CSCE 313 QUIZ 1 Fall 2021

Student Name:……Raul Escobar……
UIN:……328003859……..

Student Score / 100

True/False Questions **[25 pts]**

1. The executable image of a program must be loaded into the main memory first before executing
   a. **True**
2. An Operating System (OS) does not trust application programs because they can be either buggy or malicious
   a. **True**
3. There was no concept of OS in first generation computers
   a. **True**
4. Second generation computers still executed programs in a sequential/batch manner
   a. **True**
5. Time sharing computers gave a fixed time quantum to each program
   a. **True**
6. An OS resides in-between the hardware and application programs
   a. **True**
7. The primary goal of OS is to make application programming convenient
   a. **True**
8. Multiprogramming cannot work without Direct Memory Access (DMA) mechanism
   a. **True**
9. A program can be kicked out of a CPU when it requests I/O operation, or when another Interrupt occurs
   a. **True**
10. A program error can kick a program out of CPU
    a. **True**
11. Interrupts are necessary to bring a program back to CPU if it was previously kicked out
    a. **True**
12. Modern operating systems come with many utility services that are analogous to the “Glue” role of the OS
    a. **True**
13. Resource allocation and Isolation are not part of the core OS, rather common services included with OS
    a. **True**
14. Efficiency is the secondary goal of an OS
    a. **True**
15. After handling a fault successfully, the CPU goes (when it does go back) to the instruction immediately after the faulting one
    a. **True**
16. Interrupts are asynchronous events
    a. **True**

17. Memory limit protection (within a private address space using base and bound) is implemented in the hardware instead of software
    a. **True**
18. Memory limit protection checks are only performed in the User mode
    a. **True**
19. Divide by 0 is an example of a fault
    a. **False**
20. Multiprogramming can be effective even with one single-core CPU
    a. **True**

Short Questions

21. **[5 pts]** Define multiprogramming. How is this better than sequential program execution?

    Multiprogramming is a way of running a program which interrupts if that program performs an I/O operation, and runs another program while doing I/O operations, and it does the same for that program.
    Good way of utilizing the CPU's time and resources to its fullest extent rather than having it run sequentially and wait for I/O operations to complete.

22. **[5 pts]** Define time-sharing. Can you combine time-sharing with multiprogramming?

    Method of sharing resources of a computer among various users or programs. It gives each program a quantum of time so that it may perform some operations, then it moves on to another program. This can be combined with multiprogramming.

23. **[5 pts]** Say you are running a program along with many other programs on a modern computer. For some reason, your program runs into a deadlock and never comes out of that. How does the OS deal with such deadlock? How about infinite loops? How does the OS detect, if at all, such cases?

    One method is to give each job/program a time quantum so that after a certain amount of time it hands off the cpu resources to another program, and the program gets pushed to the back of the queue. This makes it so that if there is a certain program that is too large, that it will not hog all the computer resources.

24. **[5 pts]** Which of the following are privileged operations allowed only in Kernel mode?

    Highlighting the answer
a) Modifying the page table entries
b) Disabling and Enabling Interrupts
c) Using the "trap" instruction
d) Handling an Interrupt

e) Clearing the Interrupt Flag

25. **[25 pts]** In a single CPU single core system, schedule the following jobs to take the full advantage of multiprogramming. The following table shows how the jobs would look like if they ran in isolation. [Use the attached pages from W. Stallings book to solve this problem]

	JOB1	JOB2	JOB3
Type of job	Full CPU	Only I/O	Only I/O
Duration	5 min	15 min	10 min
Memory required	50MB	100MB	75MB
Needs disk?	No	No	Yes
Needs terminal?	No	Yes	No

a. What is the total time of completion for all jobs in a sequential and multi-programmed model?

a. In sequential mode it takes 30 minutes
b. In multi-programmed mode it takes 15 minutes

b. Fill out the multiprogramming column in the following table (i.e., when the jobs are scheduled in multiprogramming). Assume that the system's physical memory is 256MB.

Average Resource Use	Sequential	Multiprogramming
Processor	5/30 = 16.67%	5/15 = 33.3%
Memory	32.55%	65%
Disk	33.33%	66%
Terminal	50%	100%

Memory usage is computed as follows: (5minx50MB + 15minx100MB + 10minx75MB) / (30minx256MB) = 32.55%

Other resources are fully utilized during the time they are utilized. So, you compute utilization only based on the duration they are used.

26. **[15 pts]** The following are steps in a "sequential" Interrupt handling. These steps are in the user-to-kernel direction, while the steps in the opposite direction are simply reversed.

Hardware does the following:

---

1. Mask further interrupts
2. Change mode to Kernel
3. Copy PC, SP, EFLAGS to the Kernel Interrupt Stack (KIS)
4. Change SP: to the KIS (above the stored PC, SP, EFLAGS)
5. Change PC: Invoke the interrupt handler

Software (i.e., the handler code) does the following:

---

1. Stores the rest of the general-purpose registers being used by the interrupted process
2. Performs the rest of the interrupt handling operation

Now, answer the following questions:

(a) [7 pts] What changes would you make in the steps below so that "nested" Interrupts can be handled?

(b) [3 pts] For sequential interrupt handling, can you interchange steps 2 and 3? Explain.

(i) No because User mode cannot handle these operations due to privileges

(c) [2 pts] For sequential interrupt handling, can we interchange step 1 and 2? Explain.

(i) Yes

(d) [3 pts] For sequential interrupt handling, can we interchange step 1 and 3? Explain.

(i) No

32. [20 pts] Assuming each page is 4KB, how many page faults or faults of any kind will the following program generate during the 2 for-loops? Explain your answer.

```
int size = 4 * 1024 * 1024; // size = 4 megabytes
char * memory = new char [size]; // allocate 4 mega bytes
for (int i=0; i< size ; i++)
      mem [i] = 0;

__int64_t * a = (__int64_t *) memory;//__int64_t is 64-bit or 8-byte integer
for (int i=0; i< size ; i++)
      a [i] = 0;
```

Assuming 'mem' is a typo that was meant to be memory defined on the second line. If not this won't even compile

1 page fault

Occurs when first (memory[i] = 0) is run because memory has not been mapped to physical memory yet. But does not happen on the 2nd for-loop because the memory has already been mapped.