Midterm Exam
CS 111, Principles of Operating Systems
Fall 2017

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

page frame is a fixed-size memory unit for memory management in physical address. Page is a fixed-sized memory unit in virtual address space. Page maps to page frame, and user could use page table to find the corresponding page frame number, and plus the offset in page will we get the desred the data from physical memory. The amount of page does not always equal to the amount of page frame since virtualization tends to give each process the illusion that it owns all physical memory, so usually # pages > # page frames.

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

ABI is application binary interface, which faces to user

Because an ABI, unlike API, will not need to recompile when moved to a new machine. The ABI is guaranteed to be able to run on any compliant machine that has compatiable ISAs. Then user does not need to debug it if it fails to compile.

And don't need to have the capability of compiling things

On the other hand, ABI supports API. With a ABI-adapting tool, an ABI-specific API could run on any machine without restriction.

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

Because it could hide unrelated details and provide more powerful service. The interface should hide the complexity under it because as an abstraction, it should not expose every details of its work so that lose the core goal of abstraction; making things easier to use and manage. If much information is exposed, user reading through might be confused about the service this interface provided and become too focused on unrelated details. Also, Too much information provided makes the interface service subject to danger since the attacker knows in details how the interface is implemented. Attacker could easily locates its vulnerable part. For safety concern, information hiding is also good.

4. When an operating system performs a context switch between processes, what information must the OS save?

Each process has a process descriptor that records the process's state. Doing context switch means we need to store its state for current process to make sure when it get switched back, everything looks like unchanged. Since processes has their own stack, heap and data segment, they are part of the process state and should be saved. Registers and PC/PS also need to be saved. Also, the working set of the page it's currently using need to be saved.

5. What is the purpose of a trap table?

Trap is a way of switching the control from process to OS that invoked by process itself. After a trap is invoked, it'll go to trap table to find corresponding entry and goes into the next level to find what to do.

The trap table exists to translate the information passed by the system call user made, and find the right thing for OS to handle the job because trap is called when user faces something that could not be handled in the user mode but requires privilege.

6. What is a race condition?

Race condition is the case that parallel threads try to modify the same data and results in indeterministic output. Suppose two threads are supposed to add 1 separately on a global variable, but B reads the variable before A writes back, and then B writes back. So in reality A's write makes no effect. Parallelism makes it unpredictable of the running order of instructions so that the race condition's output could vary in different execution

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

For user-mode threads, if one of them is blocked, the processor is not able to run the other runnable threads because in its point of view, each process has only one thread. and one thread blocks means the block of whole process.

For kernel mode, processor clearly knows each thread separately and even one of them is blocked, others could still run properly. since others are still runnable so kernel would not regard them as blocked as a whole.

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

Because the rule of STCF scheduling is to let the process with shortest run time goes first. we can not achieve it without interrupting the currently running process if a more valid candidate shows up.

Pre-emption refers to the case that instead of waiting for a process to finish and then schedule something else to run, it'll stop the currently running process to makes the other run immediately. Thus in this case, suppose we have process A comes first, as the only process the scheduler has to select it even it's of long run-time, and without preemption, a shorter process B has to wait until A completes, which disobey the rule of STCF.

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

A fork copies the parent process's stack, code, ... to child process and exec makes the child process like an newly created process.

The code segments, heap, stack, data that are process-specific registers, etc. resources are replaced.

Unix choose to do fork → exec to create new process because it'll be costly to generate process from begining to reference back to kernel for its structure. Instead, fork → exec saves such time and overhead, and child process could easily and cheaply remove the data, stack, etc.

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?
fixed size segments

If we use paging memory system, which is a improved memory management system based on fixed-size segment allocation, We will suffer internal fragmentation. Since it's fixed-size, it'll not have external fragmentation, which refers to the segmented chunks of memory caused by variable sized memory segments that are created on demand.
Internal fragment, on the other hand, refers to the situation that the process does not use up all the memory it has been allocated. Thus causing free memory not available if new process comes even if there is enough space.

For a segmented paging system, on average the internal fragmentation will be $\frac{1}{2}$ page, which is the last page's $\frac{1}{2}$ of all the pages assigned to the process. Compared to the naive fixed-size memory allocation, which has on average 50% memory fragmentation, the paging system is obviously