System Calls and Kernel Modules

1. What is a system call? How do system calls differ from ordinary function calls?

2. What steps take place when a system call is invoked by a process?

3. What is a system call table? Why is it needed? OR What role does it play in OS security?

4. Explain the CPU-privilege transitions during a system call.

5. (a) Why do some operating systems, such as Linux, map themselves (i.e. the kernel code and data) into the address space of each process? (b) What is the alternative

6. Why are memory access errors (such as segmentation fault and null pointer dereferencing) more dangerous in kernel space than in user space?

7. What are kernel modules?

8. What are some advantages of kernel modules?

Concurrency

1. Define Concurrency. How does it differ from parallelism?
2. Explain the differences between apparent concurrency and true concurrency.
3. Briefly explain with examples
   A. Critical Section
   B. Race condition
   C. Deadlock
4. What's wrong with associating locks with code rather than shared resources?
5. Describe the behavior of (a) UP and DOWN operations on a semaphore, (b) WAIT and SIGNAL operations on a condition variable.
6. Under what situation would you use (a) Blocking locks, (b) Non-blocking locks, and (c) Spin locks. Which of these locks can be used in interrupt handlers and how?
7. When should you NOT use (a) blocking locks, (b) non-blocking locks, and (c) spin-locks?
8. What is the main difference between a binary semaphore and a counting semaphore?
9. What is priority inversion? How can prevent it?
10. Explain how a deadlock can occur in the operating system between code executing in the user-context and code executing in interrupt handlers. Also explain how you would prevent such a deadlock.
11. How does the ***Test-and-Set Lock (TSL)*** instruction work?
12. Explain how you can implement the UP and DOWN operations on a mutex (binary semaphore) using the TSL instruction.
13. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. In the following skeleton of pseudo-code, ***demonstrate the use of SEMAPHORES and MUTEXES*** to complete the pseudo-code for producer and consumer functions. Your code should have ***no race conditions*** and ***no busy loops***.

You can assume that the following functions are available to you. You shouldn't need anything more than these functions in your pseudo-code.

```
produce_item() produces and returns an item
insert_item(item) inserts the item in the common buffer
remove_item() removes and returns an item at the head of the buffer
consume_item(item) consumes the item supplied
up(&semaphore) and down(&semaphore) have their usual meanings
```

```
==========================Pseudo-code Skeleton ================================
#define N 100                  /* Number of slots in the buffer */
typedef int semaphore;         /* semaphores are a special kind of counter */
semaphore mutex = (initialize this);      /* figure out the role of mutex */
semaphore empty = (initialize this);      /* figure out the role of empty sem
*/
semaphore full = (initialize this);       /* figure out the role of full sem
*/

void producer(void)
{
      /* complete this function */
}

void consumer(void)
{
      /* complete this function too */
}
=====================================================================
```

14. Consider the classical producer-consumer problem. Producers produce items and insert them in a common buffer. Consumers remove items from the common buffer and consume them. Complete the following skeleton pseudo-code to explain how you can solve the producer-consumer problem using **a monitor and condition variables.**

```
procedure Producer
begin
        /* complete this procedure */
end

procedure Consumer
begin
        /* complete this procedure */
end

monitor ProducerConsumer
        condition /* declare the condition variables you need */
        integer /* declare any other variables you need */

        procedure  insert(item)
        begin
                /* complete this procedure */
        end

        procedure item *remove()
        begin
                /* complete this procedure */
        end
end monitor
```

15. What are the tradeoffs in using semaphores versus monitors with condition variables?

Memory management

(1) Show the typical memory hierarchy of a computer system. Explain the tradeoffs between latency (access times), capacity, and persistence, across different levels of memory hierarchy.

(2) If there were no TLB, how would memory accesses be affected?

(3) How many page tables are maintained by the operating system?

(4) What are the following? What do they do? Where are they located?

   A. Memory Management Unit (MMU)
   B. Translation Lookaside Buffer (TLB)
   C. Page tables

(5) How is a virtual address converted to a physical address in a virtual memory system? Explain the roles of MMU, TLB, and Page Tables.

(6) If you increase or decrease the page size in a system, how (and why) will it affect ***(a)*** the size of the page tables, and ***(b)*** the TLB miss ratio?

(7) What's TLB Coverage? Why is TLB coverage important?

(8) How can one increase the TLB coverage?

(9) In memory management, what is meant by relocation and protection?

(10)How is segmentation different from paging? Why was each technique invented?

(11)Using either Multics or Pentium architecture as an example, explain how segmentation is used in enforcing protection?

(12)A machine has a 32-bit address space and an 8-KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at the rate of one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables? (Assume that each process uses its entire virtual address space during execution.)

(13)A machine has a 32-bit address space and an 4KB page. The page table is entirely in hardware. Each page-table entry is 4 bytes in size. When a process starts, the page table is copied to the hardware from memory, at the rate of one byte every 25 nano-second. If each process has a CPU burst of 200 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables? (Assume that each process uses its entire virtual address space during execution.)

(14)Consider a machine having a 32-bit virtual address and 8KB page size.

   A. What is the size (in bytes) of the virtual address space of a process?
   B. How many bits in the 32-bit virtual address represent the byte offset into a page?
   C. How many bits in the 32-bit address are needed to determine the page number ?
   D. How many page-table entries does a process' page-table contain?
   E. Answer A—D when virtual address is 64-bits and page size is 16KB.

F. Answer A—D when virtual address is 64-bits and page size is 4KB.

G. Answer A—D when virtual address is 64-bits and page size is 32KB.

(15)Consider a machine having a 64-bit virtual address and 32KB page size.

A. What is the size (in bytes) of the virtual address space of a process?

B. How many bits in the virtual address represent the byte offset into a page?

C. How many bits in the virtual address are needed to determine the page number ?

D. How many page-table entries does a process' page-table contain?

(16)For each of the following decimal virtual addresses, compute the virtual page number and offset for a 4-KB page, an 8 KB page, and a 16KB page: 20000, 32768, 60000.

(17) A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into a 9-bit top-level page table field, an 10-bit second-level page table field, and an offset. How large are the pages and how many pages are there in the address space?

(18)Which system component handles a page-fault and a TLB miss in a machine with (a) architected page-table, and (b) architected TLB?

(19)What is the purpose of "Referenced" and "Modified" ("Dirty") bits in the page table entry? How are they manipulated by the (a) hardware, and (b) operating system?

(20)What is the difference between internal and external fragmentation?

(21)What are superpages? Why are they useful? What are the constraints on their sizes and placement?

(22) If a superpage has a size equal to 4 base pages, how many TLB entries are occupied by the superpage? How many page table entries are occupied by the same superpage? Explain why.

(23)Why is it that using a mix of multiple superpage sizes tends to yield better application speedups than using superpages of only one size?

(24)In the superpage paper, explain how ***preemptible reservations***, ***incremental promotions*** and ***speculative demotions*** work.