New York University

CSCI-UA.202: Operating Systems (Undergrad): Fall 2023

Midterm Exam

SOLUTIONS

- **Write your name and NetId 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 what is in 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 the exam print-out and blue books; we will not collect it.* 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 **14** 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.
- 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/12)	II (xx/28)	III (xx/31)	IV (xx/29)	Total (xx/100)

I Fundamentals

1. [2 points] What does the `yield` system call do?

Return control to the scheduler, allowing another thread to run.

2. [2 points] When debugging some code you see an instruction `movq 0x77ff00101200, %rbx`. Is the address `0x77ff00101200` a physical address or a virtual address?

Virtual address.

3. [4 points] Which of the following calls are blocking, that is, after which of the following calls can the calling thread be blocked (you should list all options that apply):

(a) `smutex_unlock`

(b) `smutex_lock`

(c) `scond_wait`

(d) `scond_signal`

`smutex_lock` and `scond_wait`

4. [4 points] In class, we discussed three traps, which could cause a processor executing in user mode to switch to kernel mode. Name any two of these.

System calls, exceptions and interrupts.

II Processes and Stack Frames

5. [8 points] Below is a C program that calls into a function `g` written in assembly. As a reminder, the compiler does not add a prolog or epilog to functions written in assembly, and pushing values on the stack decrements the stack pointer.

```
1  #include <stdio.h>
2
3  void h() {
4      printf("In h\n");
5      exit(0);
6  }
7
8  void f() {
9      g();
10     printf("Done with g\n");
11 }
12
13
14 int main(int argc, char* argv[]) {
15     f();
16     return 0;
17 }
18
19 g:
20     movq h, 8(%rbp) # Copy address of h to %rbp+8.
21     ret
```

What does the program above print? Provide a brief (2 sentences or less) explanation for your answer.

```
Done with g
In h
```

`g` overwrites `f`'s return address (which is at `%rbp + 8`to `h`'s address, and thus control returns from `g` to `f` to `h`.

6. [10 points] In the C program below, we use `uint64_t` to represent an unsigned 8-byte integer, and print such values using the format string `%llu`.

```
1  #include <stdio.h>
2  uint64_t* f() {
3      uint64_t x = 7; // sizeof(x) == 8.
4      return &x;
5  }
6
7  void g() {
8      uint64_t y = 2; // sizeof(y) == 8.
9      printf("y in g has value %llu\n", y);
10 }
11
12 int main(int argc, char* argv[]) {
13     uint64_t *t = f();
14     uint64_t t0, t1;
15     t0 = *t;
16     g();
17     t1 = *t;
18     printf("t in main has value %llu, was %llu\n", t1, t0);
19     return 0;
20 }
```

Answer the following two questions based on the program above:

(a) **What does the program (as written) print when executed? Provide a brief (2 sentences or less) explanation.**

(b) **What bug, if any, does the program contain? Identify the function and line where the bug occurs.**

(a) `y in g has value 2`
`t in main has value 2, was 7`

Function `f` returns a pointer to a stack variable, which gets overwritten by `g`, and thus becomes 2.

(b) `g` returns a pointer to `x`, a local variable on the stack.

7. [10 points] As we discussed in class, file descriptor 1 is "standard output" (stdout) in Unix, and in most cases (e.g., when you run programs from the shell) any writes to this file descriptor show up on the screen (in a console). Furthermore, the prototypes for the `open` and `write` system calls are:

`int write(int fd, const void* data, size_t size)`: writes `size` bytes, starting from `data` to the supplied file descriptor `fd`.

`int open(const char* fname, int flags, mode_t mode)`: opens file `fname` with access dictated by `flags`.

```
1  #include <stdio.h>
2  int main(int argc, char* argv[]) {
3      int iters = 5;
4      if (fork() == 0) {
5          // Change file descriptor 1 so it points to the file `/tmp/out`:
6          // (a) Open "/tmp/out" for writing, creating it if it does not exist.
7          // When this call returns, `fd` contains the file descriptor
8          int fd = open("/tmp/out", O_WRONLY | O_CREAT, 0666);
9          // (b) Close file descriptor 1.
10         close(1);
11         // (c) Change file descriptor 1 so it points to /tmp/out
12         dup2(fd, 1);
13
14         iters = 20;
15     }
16
17     for (int i = 0; i < iters; i++) {
18         write(1, "counting\n", 9);
19     }
20     return 0;
21 }
```

Carefully consider the code above, and answer the following questions:

(a) **How many lines are printed on screen?**

(b) **How many lines are added to `/tmp/out`?**

(a) 5 lines. Lines 5–14 do not execute for the original process, so it prints to the screen and `iters = 5`.

(b) 20 lines. Lines 5–14 execute in the forked process.

III Concurrency

8. [15 points] In what follows, as in class, we assume that threads are pre-emptively scheduled. Consider the code below:

```
1  #include <stdio.h>
2  #include "sthread.h"
3
4  class Interleave {
5  public:
6      void f();
7      void g();
8      Interleave();
9  private:
10     // ADD ANY VARIABLES YOU NEED HERE
11 };
12
13 void Interleave::f()
14 {
15     // MODIFY THIS FUNCTION
16     while (1) {
17         printf("f");
18     }
19 }
20
21 void Interleave::g()
22 {
23     // MODIFY THIS FUNCTION
24     while (1) {
25         printf("g");
26     }
27 }
28
29 Interleave::Interleave()
30 {
31     // INITIALIZE ANY VARIABLES YOU NEED HERE
32 }
33
34 // No changes are necessary to this function,
35 // it starts executing the Interleave::f() function.
36 void* start_f(void* arg)
37 {
38     Interleave* i = (Interleave*)arg;
39     i->f();
40     return NULL;
41 }
42
43 // No changes are necessary to this function,
44 // it starts executing the Interleave::g() function.
45 void* start_g(void* arg)
```

```
46 {
47     Interleave* i = (Interleave*)arg;
48     i->g();
49     return NULL;
50 }
51
52
53 int main(int argc, char* argv[]) {
54     Interleave i;
55     sthread_t f, g;
56     sthread_create(&f, start_f, &i);
57     shtread_create(&g, start_g, &i);
58     sthread_join(f);
59     sthread_join(g);
60 }
```

In the code above, **use monitors to interleave the output from functions `f` and `g`**, that is use monitors to ensure that each `f` printed on the screen is followed by a `g`, and each `g` is followed by a `f`. Your code must first output a `f`, that is your output must look like:
`fgfgfgfgfgfgfgfgf`...

You can ignore destructors for this problem, since the program never exits.

```
1  #include <stdio.h>
2  #include "sthread.h"
3
4  class Interleave {
5  public:
6      void f();
7      void g();
8      Interleave();
9  private:
10     smutex_t mutex;
11     scond_t wait_next;
12     int current_state;
13 };
14
15 void Interleave::f()
16 {
17     smutex_lock(&mutex);
18     while (1) {
19         while (current_state%2 != 0) {
20             scond_wait(&wait_next, &mutex);
21         }
22         printf("f");
23         current_state++;
24         scond_signal(&wait_next, &mutex);
25     }
26     smutex_unlock(&mutex);
```

```
27 }
28
29 void Interleave::g()
30 {
31     smutex_lock(&mutex);
32     while (1) {
33         while (current_state%2 != 1) {
34             scond_wait(&wait_next, &mutex);
35         }
36         printf("g");
37         current_state++;
38         scond_signal(&wait_next, &mutex);
39     }
40     smutex_unlock(&mutex);
41 }
42
43 Interleave::Interleave()
44 {
45     smutex_init(&mutex);
46     scond_init(&wait_next);
47     current_state = 0;
48 }
49
50 // No changes are necessary to this function,
51 // it starts executing the Interleave::f() function.
52 void* start_f(void* arg)
53 {
54     Interleave* i = (Interleave*)arg;
55     i->f();
56     return NULL;
57 }
58
59 // No changes are necessary to this function,
60 // it starts executing the Interleave::g() function.
61 void* start_g(void* arg)
62 {
63     Interleave* i = (Interleave*)arg;
64     i->g();
65     return NULL;
66 }
67
68
69 int main(int argc, char* argv[]) {
70     Interleave i;
71     sthread_t f, g;
```

```
72      sthread_create(&f, start_f, &i);
73      shtread_create(&g, start_g, &i);
74      sthread_join(f);
75      sthread_join(g);
76  }
```

9. [10 points] The class `NumberWaiter`, shown below, uses monitors to allow threads to wait for a specific value to be set. In particular, threads can call `set_number(i)` to set the current number, or call `wait_number(i)` to block until the current number is equal to `i`. Any number of threads should be able to safely use these function.

Unfortunately, we introduced a few bugs when writing this code. **Please identify all bugs in the code below and provide fixes for them. You can use the line numbers in the listing to specify where bugs occur, and provide code snippets to show how we fix them.**

```
1   class NumberWaiter {
2   public:
3       NumberWaiter();
4       ~NumberWaiter();
5       void set_number(int i);
6       void wait_number(int i);
7   private:
8       smutex_t mutex;
9       scond_t number_changed;
10      int number;
11  };
12
13  NumberWaiter::NumberWaiter() {
14      number = 0;
15      smutex_init(&mutex);
16      scond_init(&number_changed);
17  }
18
19  NumberWaiter::~NumberWaiter() {
20      smutex_destroy(&mutex);
21      scond_destroy(&number_changed);
22  }
23
24  // Update `number` to the input value.
25  void NumberWaiter::set_number(int i) {
26      smutex_lock(&mutex);
27      number = i;
28      smutex_unlock(&mutex);
29      scond_signal(&number_changed, &mutex);
30  }
31
32  // Block until `number == i`, where `i` is the
33  // function's input.
```

```
34 void NumberWaiter::wait_number(int i) {
35     smutex_lock(&mutex):
36     if (number != i) {
37         scond_wait(&number_changed, &mutex);
38     }
39     smutex_unlock(&mutex);
40 }
```

```
1  // Update ‘number‘ to the input value.
2  void NumberWaiter::set_number(int i) {
3      smutex_lock(&mutex);
4      number = i;
5      scond_broadcast(&number_changed, &mutex);
6      smutex_unlock(&mutex);
7  }
8
9  // Block until ‘number == i‘, where ‘i‘ is the
10 // function's input.
11 void NumberWaiter::wait_number(int i) {
12     smutex_lock(&mutex):
13     while (number != i) {
14         scond_wait(&number_changed, &mutex);
15     }
16     smutex_unlock(&mutex);
17 }
```

10. [6 points] You are writing an application that acts as a bank: a user can deposit or withdraw money, or transfer them between bank accounts. You represent bank accounts in a large array, and your application consists of multiple threads that manipulate the accounts in this array. To maximize concurrency, each account has its own lock. When depositing or withdrawing money from an account A, your program acquires A's lock and then performs the appropriate operation.

However, transferring money from account A to B requires acquiring A's lock and B's lock. **Briefly (in 2 sentences) state what do you do to avoid deadlocks in this scenario.**

Acquire locks in the same order: for example, acquire the lock for the account for the lower index in the array first.

IV Scheduling

11. [12 points] A system with a single processor needs to execute two batch tasks (threads):

Task A that completes in 1000 ms.

Task B that completes in 600 ms.

What is the average turnaround time when:

(a) The tasks are scheduled using a round-robin scheduler with scheduling quanta of 10ms.

(b) The tasks are scheduled using shortest time to completion first (STCF).

(c) The tasks are scheduled using first-come first-served (FIFO/FCFS), with task A arriving before task B.

(a) Task B takes 1200ms, Task A takes $1200 + 400 = 1600$ms. Average is 1400ms.

(b) Task B takes 600ms, Task A takes $600 + 1000 = 1600$ms. Average is 1100ms.

(c) Task A takes 1000ms, Task B takes $1000 + 600 = 1600$ms. Average is 1300ms.

12. [6 points] Multi-level feedback queuing (MLFQ) aims to avoid starvation when scheduling tasks (threads or processes) which have different priorities. Explain briefly, in two or three sentences how it does so? It uses feedback to boost the priority of task (e.g., low-priority tasks) that have not run recently.

13. [10 points] The following table lists processes, their arrival time and completion time (in abstract time units). Assume we are scheduling these processes using STCF.

Process	Arrival Time	Completion Time (Burst Time)
A	0	20
B	1	4
C	2	1
D	3	2

State what process is running at each of the following time steps:

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10
A	B	C	D	D	B	B	B	A	A	A

14. [1 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