National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Advanced Operating Systems	Course Code:	CS-505
	Program:	MS(Computer Science)	Semester:	Fall 2018
	Duration:	3 hours	Total Marks:	70
	Paper Date:	24^{th} December, 2018	Weight:	50%
	Section:	All	Page(s):	5
	Exam:	Final	Roll No.	

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, **cutting and blotting on this sheet will result in deduction of marks**. The space given for the answers is more than enough in all cases.

Question 1 (5 points): Reorder the following steps in handling a page fault so that it is correct:

1. The OS restarts the interrupted instruction
2. CPU executes a memory referencing instruction
3. The OS swaps out the victim page
4. The OS loads the desired page
5. MMU looks up the TLB to find invalid entry

Enter the correct order of steps in the box below

Question 2 (5 points): Depending upon what IPC technique will be suited best for the requirements given below, write ”shared memory” or ”message passing” in front of each of the following phrases:

1. matrix multiplication
2. multiple processes residing on different machines
3. email
4. multiple processes residing on same machine
5. the IPC method that requires synchronization

Question 3 (10 points): Give all possible outputs for the following program.

```
int main() {
        int x = 1;
        ++x;    // 2
        cout << x << endl;
        pid_t pid = fork();
        if (pid == 0) {
                ++x;
                cout << x << endl;
        }
        else if (pid > 0) {
                x = x + 3;
                cout << x << endl;
        }
        ++x;
        cout << x << endl;
        return 0;
}
```

Output 1	Output 2	Output 3	Output 4	Output 5	Output 6

Question 4 (10 points): Implement the First in First Out Scheduling algorithm using the following declarations. **Hint:** There is no loop required in the solution.

```
class ProcessQ{ // it is the list of all processes ready to run.
public:
void addToQ(int pid); // adds a process to the ready queue.
int removeFromQ(); // removes a process from the ready queue. If queue is empty it returns
    NULL.
};
//-------------------------------------------------------------------------------------
ProcessQ getReadyQ() // returns the ready queue of the system.
```

```
int getNextProcessToRun(int leavingProcess) // Returns the process id which should run next.
    The parameter is the ID of the process which is reliquishing CPU.
{




}
```

Question 5 (10 points): A node in Memnet ring wants to read a byte from some block. Implement the function which does the task. Use following function declarations. Do not use any semaphores here. **Hint:** There is no loop required in the solution.

```
class Token; // represents a token in the ring, definition is not needed here.
bool isBlockPresent(int blockNumber);// checks whether the block is present on the local
    memory or not.
Token waitForToken(); // used to wait for the token to come. If token has reached it will
    return the token, else it will put the process in wait. Meaning it will always return the
    valid token.
Token prepareReadToken(int blockNumber);// prepares a token to send to others. The token
    contains the request to read a block, identified by blockNumber.
void sendToken(Token tok);// sends any token on the ring.
void saveBytesFromTokenToMemory(Token tok);// reads the token and extracts the block of bytes
    into memory.
byte* getBlockAddress(int blockNumber);// finds the block in the memory and return the address
     of its first byte.
```

```
byte readByte(int blockNumber, int byteNumber) // Returns the value of the byte to be read.
{









}
```

Question 6 (10 points): Implement leader election algorithm for ring topology. You have to implement the function `onReceiveMessage`. Use following declarations. **Hint:** Not more than one loop is required in the solution.

```
1 class LeaderElectMsg{ // A class to be used for the message in leader election.
2 public:
3 int* procs; // array of processes
4 void addToArray(int pid); // adds a process specified by 'pid' into the array 'procs'.
5 int numProcs(); // returns the number of processes in the array 'procs'.
6 };
7 void forwardMessage(LeaderElectMsg msg, int procID);// forwards the message to the process
      specified by the paramater 'procID'
8 int getCurrentProcessID();//returns the process id of the current process
9 int getNextProcessID();//return the process id of the next process in the ring.
```

```
int onReceiveMessage(LeaderElectMsg msg)// returns the process ID of the leader, if found,
    otherwise forwards the message and returns -1.
{









}
```

Question 7 (10 points): Two processes P_1, P_2 having their own caches use bus snooping to keep their caches consistent, by using write-once protocol. In the beginning their caches are empty. Following is the sequence of actions performed by different processes. You have to tell the states of all caches in result of the actions. Draw diagrams just like the ones shown in slides and book.

1 Process P_1 reads a word W_1 from memory

2 Process P_2 reads a word W_1 from memory

3 Process P_2 changes the word W_1

4 Process P_1 changes the word W_1

5 Process P_2 reads a word W_1 from memory

Question 8 (10 points): Consider the following code for a simple Stack. You can see from the code that a process blocks if it calls push() when the stack is full, or it calls pop() when the stack is empty, the same behavior should be present in the answer. Assuming that the functions push and pop can execute concurrently, synchronize the code using semaphores. Also eliminate the busy waiting. For **partial marks** you can only remove the busy waiting.

```
1  class Stack {
2  private:
3          int* a; // array for stack
4          int max;         // max size of array
5          int top;         // stack top
6  public:
7          Stack(int m) {
8                  a = new int[m]; max = m;         top = 0;
9          }
10         void push(int x) {
11                 while (top == max)
12                 ;        // if stack is full then wait
13                 a[top] = x;
14                 ++top;
15         }
16         int pop() {
17                 while (top == 0)
18                 ;        // if stack is empty then wait
19                 int tmp = top;
20                 --top;
21                 return a[tmp];
22         }
23 };
```