Operating Systems Design
Spring 2011 Exam 1 Review

Paul Krzyzanowski
pxk@cs.rutgers.edu

Question 1

Most operating systems are designed for general-purpose computation. A proposal has been put forth for an OS that is optimized for running math-intensive programs. In MathOS, the kernel includes system calls for many useful mathematical operations, such as matrix arithmetic, Bessel functions, Euclidean distance, etc. These system calls are written in highly optimized assembly language for maximum performance. Is this concept for MathOS a good idea? Explain why or why not.

No!

Key part: Math functions won't benefit from running in the kernel. They do not need privileged instructions or special facilities. You're putting crap in the kernel that has no reason to be there.

Moreover: The overhead of calling them: mode switch + data copy is more time-consuming than a function call

NOT: Assembly code is difficult to write/read/debug/etc.

Question 2

What is the difference between a *mode switch* and a *context switch*?

Mode switch: change CPU execution mode from one privilege level to another
e.g., user → kernel via a trap or syscall.

Context switch:
save one process' execution context & restore that of another process

Question 3

List two events that may take a process to a *ready* state.

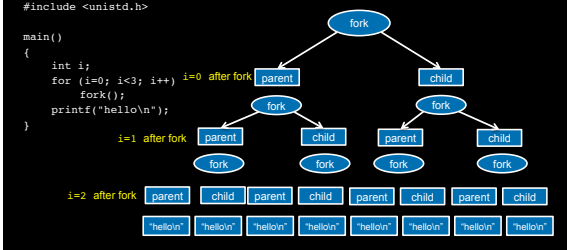
1. Startup: *created* → *ready*
2. Preemption: *running* → *ready*
3. I/O complete: *blocked* → *ready*

Question 4

How many times does the following program print hello?

```
#include <stdio.h>
#include <unistd.h>

main()
{
    int i;
    for (i=0; i<3; i++)
        fork();
    printf("hello\n");
}
```



Answer: 8

Question 5

Given that we can create user-level code to control access to critical sections (e.g., Peterson's algorithm), why is it important for an operating system to provide synchronization facilities such as semaphores in the kernel?

Question is about offering sync services via the kernel than via user-level code.

To avoid busy waiting: the waiting thread can go to sleep
→ creates better cpu utilization; avoids priority inversion (if an issue)

NOT:
- The kernel needs to manage critical sections
- Software solutions might be buggy
- Semaphores are useful because …

Question 6

A short quantum allows a scheduler to cycle through more processes more quickly than with a long quantum. What is the downside of this?

Increased overhead due to context switching.
Context switching takes time. We'll be doing more of it.

NOT:
- The thread may not finish in one quantum.
- The thread may never finish.
- Anything to do with thread priorities.

Part II: Question 7

What does a time-sharing system need that a multiprogramming system does not?

(a) Trap mechanism
(b) Kernel mode execution privileges
(c) Shorter time slices
(d) Interval Timer

Question 8

In an Intel PC architecture, the Master Boot Record (MBR): *contains code that…*

(a) Loads the operating system.
(b) Loads the system BIOS.
(c) Loads the Volume Boot Record (VBR).
(d) Allows the user to choose which operating system to load.

Question 9

When does preemption take place?

(a) When a quantum expires.
(b) When a process issues an I/O request.
(c) When a process exits.
(d) All of the above.

Question 10

With DMA (Direct Memory Access):

(a) The processor can read or write directly to a device.
(b) The kernel can read or write directly to a process' memory without intermediate buffers.
(c) A process can read or write to kernel memory without intermediate buffers.
(d) The device can read or write directly to the system's memory.

Question 11

When a process is first launched, the operating system does not know the size of this segment:

(a) text
(b) data
(c) bss
(d) heap

Question 12

In contrast to a cooperative scheduler, a preemptive scheduler supports the following state transition:

(a) Ready → running

(b) Running → ready

(c) Ready → blocked

(d) Blocked → running

Question 13

On POSIX systems, one process can send a signal to another process via:

(a) notify

(b) signal

(c) wait

(d) kill

Question 14

What information is stored in a thread control block (TCB)?

(a) List of open files.

(b) Stack pointer.

(c) Memory map.

(d) Thread owner ID.

Question 15

To implement a user-level threads package, it helps if the operating system provides:

(a) Non-blocking system calls.

(b) Kernel threads.

(c) An execve mechanism.

(d) Direct memory access

Question 16

Switching between user level threads of the same process is often more efficient than switching between kernel threads because:

(a) User level threads require tracking less state.

(b) User level threads share the same memory address space.

(c) Mode switching is not necessary.

(d) Execution stays within the same process with user level threads

a, b, and d apply to kernel threads as well.

Question 17

A compare-and-swap instruction (CAS, or CMPXCHG on Intel systems) allows you to:

(a) Modify a memory location only if its contents match a given value.

(b) Exchange the contents of two memory locations if their values are different.

(c) Exchange the contents of two memory locations if a lock is not set.

(d) Exchange the contents of two memory locations if a lock is set.

Question 18

Starvation is the case when a thread:

(a) Loops continuously until it runs out of memory.

(b) Is never scheduled to run.

(c) Can never acquire a lock on a critical section.

(d) Cannot create a child process or thread

Question 19

Two threads are considered to be asynchronous when:

(a) They have no reliance on one another.

(b) The outcome of a thread is dependent on the specific sequence of execution of both threads.

(c) Only one thread is allowed to access a shared resource at a time.

(d) The threads require occasional synchronization.

Question 20

A thread that is blocked on a semaphore is awakened when another thread:

(a) Tries to decrement a semaphore's value below 0.

(b) Tries to increment the semaphore.

(c) Causes the semaphore's value to reach a specific number.

(d) Tries to block on the same semaphore

Question 21

Condition variables support these operations:

(a) Wait / notify

(b) Read-and-increment / wait-for-value

(c) Increment / decrement-and-wait

(d) Set-value / wait-for-value

Question 22

A quantum is:

(a) The absolute minimum time that a process can run.

(b) The maximum time that a process can run before being preempted.

(c) The amount of time that a process runs before it blocks on I/O.

(d) The fraction of a time slice during which the process is running.

Question 23

Process aging is:

(a) Computing the next CPU burst time via a weighted exponential average of previous bursts.

(b) The measurement of elapsed CPU time during a process' execution.

(c) Boosting a process' priority temporarily to get it scheduled to run.

(d) Giving a process a longer quantum as it gets older.

Question 24

Differing from a soft deadline, a hard deadline:

(a) Is one where it is difficult to predict when the thread will exit.

(b) Applies to periodic (nonterminating) rather than terminating processes.

(c) Is one where there is no value to the computation if the deadline is missed.

(d) Is one where it is difficult to predict when the CPU burst period will end

Question 25

Which scheduler gives each process an equal share of the CPU?

(a) Round robin.

(b) Shortest remaining time first.

(c) Priority.

(d) Multilevel feedback queues

Question 26

Push migration is:

(a) When a processor has nothing in its run queue and grabs a process from another run queue.

(b) When a processor forks a new process to run on another processor.

(c) The migration of a process over a network from one computer to another one.

(d) The periodic rebalancing of the run queues among multiple processors

Question 27

A multilevel feedback queue scheduler generally assigns a long quantum to:

(a) High priority processes.

(b) Low priority processes.

(c) New processes.

(d) Old processes.

Question 28

Which scheduler relies on predicting the next CPU burst based on an average of previous bursts?

(a) First-come, first served.

(b) Round robin

(c) Shortest remaining time first.

(d) Multilevel feedback queues

Part III: Question 29

Software interrupts are synchronous with the current process.

True

Part III: Question 30

A context switch takes place at every system call

False

Part III: Question 31

Programmed I/O (PIO) uses fewer CPU resources than DMA.

False

Part III: Question 32

The POSIX *execve* system call creates a new process.

False

Part III: Question 33

Switching among threads in the same process is more efficient than switching among processes.

True

Part III: Question 34

Using mutual exclusion ensures that a system avoids deadlock.

False

Part III: Question 35

Rendezvous is a form of messaging that uses indirect addressing.

False

Part III: Question 36

Rate monotonic analysis assigns the highest priority to the process that has the most computation remaining.

False

Part III: Question 37

Multilevel queues allow multiple processes to share the same priority level.

True

Part III: Question 38

The value of a semaphore can never be negative.

True

The End