2023 Fall Final Examination

Course: ECE344: Operating Systems
Examiner: Jon Eyolfson
Date: December 8, 2023
Duration: 2 hours 30 minutes (150 minutes)

Exam Type: C3

A "closed book" examination permits the use of a single 8.5″×11″ aid sheet. Both sides of the sheet may be filled with any relevant information, including handwritten notes, typewritten text, images, or other formats, as long as they fit within the sheet's dimensions.

Calculator Type: 3

Non-programmable calculators from a list of approved calculators as issued by the Faculty Registrar.

Instructions:

Do not write answers on the back of pages as they will not be graded. Use the blank sheet at the end of the exam for extra space, and clearly indicate in the provided answer space if your response continues.

If a question seems unclear or ambiguous, state your assumptions and answer accordingly. In case of an error, identify it, provide a corrected version, and respond as if the question has been fixed.

Provide brief and specific answers. Clear and concise responses will receive higher marks compared to vague and wordy ones. Note that marks will be deducted for incorrect statements in your answers.

Short Answer (14 marks total)

Q1 (2 marks). When you press `Ctrl+C` in a shell, what happens to a process and does it always terminate? Provide a general explanation without referencing specific names or numbers.

The shell sends a signal to the process (specifically `SIGINT`, but not needed). The process may not always terminate, it could ignore the signal.

Q2 (3 marks). Your friend decides to create a macrokernel that incorporates a web server. What is one advantage and one disadvantage to this approach?

One advantage would be performance, it would not need to do any system calls since it's already running in kernel mode.

One disadvantage is that any exploit in the web server would allow the attacker to run in kernel mode and have full control of the machine.

Q3 (3 marks). Comparing two scenarios where you have 4 disks configured in RAID 1 and RAID 5, respectively: if a single disk fails and is replaced, which RAID configuration would recover faster and why?

RAID 1 would recover faster, because we could simply copy a third of the data from the 3 remaining disks. RAID 5 would have to read all the contents from the remaining disks and reconstruct the data (or compute the parity).

Q4 (3 marks). What is the outcome of executing the given code snippet?

```
1  ucontext_t context;
2  getcontext(&context);
3  setcontext(&context);
```

It would be an infinite loop, constantly going from line 2 to 3, then 3 to 2.

Q5 (3 marks). Is it possible for a buddy allocator, managing 4096 bytes of memory, to successfully make three allocations of 1300 bytes each? Justify your answer.

No, each allocation would need a 2048 byte block (they cannot fit on a 1024 byte block). This buddy allocator would only have 2 2048 byte blocks, so the third allocation would fail.

Processes (22 marks total)

Consider the following code, assuming that all system calls (except for `read`) are always successful, running under process ID (`pid`) 100:

```c

int fds[2];

void read_int(void) {
    int x;
    read(fds[0], &x, sizeof(int));
    printf("%d: read %d\n", getpid(), x);    /* (A) */
}

void write_int(int x) {
    write(fds[1], &x, sizeof(int));
}

int create_process(void) {
    int pid = fork();
    if (pid > 0) {
        return pid;
    }
    read_int();
    exit(0);
}

bool is_end_of_input(void) {
    int x;
    return read(fds[0], &x, sizeof(int)) == 0;
}

int main(void) {
    pipe(fds);

    int child1 = create_process();
    int child2 = create_process();

    write_int(42);
    write_int(1337);
    close(fds[1]);    /* (D) */

    waitpid(child1, NULL, 0);    /* (C1) */
    waitpid(child2, NULL, 0);    /* (C2) */

    assert(is_end_of_input());    /* (B) */

    return 0;
}
```

Q6 (2 marks). How many (new) processes get created? Write down the `pid`s they would likely get.

2, `pid` 101 and 102.

Q7 (1 marks). What's the term used to describe the transfer of bytes or data between different processes?

IPC or inter-process communication

Q8 (2 marks). Can process 100 have zombie processes when it terminates (assuming it terminates)? Explain why or why not.

No, in this case it wouldn't have any zombie processes when it terminates. It only has 2 child processes, and we `waitpid` on both of them. They'll complete their `read` system calls eventually after process 100 completes its `write`.

Q9 (3 marks). Can two separate processes both read the value 42? Explain the reasoning behind your answer. Your answer should include the term: "global open file table".

No, they cannot. Both children share the same global open file table entry through their file descriptors. A `read` in one would update the position in the other. Therefore, we would not re-read the same data.

Q10 (5 marks). Using your previous `pid` values. Write **all** valid outputs of running process 100 to completion. The only output comes from line 6, which includes the comment: `/* (A) */`.

```
101: read 42
102: read 1337
```

```
102: read 42
101: read 1337
```

```
102: read 1337
101: read 42
```

```
101: read 1337
102: read 42
```

(Note, only a minor deduction for not including the last two outputs, as a context switch must happen between the `read` and `printf`)

Q11 (3 marks). Consider the `is_end_of_input` function call on line 41, marked with the comment `/* (B) */`. What are the possible outcomes of this function call with the program provided? Briefly explain each of the possibilities.

> This function call will always return `true`. Both child processes only `read` to the "read" end of the pipe, and consume all the data written to it. Afterwards they'll eventually terminate and also close the "write" end of the pipe. Process 100 waits for both children to finish after closing its "write" end of the pipe. No

Q12 (3 marks). Consider a modified program where we **only** remove the calls to `waitpid` on lines 38 and 39, marked with comments `/* (C1) */` and `/* (C2) */`.

Again, consider the `is_end_of_input` function call on line 41, marked with the comment `/* (B) */`. What are the possible outcomes of this function call with the modified program? Briefly explain each of the possibilities. You may refer to your previous answer if its outcome is still possible.

> The situation above may happen, it may still return `true`. However, it may now return `false`. Process 100 could write both **`int`** values, then before either child process runs, it could `read` one of the **`int`** values, which would read 4 bytes (not needed in the answer), and return `false`.

Q13 (3 marks). Consider a modified program where we **only** remove the call to `close` on line 36, marked with comment `/* (D) */`. Note, for this question, we still have the calls to `waitpid` present.

Again, consider the `is_end_of_input` function call on line 41, marked with the comment `/* (B) */`. What are the possible outcomes of this function call with the modified program? Briefly explain each of the possibilities. You may refer to your previous answer if its outcome is still possible.

> We will never see the function return `true` or `false`, instead it will get blocked forever. Both child processes will `read` different values and terminate. Now, the `read` in process 100 (in `is_end_of_input`) will block forever. Process 100 still has the "write" end of the pipe open, so more output may be possible, and process 100 blocks.

Threads (15 marks total)

Consider the following code:

```c
pthread_t threads[2];
int x = 0;

void* t0(void*) {
    int y = 3;
    pthread_join(threads[1], NULL);
    x += y;
    printf("t0: x = %d, y = %d\n", x, y); /* (A) */
    return NULL;
}

void* t1(void*) {
    int y = 2;
    x += y;
    printf("t1: x = %d, y = %d\n", x, y); /* (B) */
    return NULL;
}

int main(void) {
    x = 1;
    pthread_create(&threads[0], NULL, t0, NULL);
    pthread_create(&threads[1], NULL, t1, NULL);
    pthread_join(threads[0], NULL);
    printf("main: x = %d\n", x); /* (C) */
    return 0;
}
```

Q14 (6 marks). For the given multi-threaded program, determine if there is a specific order in which the threads execute. Explain the possible sequence of threads that could be running at each key point in the program.

The main thread creates thread 0, then thread 1. The main thread then joins on the thread running `t0`. There's two scenarios:

1. Main thread finishes creating thread 0 and thread 1 first.
2. Main thread creates thread 0, the OS context switches immediately, thread 0 gets an error from the join.

In the first case: either `t0` or `t1` could execute next, but `t0` only creates a local variable and joins `t1`. Now `t1` is the only thread that can run, it runs until completion.

In the second case: there's no set order between the threads.

(Note, the intent of the question was to only have the first case. The calls to `pthread_create` should be switched around.)

Q15 (6 marks). Identify all potential outputs this program may generate when executed.

In the case the main thread creates two threads first, there is only one possible sequence of outputs:

```
t1: x = 3 y = 2
t0: x = 6, y = 3
main: x = 6
```

If we consider the case where the main thread creates thread 0, and thread 0 executes with an error in `pthread_join`:

```
t0: x = 4, y = 3
t1: x = 6 y = 2
main: x = 6
```

```
t1: x = 3 y = 2
t0: x = 4, y = 3
main: x = 3
```

```
t0: x = 4, y = 3
t1: x = 3 y = 2
main: x = 3
```

```
t1: x = 3 y = 2
t0: x = 4, y = 3
main: x = 4
```

```
t0: x = 4, y = 3
t1: x = 3 y = 2
main: x = 4
```

(Note: identifying the first case was 5 marks.)

Q16 (3 marks). Assess if there is a data race in the provided code. Explain your reasoning, highlighting any lines of the code where shared resources may be accessed concurrently without proper synchronization.

In the case the main thread creates two threads first: no, there's not a data race. The only shared resource is `x` and only one thread can execute at any time when we read or write `x`.

In the second case, where we get an error from `pthread_join` then there is a data race. There are two concurrent accesses to `x` with at least one being a write.

(Note: as long as it was clear what case you were considering, full marks.)

Locking (18 marks total)

Examine the provided code where two threads, executing t1 and t2 respectively, each acquire and release multiple mutexes. We have three mutexes in total: m1, m2, and m3.

```c
static pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t m3 = PTHREAD_MUTEX_INITIALIZER;

void* t1(void*) {
    pthread_mutex_lock(&m2);
    pthread_mutex_lock(&m1);
    /* Some computation doing reads and writes. */
    pthread_mutex_unlock(&m1);
    pthread_mutex_unlock(&m2);
    return NULL;
}

void* t2(void*) {
    pthread_mutex_lock(&m1);
    pthread_mutex_lock(&m3);
    /* Some computation doing reads and writes. */
    pthread_mutex_unlock(&m3);
    pthread_mutex_unlock(&m1);
    return NULL;
}
```

Q17 (6 marks). Analyze if the two threads executing functions t1 and t2 in the provided code can result in a deadlock. Explain your reasoning behind whether a deadlock is possible or not in this scenario

This code cannot deadlock. We share m1 between the two threads, however there is not a circular wait between them. The other locks are used only one per thread, so we will not cause a deadlock.

Q18 (12 marks).
Assuming the addition of a third thread, Thread 3, executing function t3, devise a sequence of lock calls involving mutexes m2 and m3 that could lead to a deadlock. Describe this sequence and explain how it results in a deadlock when combined with the lock calls in t1 and t2.

```
void* t3(void*) {




    return NULL;
}
```

We could add the following code:

```
void* t3(void*) {
    pthread_mutex_lock(&m3);
    pthread_mutex_lock(&m2);
}
```

We could now lead to a deadlock.

Consider t1 locks m2, then

t2 locks m1, then

t3 locks m3

Now, no thread can make progress.

t1 cannot lock m1 (held by t2)

t2 cannot lock m3 (held by t3)

t3 cannot lock m2 (held by t1)

Semaphores / Condition Variables (25 marks total)

Given the **struct** and functions for a semaphore implementation using condition variables in the provided code, you'll identify and explain any issues. Assume the existence of a `semaphore_init(int value)` function that initializes the semaphore's mutex, condition variable, and value. Focus particularly on thread safety and correct semaphore behavior.

```c
1  struct semaphore {
2      int value;
3      pthread_mutex_t mutex;
4      pthread_cond_t is_positive;
5  };
6
7  void semaphore_post(struct semaphore* sema) {
8      ++(sema->value);
9      pthread_cond_signal(&sema->is_positive);
10 }
11
12 void semaphore_wait(struct semaphore* sema) {
13     pthread_mutex_lock(&sema->mutex);
14     while (sema->value == 0) {
15         pthread_cond_wait(&sema->is_positive, &sema->mutex);
16     }
17     --(sema->value);
18     pthread_mutex_unlock(&sema->mutex);
19 }
```

Q19 (5 marks). Assuming an initial semaphore value of 0, describe a sequence of context switches between two threads concurrently calling `semaphore_post` that could lead to unexpected output. Specifically, consider the final value of the semaphore after these operations.

T1 starts to execute line 8 and reads `value = 0` from the **struct**

Context switch

T2 starts to execute line 8 and reads `value = 0` from the **struct**

T2 writes its incremented `value = 1` to the **struct**

Context switch

T1 writes its incremented `value = 1` to the **struct**

The final value is 1 when we would expect 2

Q20 (5 marks). With the semaphore initialized to a value of 0, describe a scenario where a "lost wakeup" occurs due to concurrent execution of `semaphore_post` and `semaphore_wait` by two different threads.

T1 starts to execute line 13, locks the mutex, and reads `value = 0` from the **`struct`** on line 14

Context switch

T2 starts to execute and updates `value = 1` in the **`struct`**

T2 calls `pthread_cond_signal`: "lost wakeup"

Context switch

T1 calls `pthread_cond_wait`: this will never return

Q21 (5 marks).
Revise the provided semaphore implementation to address the concurrency issues identified in the previous two problems ("lost wakeup" and unexpected value). You may annotate directly on the code listing or refer to line numbers in your explanation. Briefly justify why your solution resolves these issues effectively.

You need to place:

`pthread_mutex_lock(&sema->mutex);` before line 8.

and

`pthread_mutex_unlock(&sema->mutex);` after line 8 (or line 9).

In the first question, it prevents the other thread from executing after the context switch, due to the lock. It cannot read an old value.

In the second question, it prevents the thread running `semaphore_post` from continuing before the thread running `semaphore_wait` calls `pthread_cond_wait`. After `pthread_cond_wait`, it will `unlock` the mutex, and the thread running `semaphore_post` can increment the value and wake up the other thread.

Assume your semaphore works Assuming a functional semaphore implementation, analyze the given code where init initializes the semaphore, sema before we create two threads. One thread executes t1, and the other executes t2. Our goal is to consistently print "Tic", "Tac", "Toe", in that order.

```
1  struct semaphore sema;
2
3  void init(void) {
4      semaphore_init(&sema, 0);
5  }
6
7  void* t1(void*) {
8      printf("Tic\n");
9      semaphore_post(&sema);
10     semaphore_wait(&sema);
11     printf("Toe\n");
12     return NULL;
13 }
14
15 void* t2(void*) {
16     semaphore_wait(&sema);
17     printf("Tac\n");
18     semaphore_post(&sema);
19     return NULL;
20 }
```

Q22 (5 marks). Analyze if this implementation can lead to a deadlock. Explain your reasoning.

Yes, it is possible to deadlock.

t1 executes, prints "Tic", changes the semaphore value to 1, then changes the value back to 0, prints "Toe"

t2 can now not make any progress, therefore it's deadlocked

Q23 (5 marks). Modify the given semaphore-based code to reliably print "Tic", "Tac", "Toe" in sequence. You may use additional semaphores if necessary. You may annotate directly on the code listing or refer to line numbers in your explanation.

Create another semaphore, sema2 and initialize it to 0 in init

Change line 10 to wait on sema2 instead of sema

Change line 18 to post sema2 instead of sema

Virtual Memory (20 marks total)

Assume a virtual memory system, Sv39, with 39-bit virtual addresses, 8 byte page table entries (PTEs), 56-bit physical addresses, and a 4 KiB page size.

Q24 (1 marks). How many bits are needed to store the physical page number (PPN) in a PTE?

44 bits.

Q25 (1 marks). What are other essential piece of data is required in a PTE?

Valid bit

Q26 (4 marks).
What are the limitations of this Sv39 virtual memory system when handling 1 TiB (1024 GiB) of physical memory?

Our kernel can physically map all the pages, but a single process would be limited to addressing 512 GiB of virtual memory.

Q27 (3 marks). Determine the starting physical address of the L1 page given a PTE in the L2 page table with the PPN `0xE`, and explain why this is the case.

The L1 page table would start at address `0xE000` because our pages are aligned, and the page table consumes an entire page.

Q28 (1 marks). What is the starting address of the index 3 PTE in the L1 page table given above?

Since our PTEs are 8 bytes, index 3 would start 24bytes (8×3) bytes from the start of the page table. Therefore, its starting address would be `0xE018`.

Q29 (3 marks). When can two independent processes, sharing a physical page but not explicitly sharing memory, no longer share this page?

As soon as either process tries to modify that page, we would have to create a new copy of it for that process (copy-on-write).

Q30 (3 marks).
What outcome would occur if you attempted to use any virtual address in this system where the L2 page table points to itself in every PTE?

Every single virtual address would map to the page that the L2 page table is stored.

Q31 (4 marks). Given that iOS utilizes 16 KiB page sizes, are there benefits to this approach with respect to a TLB that can store a fixed number of entries?

Yes, each entry would cover 4 times more memory, so it's more likely entries would still be in the TLB, increasing our hit ratio and performance.

Page Replacement (18 marks total)

Assume the following accesses to physical page numbers:

1, 2, 3, 4, 2, 5, 2, 3, 2, 1, 5

You have 3 physical pages in memory. Assume that all pages are initially on disk.

Q32 (10 marks). Use the clock algorithm for page replacement. Recall on a page hit, you'll set the reference bit to 1. For each access write all the pages in memory *after the access* in the boxes below. State the number of page faults after all the accesses.

1	2	3	4	2	5	2	3	2	1	5
1	1	1	***4***	4	4	4	***3***	3	3	3
	2	2	2	2	2	2	2	2	2	***5***
		3	3	3	***5***	5	5	5	***1***	1

8 page faults.

Q33 (6 marks). Now, use the FIFO algorithm for page replacement. All the other constraints are the same as the previous clock algorithm question. For each access write all the pages in memory *after the access* in the boxes below. State the number of page faults after all the accesses.

1	2	3	4	2	5	2	3	2	1	5
1	1	1	***4***	4	4	4	***3***	3	3	3
	2	2	2	2	***5***	5	5	5	***1***	1
		3	3	3	3	***2***	2	2	2	***5***

9 page faults.

Q34 (2 marks). For page replacement algorithms other than FIFO, what is the relationship between the number of frames (or the size of physical memory) and the number of page faults.

The more frames you have, a fewer number of page faults will occur.

Filesystems (18 marks total)

Assume a filesystem with a block size of 4096 bytes, 4-byte block pointers, and 128-byte inodes. inodes have 12 direct pointers, 1 indirect pointer, 1 double indirect pointer, and 1 triple indirect pointer. Analyze the output of `ls -lia` executed in a directory. The output format includes inode numbers as the first column.

```
28 drwxr-xr-x   2 jon  staff 4096 Nov 23 18:35 .
19 drwxr-xr-x   5 jon  staff 4096 Nov 23 16:44 ..
20 -rw-r--r--   2 jon  staff  100 Nov 23 18:32 a
20 -rw-r--r--   2 jon  staff  100 Nov 23 18:32 b
22 lrwxr-xr-x   1 jon  staff    1 Nov 23 18:34 c -> b
26 -rw-r--r--   1 jon  staff 5000 Nov 23 18:35 e
```

Q35 (1 marks). When editing file `c`, which inode's contents are modified?

20.

Q36 (2 marks). Is it possible to store all the inodes associated with this directory in a single block? Provide an explanation supporting your conclusion.

Yes, the first inode block would store inodes 1-32. ($\frac{2^{12}}{2^7} = 2^5$)

Q37 (1 marks). Determine the number of data blocks that file `e` will occupy.

2.

Q38 (1 marks). Calculate the number of bytes lost to internal fragmentation for file `e`.

3192

Q39 (4 marks). Determine the number of directories contained within the parent directory, represented by inode 19. Describe the names of all directories linked to this inode.

There would be 3 directories.

Assume the name of the parent directory is just `parent`

1. `parent` in some directory
2. `.` in the `parent` directory
3. `..` in the first directory
4. `..` in the second directory
5. `..` in the third directory

Q40 (2 marks). What occurs when you execute `rm b`? Is it possible for the file to be deleted?

> The number of hardlinks decreases from 2 to 1. It is not deleted.

Q41 (1 marks). Assuming `b` is removed, what is the impact on `c` and the associated inodes?

> Nothing changes in `c`, it's just now referring to a name that does not exist in this directory.

Q42 (2 marks). If file `a` is expanded to use 13 data blocks, are only 13 blocks in total required for its contents? Explain the storage mechanism for these blocks.

> We would need 14 data blocks, 12 directly on the inode itself, 1 for an indirect block, which would store 1.

Q43 (4 marks). How many total data blocks are needed if file `a` grows to fill 1040 blocks? Explain the storage arrangement of these blocks and the rationale behind it

> We would need 1043 data blocks, 12 directly on the inode itself, 1 for an indirect block, which would store 1024. 1 block for the first level of the double indirect pointer. 1 block for the second level of the double indirect pointer. This final level would store the pointers for the remaining 4 blocks.