processes and IPC (23 pts)

2.1 Of Pipes and Tees (8 pts)

In lecture, we had discussed how Unix pipes work: among other uses, they allow one program to take their input from another program's output. The program's output may in turn become the input of yet another program, and so on. Sometimes, it is desirable to save a copy of all data that flowed through one of these pipes. To do that, Unix introduced the program `tee`. It works like so. Suppose we want to count on how many lines the word `fopen` occurs in `T.c`. We could run

```
$ grep fopen T.c | wc -l
1
```

and obtain the answer 1. If we run

```
$ grep fopen T.c | tee pipe.log | wc -l
1
```

(we insert a 'T' into the pipe), a file `pipe.log` is created with this content:

```
$ cat pipe.log
   FILE *f = fopen(av[1], "w");
```

Implement `tee` in a language of your choice. For the purposes of this problem, we will ignore the distinction between byte streams that represent a valid character encoding and those that do not. Put another way, your implementation of `tee` may assume that only ASCII data is being sent through the pipe into whose place your implementation of the `tee` command is inserted. Even though the question does not ask for a specific language, pseudocode is not accepted.

2.2 What Did the User Type? (15 pts)

Consider the following excerpts of 3 system call traces that were taken from Dr. Back's cush implementation:

```
1  Excerpt from cush's strace:
2  pipe2([4, 5], O_CLOEXEC)                = 0
3  clone(child_stack=0x14ce70b78210, flags=CLONE_VM|CLONE_VFORK|SIGCHLD) = 604094
4  clone(child_stack=0x14ce70b78210, flags=CLONE_VM|CLONE_VFORK|SIGCHLD) = 604095
5  close(5)                                = 0
6  close(4)                                = 0
7  wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 1}], WSTOPPED, NULL) = 604094
8  wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], WSTOPPED, NULL) = 604095
9
10 Excerpt from strace of process 604094:
11 setpgid(0, 0)                           = 0
12 dup2(5, 1)                              = 1
13 close(5)                                = 0
14 dup2(1, 2)                              = 2
15 execve("/usr/bin/gcc", ["gcc", "-v", "badfile.c"], 0xb1af40 /* 48 vars */) = 0
16
17 Excerpt from strace of process 604095:
18 setpgid(0, 604094)                      = 0
19 dup2(4, 0)                              = 0
20 close(4)                                = 0
21 execve("/usr/bin/grep", ["grep", "error"], 0xb1af40 /* 48 vars */) = 0
```

Note that these are given in order, but without timestamps, so nothing should be assumed about how much time has passed between calls. Answer the following questions:

a) (2 pts) What do the calls on lines 3 and 4 do?

b) (2 pts) Why are the calls to `close` on lines 5 and 6 necessary?

c) (2 pts) What does line 14 do?

d) (2 pts) What is the combined effect of lines 11 and 18? (Recall that `setpgid(a, b)` sets process `a`'s process group to `b`. If `a` is zero, it refers to the current process. If `b` is zero, a new process group is created.)

e) (5 pts) What could the user have typed into cush to produce this strace?

f) (2 pts) Can you tell based on the excerpts whether the user started a foreground or a background job? Justify your answer.

3 Multithreading (38 pts)

3.1 What Abstraction is it Anyway? (14 pts)

Consider the following four real-world scenarios (not all equally realistic). For scenario, identify which abstraction it illustrates, **specifically within the context of multithreaded programming**. If the abstraction has multiple parts, name each one.

a) (3 pts) An elementary school classroom has a hallpass system that works like this: when a student needs to use the bathroom, they check if the hallpass is available. If so, they take it, use the bathroom, and return it. Students have to wait for the hallpass to be returned in order to use the bathroom.

b) (3 pts) A group of students is fundraising for the school band. They decide to meet at one student's house who lives in the neighborhood. They then take a map of the neighborhood and assign one street to each student. Then the students leave the meeting house to canvas their assigned street. Once a student is done with their assigned street, they'll return to the meeting house from which they started and hands their collected money to the parent of the student who lives there, who will then send the money to the band.

c) (5 pts) A person lives in a house with a particular kind of doorbell. When pressed, it will ring only in the person's bedroom and is not heard anywhere else. The person is expecting a package, but they are also tired and would like to catch up on sleep, so they devised the following system. They install a remote-controlled gate on their porch which they can operate from their bedroom. They then proceed as follows: they lock the gate (so that the delivery person cannot get onto the porch to leave packages), then they check if a package perhaps was already left (in case the delivery person had already arrived earlier). If no package was already left, they go their bedroom and remotely release the gate lock. Once in their bedroom, they will be able to hear the doorbell when the delivery occurs. Every so often, they are woken up by the doorbell, go to their front door, but don't see a package. (This may happen if their neighbor saw the delivery person after they dropped off a package and took the package for safe keeping before the homeowner had a chance to hit the remote lock button for their porch gate after hearing the doorbell.)

____

d) (3 pts) Two hikers take different paths up a mountain. The two paths cross at one point. One hiker is typically slower than the other, but they can't be certain who will reach the crossing point first. They agree on the following protocol: if the faster hiker will reach the crossing point first, they will wait there for the slower hiker. If the slower hiker will reach the crossing point first, they will leave a marker of 3 rocks and move on. If the faster hiker arrives at the crossing point later and sees the marker of rocks, they know that the slower hiker was actually there before them and they will move on (after destroying the marker).

____

3.2 Data Race or Not (8 pts)

Consider the C program on the next page

```c
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdint.h>
5  const int N_THREADS = 4;
6
7  static int X[4];
8
9  static void*
10 thread_func(void *_arg)
11 {
12     uintptr_t myindex = (uintptr_t) _arg;
13     X[myindex] = 42-myindex;
14     return NULL;
15 }
16
17 int
18 main()
19 {
20     pthread_t t[N_THREADS];
21     for (uintptr_t i = 0; i < N_THREADS; i++) {
22         pthread_create(t+i, NULL, thread_func, (void *) i);
23     }
24
25     int s = 0;
26     for (int i = 0; i < N_THREADS; i++) {
27         pthread_join(t[i], NULL);
28         s += X[i];
29     }
30     printf ("%d\n", s);
31 }
```

a) (6 pts) The C11 memory model defines a data race as follows:

> *When an evaluation of an expression writes to a memory location and another evaluation reads or modifies the same memory location, the expressions are said to conflict. A program that has two conflicting evaluations has a data race unless either*
>
> - *both conflicting evaluations are atomic operations*
> - *one of the conflicting evaluations happens-before another*

Identify all pairs of conflicting evaluations in this code. Specify, for each pair,

i) the memory location

ii) which threads are involved (call them T1, T2, T3, T4, and Main), and

iii) for each thread, on which line number the conflicting evaluation occurs for that particular thread and whether it is a read or write

b) (2 pts) Does the program contain a data race under the above definition? Justify why or why not.

3.3 How Not To Use Condition Variables (3 pts)

Consider the following excerpt from a student's p2 implementation:

```
pthread_mutex_lock(&pool->mutex);
pthread_cond_wait(&pool->cond, &pool->mutex);
pthread_mutex_unlock(&pool->mutex);
```

Describe one way in which this code will fail *independent of* the remainder of the code, that is, independent of whether this snippet is part of some larger loop or not.

3.4 Semaphore Puzzle (8 pts)

For this exam's semaphore puzzle, we will provide the answer: it's either TRICK or TREAT. Your job is to complete a multithreaded program whose *only* possible outputs consist of these two strings.

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t s1, s2, s3, s4;
char *tail;

static void* thread_T(void *_)
{
    printf("T");

    return NULL;
}

static void* thread_R(void *_)
{

    printf("R");


    return NULL;
}

static void* thread_I(void *_)
{


    tail = "ICK";


    return NULL;
}

static void* thread_E(void *_)
{


    tail = "EAT";


    return NULL;
}

static void* thread_tail(void *_)
{


    printf("%s \n ", tail);
    return NULL;
}

int main()
{
    sem_init(&s1, 0, 0);
    sem_init(&s2, 0, 0);
    sem_init(&s3, 0, 0);
    sem_init(&s4, 0, 1);    // note semaphore 4 has initial value 1
    const int N_THREADS = 5;
    void * (*f[])(void *) = { thread_T, thread_R, thread_I, thread_E,
                              thread_tail };

    pthread_t t[N_THREADS];
    for (int i = 0; i < N_THREADS; i++)
        pthread_create(t+i, NULL, f[i], NULL);

    for (int i = 0; i < N_THREADS; i++)
        pthread_join(t[i], NULL);
}
```

Fill in the necessary statements directly above. No other changes shall be made to the program, and you may not remove any statements. We provided vertical space proportional to the number of statements expected (which is a total of 14 lines needing statements). Pay particular attention to semaphore s4, which has an initial value of 1. Your program must be data race free. Make sure that all threads in the program finish, i.e., none remains blocked.

3.5 Breaking Up Locks (5 pts)

A common bit of performance advice is to start a multithreaded design with a single, global lock, breaking it up only when necessary.

i) (2 pts) Describe briefly when it would be necessary to "break up" a lock.

ii) (3 pts) What activities does "breaking up" a lock mean in practice when you apply it to your design? Provide a very brief description.

4 Development and Linking (19 pts)

4.1 Compilation, Linking, and Symbol Tables (13 pts)

Consider the following program:

```
$ cat prog.c

extern int add_10(int x);
int x = 32;
static int *y = &x;

int main(int argc, char **argv) {
    return add_10(*y);
}
```

a) suppose you typed ‘`gcc -c prog.c`‘.

   i) (2 pts) What parts of the compilation toolchain ran? (preprocessor, compiler, assembler, linker)

   ii) (2 pts) What filename would be generated?

   iii) (4 pts) What symbols would the linker see in the generated file and what type are they (e.g., the output of ‘`nm`‘)?

b) suppose you typed ‘`gcc prog.c`‘.

   i) (2 pts) What parts of the compilation toolchain ran? (preprocessor, compiler, assembler, linker)

   ii) (3 pts) What output would you expect on the shell?

4.2 Linking and Scope (6 pts)

Consider the following header file `mod.h`:

```
static int x = 42;
extern void decX(void);
extern void decY(void);
extern void dec(int *);
extern int y;
```

and the C source files `mod1.c` and `mod2.c`:

```
// mod1.c
#include "mod.h"
int y = 42;

void dec(int *p) {
    (*p)--;
}

void decX(void) {
    x--;
}

void decY(void) {
    y--;
}
```

```
// mod2.c
#include <stdio.h>
#include "mod.h"

int
main()
{
    printf ("y = %d\n", y);
    dec(&y);
    printf ("y = %d\n", y);
    printf ("x = %d\n", x);
    decX();
    printf ("x = %d\n", x);
    dec(&x);
    printf ("x = %d\n", x);
}
```

Suppose the user compiles and links those files into an executable What will the resulting program output?