CS 111 Midterm

TOTAL POINTS

94 / 100

QUESTION 1

1 Page replacement algorithm choice 10 / 10

✓ - **0 pts** Correct

- **10 pts** Incorrect/no answer
- **5 pts** Incorrect/no explanation of why algorithm choice matters
- **5 pts** Incorrect/no explanation of likely difficulties upon poor algorithm choice
- **2.5 pts** Explanation of why algorithm choice matters unclear/needs more detail
- **2.5 pts** Explanation of poor algorithm choice's consequences unclear/needs more detail

QUESTION 2

2 Spin lock performance 10 / 10

✓ - **0 pts** Correct

- **10 pts** Incorrect/no answer
- **5 pts** Incorrect/no explanation of how spin locks cause performance problems
- **5 pts** Incorrect/no explanation of how a thread can harm its own performance
- **2.5 pts** How spin locks cause performance problems unclear/needs more detail
- **2.5 pts** How a thread can harm itself with spin locks unclear/needs more detail

QUESTION 3

3 Virtual address translation 10 / 10

✓ - **0 pts** Correct

- **3 pts** Missing one case
- **6 pts** Missing two cases
- **1 pts** The page table doesn't get full in the sense of being too full. At most, it contains an entry for every page.
- **2 pts** You never "search" a disk for a page. You always know exactly where it is.
- **2 pts** You don't search page tables for invalid addresses, since they won't be there.
- **3 pts** Third case same as example case.
- **1 pts** And what happens in the third case?
- **2 pts** If the page is supposed to be somewhere and can't be found anywhere, that's an OS crash, not a page fault. This must never happen.
- **3 pts** I/O does not occur in the middle of handling an address translation.
- **1 pts** First outcome results in page fault.
- **1 pts** MMU cache page table entries, not pages
- **10 pts** Diagram does not describe cases.
- **7 pts** Imprecise description of situation and actions for all three cases.
- **2 pts** What precisely do you mean by "system will continue"?
- **1 pts** Entire page table isn't cached in MMU. Individual entries are.
- **1 pts** In third case, if page isn't in RAM, you have to pay to get it from disk. Context switches may result, but that's not the main activity required.
- **1 pts** How does the system "add a page to the frame"?
- **10 pts** You did not answer the question
- **1 pts** In case 3, cache what in the PTE?
- **2 pts** You don't make an invalid page valid by simply allocating a page frame.
- **3 pts** MMU must not allow one process to access another process' pages, regardless of their address.
- **3 pts** TLB doesn't cache actual pages.
- **2 pts** What is the consequence of case 2?
- **1 pts** If a page is on disk, it will not have an entry in the TLB.
- **6 pts** Cases 2 and 3 are not requests to translate an address.

- **3 pts** Dirty bit is only relevant for page replacement, not address translation.
- **3 pts** We don't move an invalid page into a process' working set because it issued an address in the page.
- **1 pts** Page on disk is listed in page table, just with present bit not set.
- **2 pts** If page is not in a RAM page frame, it's on secondary storage and access will be very slow.
- **2 pts** Valid bit and present bit have different meanings.
- **2 pts** In first case, must get page off disk into a page frame
- **3 pts** First case won't happen.
- **1 pts** More details on first case.
- **3 pts** Third case won't happen.
- **4 pts** Click here to replace this description.

QUESTION 4

4 Results of fork 10 / 10

✓ **- 0 pts** Correct

- **2 pts** Does not mention pid difference/ return code
- **5 pts** Unclear about differences between parent and child
- **10 pts** Completely wrong
- **3 pts** Insufficient explanation
- **1 pts** Does not mention utility of return code/ pid in differentiating between parent and child
- **1 pts** fork() call in child returns 0 not 1 or something else
- **10 pts** No answer
- **4 pts** Does not provide any explanation for why stated difference is useful
- **2 pts** Copy-on-write, not always
- **2 pts** Child does not have a PID of zero, that is the return value from fork()
- **0 pts** correct

QUESTION 5

5 Scheduling for turnaround time 10 / 10

✓ **- 0 pts** Correct

- **10 pts** No answer
- **5 pts** RR does not finish short jobs quickly, thus does not optimize average turnaround time.
- **5 pts** Non-preemptive algorithms allow long job to keep new short jobs waiting.
- **5 pts** Did not specify which algorithm to use.
- **2 pts** SJF or STCF? Which?
- **3 pts** STCF over SJF, due to preemption issue.
- **5 pts** FIFO chooses early arrivers over short jobs, harming average turnaround time. One long job could kill your average.

\+ **4 pts** Preemption is indeed necessary

- **8 pts** This approach does not consider that running short jobs first reduces average turnaround time.
- **4 pts** Earliest deadline first only applies to RT scheduling.
- **3 pts** STCF will do better, if one has a good estimate of job run time.

\+ **2 pts** Good explanation.

- **8 pts** Not clear what algorithm you mean. Poor explanation of why to use it.
- **4 pts** Insufficient explanation.
- **4 pts** Without knowledge of job run times, MLFQ will probably do better than your choice.

\+ **2 pts** Mentioned SJF, but did not favor over other incorrect choices.

- **3 pts** Preemptive or not?

QUESTION 6

6 Changing page size 10 / 10

✓ **- 0 pts** Correct

- **3 pts** No external fragmentation with either page size.
- **1 pts** More details on internal fragmentation effect.
- **3 pts** Less internal fragmentation, not more, none, or the same.
- **2 pts** More details on non-fragmentation effect
- **3 pts** No discussion of external fragmentation
- **4 pts** No discussion of another effect
- **1 pts** As long as the pages are in RAM, the speed of access won't be much different.
- **4 pts** This effect will not occur.

- **4 pts** Page size does not really affect allocation requests.
- **3 pts** With paging, need not use method like best/worst fit.
- **4 pts** Thrashing is not directly related to page size. It is based on actual memory use.
- **3 pts** Non-contiguous allocations across page frames already happens with 4K pages.

QUESTION 7

7 Flow control and shared memory 5 / 10

- **0 pts** Correct

✓ **- 5 pts Flow control for sockets not explained/incorrect**

- **5 pts** Absence of flow control for shared memory not explained/incorrect
- **2.5 pts** Flow control for sockets unclear
- **2.5 pts** Absence of flow control for shared memory unclear
- **10 pts** Incorrect
- **1 pts** Sockets aren't unidirectional
- **1 pts** Sockets don't imply 2 machines

QUESTION 8

8 ABIs and software distribution 10 / 10

✓ **- 0 pts Correct**

- **3 pts** Does not mention that ABIs specify how an application binary must interact with a particular OS running on a particular ISA
- **3 pts** Does not mention the need for fewer versions of code / If OS is made compliant then code compiled to an ABI will run on any compliant system
- **5 pts** Unclear about what an ABI is
- **2 pts** Does not mention lack of requirement for user compilation
- **3 pts** Unclear answer
- **2 pts** Needs more detail
- **10 pts** Wrong

QUESTION 9

9 Relocating partitions 9 / 10

- **0 pts** Correct

✓ **- 1 pts More generally, virtualization (both segmentation and paging) allows relocation.**

- **8 pts** Virtualization is the key to relocation.
- **7 pts** Swapping alone won't do it. You need virtualization of addresses.
- **10 pts** Totally wrong. Virtualization is the technique.
- **4 pts** Insufficient explanation.
- **10 pts** No answer.
- **2 pts** Insufficient explanation
- **2 pts** TLB is just a cache. General answer is virtualization.
- **0 pts** Not really called "address space identifiers," but the concept is right
- **3 pts** this is virtualization, not swapping.
- **4 pts** Other way around. To relocate, you change the physical address, not the virtual address.
- **7 pts** Incorrect explanation of the aspect of virtualization that allows relocation.

QUESTION 10

10 Semaphore bug 10 / 10

- **0 pts** Correct
- **10 pts** Incorrect
- **0 pts** Balance checked against withdrawal before obtaining semaphore: balance could decrease between check and lock if unspecified code contains decrement to balance

✓ **- 0 pts Balance checked against withdrawal before obtaining semaphore: balance could decrease between check and lock if concurrent run of thread 2**

- **5 pts** Balance checked against withdrawal before obtaining semaphore: incomplete assumptions
- **10 pts** Assumed bug in unspecified code
- **1 pts** semaphore should be initialized with 3
- **3 pts** b = b+a not being atomic is irrelevant here and cannot cause a bug
- **2 pts** Another strange part [...] <- That comment is incorrect

Midterm Exam
CS 111, Principles of Operating Systems
Fall 2018

Name: ____

Student ID Number: ______________________________________

This is a closed book, closed note test. Answer all questions.

Each question should be answered in 2-5 sentences. DO NOT simply write everything you remember about the topic of the question. Answer the question that was asked. Extraneous information not related to the answer to the question will not improve your grade and may make it difficult to determine if the pertinent part of your answer is correct. Confine your answers to the space directly below each question. Only text in this space will be graded. No question requires a longer answer than the space provided.

1. Why is proper choice of a page replacement algorithm critical to the success of an operating system that uses virtual memory techniques? What is the likely difficulty if a poor choice of this algorithm is made by the OS designer?

A proper page replacement algorithm allows more hits in the Transition Lookaside Buffer. This cache memory is significantly faster to access than outside memory storage on disk by many orders of magnitude. If the OS designer makes a poor choice for the page replacement algorithm many TLB misses will result in severe performance cost on the system as the TLB will not have pages needed resulting in slow page table reads to bring in required pages.

2. Spin locks can cause performance problems if not used carefully. Why? In some cases, a thread using a spin lock can actually harm its own performance. Why?

Spin locks waste cycles by repeatedly checking if a condition that likely will not change is true. While another thread is busy attempting to finish its work in the critical section to let another thread acquire the lock, All other threads will be wasting their scheduled time checking this condition rather than yielding. This can harm performance of its own thread in the case of single core thread execution, all threads are running on the same core. This means all N-1 of N threads spinning on the lock not only waste their scheduled time spinning, but also prevent the thread that acquired the lock to run and finish its work.

3. Assume you are running on a virtual memory system that uses both segmentation and demand paging. When a process issues a request to access the memory word at address X, one possible outcome in terms of how the address is translated and the content of the address is made available is: the address is valid, the page is in a RAM page frame, and the MMU caches the page table entry for X, resulting in fast access to the word. Describe three other possible outcomes of the attempt to translate this address and the actions the system performs in those cases.

1) The address isn't valid, so the page isn't in a RAM frame. The page-table will be consulted to find the translation is valid and the page will be brought into a RAM page frame. The MMU now rechecks the address to find it is valid and caches it.

2) The address isn't valid so the page table is consulted to find the translation is invalid. This causes an exception and the trap handler becomes active to resolve the issue by likely terminating the offending process.

3) The address isn't valid, but the page frame is in Ram. Page table is consulted to find location of page frame. The MMU is updated so it correctly points to the page frame resulting in a TLB hit.

4. When a Linux process executes a fork() call, a second process is created that's nearly identical. In what way is the new process different? Why is that difference useful?

The new process has an identical copy of the stack for the parent, but it is its own stack. This allows for it to allocate its own function calls and variables independent of the parent. It has its own unique Process Identifier Number which allows it to run independently of the parent who created it. These allow this new process great flexibility such as using exec() to load in entirely new code and become a new program independent of the parent who created it.

5. If your OS scheduler's goal is to minimize average turnaround time, what kind of scheduling algorithm are you likely to run? Why?

STCF (Shortest time to Completion First) is a great scheduling algorithm for minimizing average turnaround time. Average turnaround time is the average time it takes for all incoming processes to finish completion. By scheduling shorter processes before longer running ones, we prevent one really long process from hogging the CPU and prevent shorter ones from getting their turn. This has advantages over algorithms such as first come first served which would let a longer process run over shorter ones if it came first and round robin which is fair in execution, but drags out all processes maximizing turnaround time.

6. Assume you start with an operating system performing paged memory allocation with a page size of 4K. What will the effects of switching to a page size of 1K be on external and internal fragmentation? Describe one other non-fragmentation effect of this change and why it occurs.

This introduces less internal fragmentation than before. In paged memory allocation, The maximum internal fragmentation is limited by the size of the page. By decreasing it, the maximum internal fragmentation goes from 3999 bytes to 999 bytes. External fragmentation is unchanged from before as any available page can always be accessed via a map of the available pages. The non-fragmentation effect of this is increased overhead from page accesses. Each page now contains less code so programs will have to switch pages more frequently increasing the odds of TLB misses and potentially severe performance costs.

7. An operating system can provide flow control on an IPC mechanism like sockets, but cannot provide flow control on an IPC mechanism like shared memory. Why?

Sockets allow two processes to communicate by allowing each process to fill a queue with data that can be obtained or added to without issue. Shared memory comes with strong security concerns as each process can access data from the other process. It also leads to the same issues that come from concurrency with little to no way to prevent them such as one process overwriting or deleting data another process was accessing.

8. Why are application binary interfaces of particular importance for successful software distribution?

ABIs allow for binary distributions compiled under a particular ISA on an operating system (e.g. X86 Linux) to run on machines with a similar architecture and OS. This allows software manufactures to create Dlls or Dynamically Loadable Libraries that can be used by programmers to link their code with that external library without access to the source. It also allows for software manufacturers to provide compiled applications on one machine to users on a similar machine without requiring the users to compile any code at all.

9. Which memory management technique allows us to solve the problem of relocating memory partitions? How does it achieve this solution?

Paging allows us to solve the issues associated with relocating memory partitions. By breaking the address space into fixed sized chunks, we can keep a mapping from virtual addresses to physical ones. In doing so, physical memory can be reorganized as long as the mappings are properly updated. This allows for programs to use any amount of the physical memory as desired without dealing with ~~external~~ fragmentation! By keeping a list of unused pages, new programs can quickly find the number of pages it will need and fill those physical locations with its data.

10. The following multithreaded C code contains a synchronization bug. Where is it? What is the effect of this bug on execution? This is not a full program, but only a part of a program concerning some synchronization functionality. The fact that it's not a full program ISN'T the bug. I am looking here for a synchronization bug. If you find and specify some other bug that does not have synchronization issues, you will not get any credit.

```
sem_t balance_lock_semaphore;
int balance = 100;

. . . /* Unspecified code here */

sem_init(&balance_lock_semaphore,0,0); /* Initialize the balance semaphore
*/

char add_balance(amount) {
    sem_wait (&balance_lock_semaphore ); /* wait to obtain lock on balance
variable */
    balance = balance + amount;
   sem_post(&balance_lock_semaphore); /* Release lock after updating
balance */
}

void subtract_balance( amount ) {
    balance = balance – amount;
}

. . . /* More unspecified code here */

/* This code is run by thread 1. */

add_balance (deposit);

. . . /* More unspecified code here */

/* This code is run by thread 2.*/

if (balance >= withdrawal) {
    sem_wait(&balance_lock_semaphore); /* wait to obtain lock on balance
variable */
    subtract_balance (withdrawal);
   sem_post(&balance_lock_semaphore);
}

/* More unspecified code */
```

(10)

There is a synchronization bug caused by the if check. The same code can be run two different times and end up with different execution results.

Case 1: balance < withdrawal.

thread 2 executes first and sees gets to if (balance >= withdrawal). This condition fails. thread 1 now executes and balance gets updated to be larger than withdrawal.

Case 2: balance > withdrawal.

thread 1 executes and causes balance to become larger than withdrawal. Then thread 2 executes and performs the withdrawal since it passes the condition.

This code could end up resulting in someones account with adequate funds to not have a withdrawal made based on the order of execution of the threads.