CSCE 313 **QUIZ 1** Spring 2020

Student Name:................Siyuan Yang................. UIN:...........826006958..................

Student Score ☐ **/ 100**

True/False Questions [20 pts]

1. **[True]** The executable image of a program must be loaded into the main memory first before executing
2. **[True]** An Operating System (OS) does not trust application programs because they can be either buggy or malicious
3. **[True]** There was no concept of OS in first generation computers
4. **[True]** The PC register of a CPU points to the next instruction to execute in the main memory
5. **[True]** Second generation computers still executed programs in a sequential/batch manner
6. **[True]** Time sharing computers gave a fixed time quantum to each program
7. **[True]** An OS resides in-between the hardware and application programs
8. **[True]** The primary goal of OS is to make application programming convenient
9. **[False]** Context switching does not contribute much to the OS overhead
10. **[True]** Main Memory access is slower than register/cache access because it is physically outside the CPU
11. **[True]** Multiprogramming cannot work without Direct Memory Access (DMA) mechanism
12. **[True]** Interrupts are necessary for asynchronous event handling in a CPU
13. **[True]** A program can be kicked out of a CPU when it requests I/O operation, or when another Interrupt occurs
14. **[False]** A program error can kick a program out of CPU
15. **[True]** Interrupts are necessary to bring a program back to CPU if it was previously kicked out
16. **[True]** The "Illusionist" role of the CPU allows a programmer write programs that are agnostic of other programs running in the system
17. **[True]** Modern operating systems come with many utility services that are analogous to the "Glue" role of the OS
18. **[True]** Networking service is not a core OS part, rather a common service included with most OS
19. **[False]** Resource allocation and Isolation are not part of the core OS, rather common services included with OS
20. **[True]** Efficiency is the secondary goal of an OS

Short Questions

21. [10 pts] Define multiprogramming. How is this better than sequential program execution?

Multiprogramming is the ability of an OS to execute multiple programs at the same time on single processor machine, while sequential programming involves a consecutive and ordered execution of processes one after another. In other words, multiprogramming allows using the CPU effectively by allowing various users to use the CPU and I/O devices effectively and makes sure that the CPU always has something to execute, thus increasing the CPU utilization. Besides, it deceases total read time needed to execute a job and maximizes the total job throughput of a computer.

22. [10 pts] Define time-sharing. Can you combine time-sharing with multiprogramming?

Time-sharing means sharing of computing resources among many users by means of multiprogramming and multitasking. In times sharing systems, several terminals are attached to a single dedicated server having its own CPU. By allowing a large number of users to interact concurrently, time-sharing dramatically lowered the cost of providing computing capability. Time sharing systems use the concept of multiprogramming to share the CPU time between multiple users at the same time.

23. [10 pts] Say you are running a program along with many other programs in 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 loop? How does the OS detect, if at all, such cases?

To deal with such deadlock, resources can be preempted from some processes and given to others till the deadlock is resolved. When dealing with an infinite-loop deadlock, all the processes that are involved in the deadlock should be terminated; however, it might cause all the progress made by the processes is destroyed. A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes.

24. [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 sequential and multi-programmed model?

Total time in sequential model = 5 + 15 + 10 = 30 min
Total time in multi-programmed model = 15 min

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 = 33.33%
Memory	32.55%	(5*50+15*100+10*75)/(15*256) = 65.10%
Disk	33.33%	10/15 = 66.67%
Terminal	50%	15/15 = 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.

25. [25 pts] Consider the following program and provide explanations where asked in the code comment after running the program in your system. Note that there are 8 places where explanation is needed. For explanation 1 (i.e., the first commented line), does the Header size equal the sum of individual data types (i.e., a char, an int and a pointer)? Try to explain this with something called "packing".

```
#include <iostream>
#include <string.h>
using namespace std;

class Header{
private:
    char used;
    int payloadsize;
    char* data;
public:
    Header (){
        used = 0, payloadsize = -1, data = NULL;
    }
    Header (int ps, char initvalue = 0){
        used = 0;
        payloadsize = ps;
        data = new char [payloadsize];
        memset (data, initvalue, payloadsize);
    }
    int getsummation (){
        int sum = 0;
        for (int i=0; i<payloadsize; i++){
            sum += data [i];
        }
        return sum;
    }
};

int main (){
    Header h1;
    Header h2 (10);
    Header* h3 = new Header (20);
    cout << "Header type size " << sizeof (Header) << endl;
    // 1. explain why
    // sizeof(char)=1, sizeof(int)=4, sizeof(char*)=4
    // Inside the class Header, used(char) is followed by payloadsize(int), which
is larger in size as compared to used(char). Hence, padding, which is 3, is added
after used(char) to align the structure. Therefore, the total size is 12.

    cout << "Header object size " << sizeof (h1) << endl;
    // 2. explain why
    // The size of h1 is just the size of Header by default constructor, so
sizeof(h1) equals to 12.
```

```
    cout << "Header object h2 size "<< sizeof (h2) << endl;
    // 3. explain why
    // In this case, the second constructor sets payloadsize(int) to 10 and set
the pointer to new char[10]. Since the change of integer value and pointer address
won't change their size, the total size of h2 is 12.

    cout << "Header object pointer size " << sizeof (h3) << endl;
    // 4. explain why
    // By calling "Header* h3 = new Header (20)", it sets h3 to a pointer which
has the size of 4, so its data type doesn't matter anymore. Therefore, the total
size of h3 is 4.

    // 5. now allocate memory big enough to hold 10 instances of Header
    Header* container = (Header*)malloc(10 * sizeof(Header));

    // 6. Put 10 instances of Header in the allocated memory block, one after
another without overwriting.
    //The instances should have payload size 10, 20,..., 100 respectively
and they should have initial values 1,2,...10 respectively
    for (int i = 0; i < 10; i++) {
        container[i] = Header(10*(i+1), i+1);
    }

    // 7. now call getsummation() on each instance using a loop, output should be:
10, 40, 90, ...., 1000 respectively
    for (int i = 0; i < 10; i++) {
        cout << container[i].getsummation() << " ";
    }
    cout << endl;

    Header* ptr = h3 + 100;
    cout <<"Printing pointer h3: " << h3 << endl;
    cout <<"Printing pointer ptr: " << ptr << endl;

    // 8. explain the output you see in the following
    // The difference of ptr and h3 is 100 objects, and the difference of (char*)
ptr and (char*) h3 is 1200 bytes. Since ptr is a Header pointer as we defined and
equal to "h3 + 100", it means ptr points to 100 elements past h3 in memory.
Therefore, the difference of ptr and h3 is 100 objects. While subtracting (char*)
ptr from (char*) h3, it is subtracting the memory bytes which is the difference of
memory address of ptr and h3. As each element in Header pointer is 12 bytes, the
difference of 100 objects is 1200 bytes.

    cout << "Difference " << (ptr - h3) << " objects" << endl;
    cout << "Difference " << ((char*) ptr - (char *)h3) << " bytes" << endl;

}
```