# CS 111 Midterm Exam

Austin Guo

TOTAL POINTS

## 74 / 100

QUESTION 1

### 1 Pages and page frames 3 / 10

- **- 0 pts** Correct
- **- 10 pts** No answer.
- **- 2 pts** Did not explain that pages get placed into page frame.
- **- 1 pts** Said page is mapped to page frame
- **- 1 pts** No mention of location of page & page frame in system
- ✓ **- 4 pts** **Incorrect Reasoning**
- ✓ **- 3 pts** **Not accurate enough**
- **- 1 pts** No mention of virtual address
- **- 5 pts** Page frame contains exactly one page.
- **- 2 pts** Did not mention that page and page frame are the same size.

QUESTION 2

### 2 ABIs 3 / 10

- **- 0 pts** Correct
- **- 10 pts** No answer
- **- 1 pts** Did not mention operating system
- ✓ **- 4 pts** **Said applications don't need to worry about hardware differences**
- **- 2 pts** Off-topic
- ✓ **- 3 pts** **Incorrect reasoning**
- **- 2 pts** Did not mention software distribution
- **- 3 pts** Did not mention hardware and OS
- **- 2 pts** Left out hardware

QUESTION 3

### 3 Information hiding 10 / 10

- ✓ **- 0 pts** **Correct**
- **- 10 pts** No answer.
- **- 6 pts** Primary benefit is to avoid bugs arising from improper dependencies among modules.
- **- 3 pts** Major element of benefit is that it allows changes in module implementation.
- **- 7 pts** Really about hiding details of OS modules' implementation, not about concealing processes' address spaces from each other.
- **- 2 pts** Nothing to do with open vs. closed source.
- **- 9 pts** Primarily an issue involving abstraction.

QUESTION 4

### 4 Context switches 8 / 10

- **- 0 pts** Correct
- **- 2 pts** Not mentioning general registers
- **- 2 pts** Not mentioning PC
- **- 2 pts** Not mentioning Stack ptr
- **- 1 pts** Not mentioning PSW
- **- 2 pts** No discussion of memory mapping data
- **- 2 pts** No need to explicitly save data, since it's already sitting in memory.
- **- 3 pts** Generally nothing goes to disk on a context switch.
- **- 1 pts** What about memory needs to be saved?
- ✓ **- 2 pts** **Much of this stuff need not be saved, since it's already in memory. OS just needs to be sure it can be found again when process is switched back in.**
- **- 2 pts** The PCB is an OS data structure that exists as long as the process is around, so it need not be saved on a context switch.
- **- 1 pts** File size has nothing to do with a context switch.
- **- 1 pts** I have no idea what the flag you're talking about is.
- **- 1 pts** "state of the process" is vague.
- **- 2 pts** Caches aren't saved.
- **- 2 pts** No need to update a file descriptor during a context switch.

QUESTION 5

## 5 Trap tables 10 / 10

✓ - 0 pts **Correct**

- 10 pts No answer
- 3 pts Answer incomplete, should mention trap table is used to specify what code to run when trap occurs.
- 8 pts Wrong answer.
- 3 pts Answer incomplete.
- 2 pts User process has no thing to do with trap?

QUESTION 6

## 6 Race conditions 10 / 10

✓ - 0 pts **Correct**

- 10 pts No answer
- 5 pts Answer incomplete.
- 8 pts Answer incorrect.
- 2 pts Missing some details.

QUESTION 7

## 7 Blocking and threads 10 / 10

✓ - 0 pts **Correct**

- 10 pts No answer or Wrong anwser
- 5 pts Missing: User-mode threads block other threads of the same process.
- 5 pts Missing: Kernel-mode threads do not block other threads of the same process, as other threads can be scheduled to run on the same or another core.

QUESTION 8

## 8 STCF 0 / 10

- 0 pts Correct

✓ - 10 pts **No answer or Wrong answer**

- 5 pts Missing: interrupt the running one OR switch to the newly-added shorter ones.
- 8 pts Missing mention of new processes that might have shorter time to completion.

QUESTION 9

## 9 Fork and exec 10 / 10

✓ - 0 pts **Correct**

- 10 pts No answer
- 4 pts Not mentioning code replacement.
- 3 pts Not mentioning stack replacement.
- 3 pts Not mentioning heap replacement.
- 9 pts The question was about what happens after the exec, not the fork.
- 8 pts What resources are replaced by the exec?
- 7 pts Stack and code are changed by exec.
- 5 pts Fork/exec work with processes, not threads.
- 2 pts Even any data written after fork gets replaced by exec.
- 2 pts The old stack is totally overwritten.
- 10 pts Totally wrong. Nothing to do with multithreading and multicore.
- 6 pts So, what resources are replaced?

QUESTION 10

## 10 Fragmentation for memory management schemes 10 / 10

✓ - 0 pts **Correct**

- 10 pts No answer
- 5 pts Not identifying internal fragmentation for pages.
- 5 pts Paged segments suffer 1/2 page fragmentation.
- 5 pts Fixed segments suffer 1/2 internal segment fragmentation.
- 3 pts On average 50%
- 2 pts The 1.5% was a particular example. It will be 1/2 page, on average.
- 2 pts Internal fragmentation has little to do with how long the system runs, unlike external.
- 2 pts Paging and fixed size partitions never experience external fragmentation.
- 2 pts The paging form of fragmentation you describe is internal fragmentation.
- 2 pts Paging doesn't use binary buddy. It allocates in fixed size pages.
- 4 pts Fixed segments are likely to waste more memory on internal fragmentation than paging, not less.
- 3 pts No external fragmentation with paging.

- **5 pts** No answer on segmented system.
- **2 pts** Calling internal fragmentation "external".
- **1 pts** This form of fragmentation is called "internal."
- **4 pts** Paged segment fragmentation only occurs in the last page.



gradescope

# Midterm Exam
# CS 111, Principles of Operating Systems
# Fall 2017

Name: Austin Guo

Student ID Number: 604770554

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. In a system using modern virtual memory techniques, what is the relationship between a page and a page frame?

When implementing a paging memory management system, page frames contain multiple pages of memory to make swapping more efficient. Paging memory management requires storing page tables in memory, and because page tables can get very large if every page of memory a process accesses at some point is put in the page table, and because each process needs a page table, the page table can become more space efficient if we limit its size. But, then pages not in memory cannot be lost, since processes must think they own the whole address space due to transparency and should not have bad memory access to memory they have already allocated. Thus, pages not in memory go in disk, but disk is expensive to read and write to. As a result, when we swap pages in and out of disk, we read and write entire page frames, holding a number of pages that can be accessed by their offset.

2. Why are operating system ABIs of importance for convenient application software distribution?

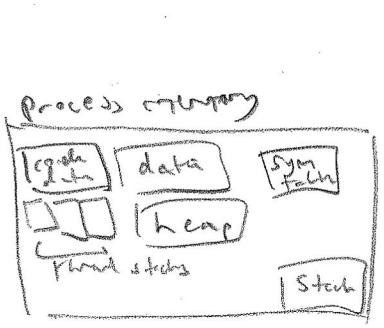
Operating system ABIs are extremely important for convenient application software distribution because applications should be distributed in executable binary, since customers are not technically skilled and want executable programs to run without the need for compilation. Having ABIs allows for OSs to easily support different standard application binary interfaces to which executables conform to means OS's will be able to support wider ranges of executables, since there are standards to which executable binaries should conform to in order to successfully support on certain machines. Historically this is important, so programs can successfully support on popular OSs that people are using that conform to popular ABIs, which improves the distribution of a certain software since people can actually use it with the machines they currently have. ABIs are also important since consumers expect their programs to just run without issues, and only ABIs can set standards for how executables should be formatted. ABIs allow the same binary executables to run on different ISAs. as long as the OS supports that ABI.

3. Why is information hiding a good property in an operating system interface?

Information hiding is a good property in an OS interface because it allows application programmers to make abstractions without knowing any nitty gritty details of how the OS operates. All the application developers need to do is follow the rules set forth by a particular API, and they will be able to easily make their program by using API modules while the OS hides the complexity behind each module's operation. For example, when creating threads using the pthread module, a programmer doesn't need to understand how the OS spawns threads or manages the removal of such threads, etc. they only need to follow the API and everything will work. This allows for building very complex systems without having to build up and understand each module from the ground up.

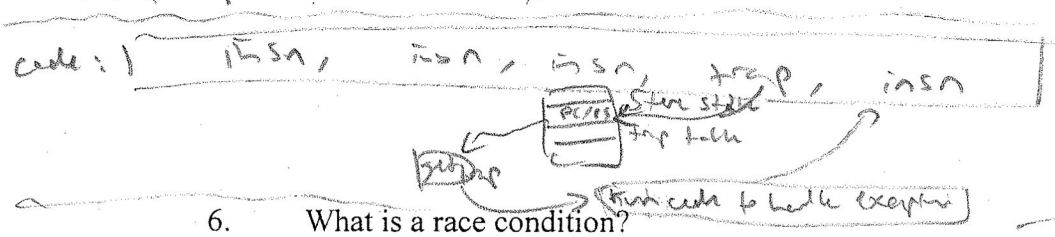
4. When an operating system performs a context switch between processes, what information must the OS save? The OS saves all the information of the current process. (the process' local memory)

The OS must save the current process' stack, register contents, heap and TCB (thread control table that keeps track of threads), along with whatever thread stacks were inside the process and additionally its heap memory and anything needed to restore the process exactly. This allows the OS to keep track of a process's state before switching it out and putting it in the ready queue, so that it may restore the process when the other processes are preempted or finish, and our process is next on the ready queue ready to be run again.



5. What is the purpose of a trap table?

The purpose of a trap table is to translate a hardware trap instruction into a particular course of action, i.e. switch this trap instruction to direct to run a particular segment of code inside an exception handler. This is important since there are many different types of exceptions that can occur that are all started by a trap instruction in the code triggered by the hardware, and the trap handler must be able to consult a trap table and decide how to handle the specific exception at hand using the state of the process's context.




6. What is a race condition?

A race condition is when the result of an execution of the same system given the same inputs is nondeterministic, but depends on the runtime execution timings of various instructions in the code data in a section of code called the critical section. race conditions often arise when there is a shared resource that is accessed and modified without controls, meaning they more often appear when threads are used rather than when processes are used. For example, if all threads share a counter and an accumulator global variable integers, and accumulate while counter is less than a value (say 100), accumulator will not necessarily be 100 at the end of the runs. This is because 2 instructions must be executed: an ① check to see if counter < 100, and an ② add to accumulator. If threads are preempted after checking that counter < 100 and see the condition is true, but another thread is switched in and increments counter to 100, all the first threads that were preempted will now switch in with their PC's pointing at the add to accumulator instruction, and accumulator will add to >100, while counter will also be incremented to >100. We can fix race conditions by atomically critical sections using locks / mutexes, or various other approaches.

7. Why is blocking a problem for user-mode threads? Why isn't it a problem for kernel-mode threads?

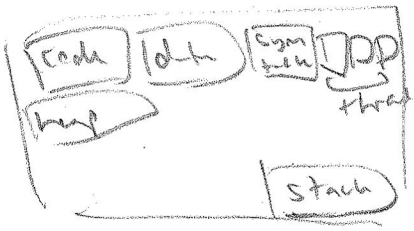
Blocking is a problem for user-mode threads because when a thread blocks on a process, unless there is preemption the thread will continue to block and none of the other threads in this process will be able to run while the thread is blocked. As a result, the other threads will be starved of resources and receive no runtime while the blocked thread wastes cycles, resulting in inefficiency (wasted cycles), unfairness (starved other threads), and no progress (blocked process). Preemption will allow us to preempt this thread and give other threads runtime/resources eventually. This is not a problem for kernel-mode threads because kernel-mode threads have access to multiple cores, and can thus run multiple threads at a time. If one thread wastes cycles blocking on one core, the other threads can still use the other cores, without preempting the blocking thread.

8. Why does Shortest Time-To-Completion First (STCF) scheduling require preemption?

Shortest Time to Completion First requires preemptive scheduling because preemption is a critical part of the implementation of the mechanism that determines length of a process. The OS doesn't know how long a process will run for, but it can arrive at this and approximate by dividing runtime on CPUs into time slices. If a process uses its entire time slice, it probably requires more time to run, and is thus a lower priority, but if a process doesn't use its whole time slice, it finishes quickly and is thus higher priority. The OS can keep track of which processes are fast or slow through some priority queue or more advanced data structure like an MLFQ that dynamically adjusts a process's priority and has separate queues for higher priority and lower priority processes. Preemption is relevant because in order to enforce the time slice for longer processes that run for longer than the time slice, the OS must have a scheduler that preempts the long process at the end of a time slice and switches to other processes in order to assess that processes length, so that after a few rounds of this the OS is slowly adjusting to a tailored implementation where shortest processes have higher priority

9. When a Unix-system follows a fork with an exec, what resources of the forked process are replaced? The forked process has all its resources replaced.

The forked process will have its stack, code data, thread data in TCB, heap and symbol table replaced, and anything else unique to this process. This is because exec will replace the current process with a fresh process running a different executable than the original process was running. Since a forked process will have the same code, stack, data, and thread data as the process that forked it, exec will replace all of this data with a clean slate for the new executable to use.



all replaced

10. What form of fragmentation do we still suffer if we use a paging memory management system? For a segmented paging system, how much fragmentation per segment do we see?

If we use a paging memory management system, we will only suffer from internal fragmentation, not external fragmentation.

For a segmented paging system with fixed size segments, we will see about 50% internal fragmentation on average. Though paging memory management systems usually exhibit about 1.5% internal fragmentation if they are variably sized segmented, meaning a variable number of pages can be allocated for each request, having fixed sizes of segments in the paging system will only solve the problem of external fragmentation by eliminating the need for contiguous blocks of memory. Within fixed size segments, however, performance acted as internal fragmentation remains the same, since only a fixed number of pages can be allocated at a time, so the memory allocation request is either satisfied exactly or not fulfilled, leading to an average of 50% internal fragmentation in the memory that is allocated.