Operating Systems Design
Exam 1 Review: Spring 2012

Paul Krzyzanowski
pxk@cs.rutgers.edu

Question 1

UNIX-derived systems execute new programs via a two-step process of fork and execve. Other systems provide a single system call to run a new program. Explain an advantage of using this two-step approach.

A. It allows a child process to do prep work using settings from the parent process, such as setting up standard input, output, and error file descriptors (including pipes) and the current directory prior to running the program.

B. (*worse answer, since it does not address the question*) It allows you to clone a process to serve a specific request or distribute computation.

NOT:

- you can execute a new program while the old one runs
- there is less overhead and fewer mode switches
- if the new process fails, the parent can detect it.

Question 2a

What is the potential danger in running this code?

```
while (1) fork();
```

Unbounded growth in the number of processes will lead to a kernel memory shortage (process list) and user memory shortage (each process requires memory).

Not:

- Creates an infinite number of processes (it's not infinite)
- Will fork an unlimited number of times (that's obvious)
- Will result in a system crash. You should explain what happens. A crash is a useless term

Question 2b

What should be done in the kernel to limit this problem?

A. Set a per-user limit on the number of processes that can be created.
B. Set a limit on the number of child processes (including arbitrarily deep grandchildren) that a process can create.

Not:

- Fix the program. The question asks for kernel-level solutions, not remedies for user code. An operating system should be robust to deal with bad user code.
- The kernel should detect the loop. The kernel does not analyze user code. If it did, it would take a lot of time and it would not be possible to detect this behavior in many circumstances.

Question 3

We need to sum up a huge array of numbers and will use multiple threads to do so.

A worker thread is started with a thread ID as a parameter, where threads are numbered 0, 1, … (*numvalues*-1).

Each of *N* worker threads sums up every *N*th element of the array in parallel into a local variable, *sum*.

At the end, they each add their answer to a running total called *total*.

A separate thread, *main*, dispatches the workers and computes and prints the final result.

Question 3

Using one semaphore, *s*, and one event variable *e*, and without waiting on (joining) threads, define the initial values of s and e and the placement of *up(s)/down(s)* and *advance(e)/await(e, value)* operations to ensure that all the summations takes place without race conditions and only when the data is ready.

Question 3

```
/* global values */
int total, values[], numvalues, N;

sem s= 1;
event e= 0;

main() {
    pthread t[N];
    int i;

    for (i=0; i < N; ++i) {
        pthread_create(&t[i], NULL, worker, i);
    }

    await(e, N); /* wait for all to finish */

    printf("total = %d\n", total);
}
```

```
worker(int i) {
    int j, sum=0;

    for (j=i; j < numvalues; j+=N) {
        sum += values[j];
    }

    down(s);
    total += sum;
    up(s);

    advance(e); /* done! */
}
```

Question 4

What is the difference between deadlock and starvation?

Deadlock: a process can make no progress even if it is scheduled because there is a circular dependency on resources coupled with exclusive (locked) access.

Starvation: the scheduler never schedules the process
(N.B.: it does not need to be a low-priority process; that's up to the scheduler).

Question 5

If short time slices improve interactive performance, what is the downside?

Context switch overhead: The number of context switches increases.

NOT: processes with large CPU bursts will not get to execute.
This question does not specify the scheduling algorithm. Do not make any assumptions on the scheduler.

Question 6

How does Multilevel Feedback Queue scheduling try to approximate Shortest Remaining Time First (SRTF) scheduling? How does it not have the disadvantage of SRTF?

Both try to give high priority to processes that exhibited short CPU bursts.

The advantage of MFQ is that there is no need for an estimation function; the decision is automatic based on whether the process used up its allotted time slice last time. Moreover, there is no need to keep a sorted queue. Insertions can be O(1).

NOT:

- MFQ provides longer time slices for low-priority or CPU-intensive processes.
- MFQ avoids starvation. Process aging can be added to either algorithm.

Part II: 7-9

7. In contrast to the BIOS, EFI (Extensible Firmware Interface):
(a) Is stored in system firmware instead of on the disk.
(b) Loads a boot loader from the master boot record (MBR).
(c) Bypasses the Master Boot Record (MBR) and load a boot loader from the volume boot record (VBR) directly.
(d) Can load a boot loader from a file system on the disk.

8. Which of the following is most likely to be a system call?
(a) The implementation of a while loop in C.
(b) Parse a token from a string.
(c) Get the cosine of a number.
(d) Get the time of day.

9. The primary use of a Programmable Interval Timer (PIT) is to:
(a) Measure the elapsed time between two events.
(b) Put the system to sleep but have it wake up later at a predetermined time.
(c) Put a process to sleep for a specified interval.
(d) Make sure that the kernel gets control of the processor periodically.

Part II: 10-12

10. Which of these statements is true?
(a) A mode switch precedes a context switch.
(b) A context switch precedes a mode switch.
(c) A context switch can occur without a mode switch.(d) A mode switch is just a different name for a context switch.

11. Disk controllers tend to use Direct Memory Access (DMA) over Programmed I/O (PIO) because:
(a) The CPU does not have to copy the disk data byte by byte.
(b) Most disks are not programmable.
(c) Transferring data to or from a disk is performed by the kernel.
(d) The entire disk appears as memory to an operating system.

12. The first demonstration of a computer system with a graphical display, keyboard, mouse, networking, shared screen collaboration, and hypertext took place in:
(a) 1956
(b) 1968
(c) 1980
(d) 1992

Part II: 13-15

13. Which process state transition is not valid?
(a) Ready → Running
(b) Running → Ready
(c) Running → Blocked
(d) Blocked → Running

14. Which process state transition is valid only on a preemptive multitasking system?
(a) Ready → Running
(b) Running → Ready
(c) Running → Blocked
(d) Blocked → Running

15. A Thread Control Block (TCB) stores:
(a) User (owner) ID
(b) Memory map
(c) The machine state (registers, program counter)
(d) Open file descriptors

Part II: 16-18

16. A race condition is:
(a) When one process is trying to beat another to execute a region of code.
(b) When a process cannot make progress because another one is blocking it.
(c) When the outcome of processes is dependent on the exact order of execution among them.
(d) A form of locking where processes coordinate for exclusive access to a critical section.

17. A process scheduler is responsible for moving processes between these states:
(a) Ready and Blocked
(b) Running and Blocked
(c) Ready and Running
(d) Ready, Running, and Blocked

18. The wait system call on UNIX systems puts a process to sleep until:
(a) A semaphore wakes it up.
(b) The specified elapsed time expires.
(c) A child process terminates.
(d) The process is preempted by another process.

Part II: 19-21

19. Priority inversion occurs when:
(a) A scheduler repeatedly schedules a low-priority thread over a high-priority one.
(b) A high-priority thread is blocked, causing the scheduler to run a low-priority thread.
(c) A scheduler schedules a high-priority thread when it would be better to schedule a low-priority one.
(d) A high-priority thread wakes up a low-priority thread, causing it to be scheduled.

20. Every process gets the same share of the CPU with a:
(a) Round-robin scheduler.
(b) Shortest remaining time first scheduler.
(c) Priority scheduler.
(d) Multilevel feedback queues.

21. A weighted exponential average is useful in process scheduling for:
(a) Determining the length of a quantum for the thread.
(b) Allowing a priority scheduler to assign a priority to the process.
(c) Estimating the length of the next CPU burst for the thread.
(d) Ordering processes in the ready queue for a round robin scheduler.

Part II: 22-23

22. Process aging:
(a) Increases the priority of a process if it sits in the ready state for a long time.
(b) Decrease the priority of a process each time the process gets to run.
(c) Increases the priority of a process as it gets older.
(d) Decreases the priority of a process as it gets older

23. Which scheduler does not risk starvation of processes?
(a) Round-robin scheduler.
(b) Shortest remaining time first scheduler.
(c) Priority scheduler.
(d) Multilevel feedback queues.

Part II: Question 24

Processes A and B are ready to run at the same time. A needs 600 msec of CPU time. B needs 800 msec of CPU time. Both need to finish within one second. Assume that a Least Slack scheduler is used with an infinitesimally small time slice. When do A and B terminate?

(a) A at 600 msec, B at 1400 msec.

(b) A at 1400 msec, B at 800 msec.

(c) A and B at 1400 msec.

(d) A and B at 1000 msec (1 sec).

Initial slack:
A: 1000 - 600 = 400 msec
B: 1000 - 800 = 200 msec
B is scheduled for the first 200 msec. After that, slack(B) = slack(A) = 200 msec
Both have 600 msec of execution left.
If A runs, then B's slack decreases.
If B runs, then A's slack decreases.
A & B run in alternate time slices.

Part II: Question 25

Now assume an Earliest Deadline First scheduler is used. When to A & B terminate?

(a) A at 600 msec, B at 1400 msec.

(b) A at 1400 msec, B at 800 msec.

(c) A and B at 1400 msec.

(d) A and B at 1000 msec (1 sec).

Both processes have the same deadline. The scheduler may:
(a) run A first, then B
(b) run B first, then A
(c) alternate among A and B

Part II: Question 26

Process A runs 30 times a second. Each run lasts for 4 msec (a total of 120 msec CPU per second).

Process B runs 60 times a second. Each run lasts for 2.5 msec (a total of 150 msec CPU per second).

Process C runs 4 times a second. Each run lasts for 45 msec (a total of 180 msec CPU per second).

Which is a valid priority assignment using Rate Monotonic assignment where higher numbers correspond with higher priorities?

(a) A=2, B=1, C=3

(b) A=2, B=3, C=1

(c) A=1, B=2, C=3

(d) A=3, B=2, C=1

Highest frequency → highest priority

B > A > C

Part II: 27-30

27. A system call always results in a context switch.

True False

28. On a pre-EFI PC architecture, the BIOS is loaded from the Master Boot Record (MBR), which then loads the Volume Boot Record (VBR)

True False

29. A process executes faster when running in kernel mode than when running in user mode.

True False

30. Hardware support for mutual exclusion, such as test-and-set locks has the advantage of avoiding the need for spin locks.

True False

Part II: 31-33

31. Mailboxes are a form of messaging using indirect addressing.

[✓ True] False

32. Rendezvous messaging between processes on the same system reduces the amount of message copying needed.

[✓ True] False

33. Push migration may be used in multiprocessor scheduling with hard CPU affinity.

True [✓ False]

The End