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:	1.5 hour	Total Marks:	50
	Paper Date:	5^{th} November, 2018	Weight:	20%
	Section:	All	Page(s):	3
	Exam:	Mid	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**.

Question 1 (2 points): In a multitasking environment, if a process is continuously denied necessary resources, then the problem is called

1. deadlock
2. **starvation**
3. inversion
4. aging

Question 2 (2 points): The processes that are ready and waiting to execute are kept in a list called

1. job queue
2. **ready queue**
3. execution queue
4. process queue

Question 3 (2 points):
If a page fault occurs and there are no free frames in the memory, then how many pages have to be transferred to and from main memory.

1. one
2. **two**
3. three
4. four

Question 4 (2 points): Threads within a program do not share

1. Data Section
2. Code Section
3. **Stack**
4. Files

Question 5 (2 points): Multithreading provides efficiency if and only if we have

1. At least two processors
2. At least three processors
3. Large RAM
4. **Even without the above**

Question 6 (2 points): Threads are economical as they

1. Share data section
2. Take less time to start
3. Take less time to context switch
4. **All of the above**

Question 7 (8 points): Suppose that each memory page in our operating system has a size of 200 bytes. The frame size is also equal to the page size. Provide the physical addresses of the bytes, given their logical addresses as under. The table given below shows which page is loaded onto which frame. (**Note:** recall that although the logical address here is two dimensional, the physcial address is always one dimensional.)

(a) (3,30) = $200 \times 13 + 30 = 2630$

(b) (6,10) = $200 \times 11 + 10 = 2210$

(c) (8,0) = $200 \times 3 + 0 = 600$

(d) (7,23) = $200 \times 7 + 23 = 1423$

Page #	1	2	3	4	5	6	7	8	9	10	11	12	13
Frame #	5	9	13	2	8	11	7	3	6	4	12	10	1

Question 8 (10 points): Given the table, the highest priority task should run before the lower one. For same priority tasks use a scheme which gives the minimum waiting time. Write the **order of execution** and calculate the **waiting time** in the end. Give **reason** (or proof), why under the given conditions, is it impossible to achieve lower waiting time than your scheme?

Process ID	Time of Arrival	Priority	Computation Time
1	20	1	10
2	10	1	20
3	0	1	30
4	20	2	100
5	20	2	10
6	20	2	20
7	30	3	30
8	30	3	60
9	30	3	20
10	40	4	10
11	40	4	50

As we want to have the minimum waiting time, and we know that **shortest job first** algorithm gives the minimal waiting time, so we will use SJF in accord with the priorities. We will use non-preemptive algorithm because, preemption itself introduces dispatch latency which increases waiting time. So the order is as follows: 3,1,2,5,6,4,9,7,8,10,11

The following formula gives the average waiting time. The values are given in the order of execution listed above:

$$\frac{0+10+30+40+50+70+160+180+210+260+270}{11} = \frac{1280}{11} \tag{1}$$

Question 9 (10 points): Three processes are running in parallel sharing variables i and j, which are initialized as $i = 1$ and $j = 1$. You have to synchronize the following processes such that the output on the console is the Fibonacci series $1, 1, 2, 3, 5 \ldots$

NOTE: You can change the following code only bt calling the **wait** and **signal** methods of semaphores, and putting if else conditions. You are allowed to use as many semaphores as you like. No other change in the code is allowed. Also, you have to mention the initial values of all the semaphores. You cannot add any statement which alters the values of variables i and j.

Process 1

```
while(true)
{

  j = i + j

}
```

Process 2

```
while(true)
{

  i = i + j

}
```

Process 3

```
while(true)
{

  cout << j<<",";
  cout << i<<",";

}
```

All semaphores are initialized to 0, except sem_3 = 1

Process 1

```
while(true)
{
        sem_1.wait()
        j = i + j
        sem_2.post()

}
```

Process 2

```
while(true)
{
        sem_2.wait()
        i = i + j
        sem_3.post()
}
```

Process 3

```
while(true)
{
        sem_3.wait();
        cout << j<<",";
        cout << i<<",";
        sem_1.post();

}
```

Question 10 (10 points): A VMM is implemented using trap-and-emulate scheme. A process inside a VM calls a privileged instruction. The VMM takes necessary steps according to trap-and-emulate scheme. Write the code to complete this by using the helper functions. The function `handleTrap` is called by the VMM when a process inside a VM executes a privileged instruction.

```

CPU_State executeInstrcution(int command); // takes the op-code of the privileged instruction
    and executes it on the CPU. Returns the resulting state of the CPU.
int getVCPU_ID(int VM_ID);// takes ID of the VM and returns the ID of the virtual CPU on which
     it is running
void updateVCPU(int VCPU_ID,CPU_State state);// takes ID of the VCPU and the state of the CPU
    to update it on VCPU
```

```
int handleTrap( int command, int VM_ID) // First parameters is the privileged instruction,
    second is the ID of the Virtual Machine.
{
        CPU_State result = executeInstruction(command);
        int VCPU_ID = getVCPU_ID(VM_ID);
        updateVCPU(VCPU_ID,result);

}
```