


National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2018
	Duration:	1 hour	Total Marks:	50
	Paper Date:	16 th November, 2018	Weight:	15%
	Section:	All	Page(s):	3
	Exam:	Mid-2	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 (10 points): Write the code to fetch a byte from permanent storage using FAT file system. The helper functions are given below.

```

1  class FCB;
2  class FATHelper
3  {
4      public FATHelper(string partition); // the constructor takes the partition name as
        input and loads its FAT table
5      public int getBlockSize(); // returns the block size of the partition.
6      public FCB findFCB(string path); // takes the path of a file and returns it FCB
7      public int getDataBlock(FCB fcb, int logicalblock); // takes the FCB and the logical
        block number, and returns the physical block number of the respective logical
        block.
8      public byte* readData(int physicalBlock); // takes the physical block number as input
        and returns the data written on it in form of a byte array
9  };

```

```

byte getByte(string partition, string path, int byteNumber) // takes partition name, path of
the file and the byte number as input and returns the data written on that byte.
{
    FATHelper helper(partition);
    FCB fcb = helper.findFCB(path);
    int logicalBlock = byteNumber/helper.getBlockSize();
    int block = helper.getDataBlock(fcb, logicalBlock);
    byte* data = helper.readData(block);
    return data[byteNumber%helper.getBlockSize()];
}

```

Question 2 (10 points): We have following functions written for producer and consumer. Assume that the **buffer** is an infinite list. Now **identify** and **fix** the problem. The fix needs only repositioning two statements. Fill the blanks below to identify and fix the problem. **Note:** All variables are dealt in terms of pointers.

Producer	Consumer
sem_1=1, sem_2=0, buffer // among shared variables buffer is an infinite list of elements , rest are semaphores	
<pre> 1 void *producer(void *param) 2 { 3 void* item = NULL; 4 while(true) 5 { 6 item = produce(); 7 sem_wait(sem_1); 8 buffer->add(item); 9 sem_post(sem_2); 10 sem_post(sem_1); 11 } 12 } </pre>	<pre> 1 void *consumer(void *param) 2 { 3 void* item = NULL; 4 while(true) 5 { 6 sem_wait(sem_1); 7 sem_wait(sem_2); 8 item = buffer->get(); 9 sem_post(sem_1); 10 process(item); 11 } 12 } </pre>

1. The problem comes when the function _____ is being executed. At the start of while loop

(line 5) the value of semaphore **sem_1** is _____ and the value of **sem_2** is _____. This type of problem is called _____

2. In order to fix the problem, we just need to swap the code written at line _____ with the code written at line _____, in the function _____

1. The problem comes when the function **consumer** is being executed. At the start of while loop (line 5) the value of semaphore **sem_1** is **1** and the value of **sem_2** is **0**. This type of problem is called **deadlock**

2. In order to fix the problem, we just need to swap the code written at line **6** with the code written at line **7**, in the function **consumer**

Question 3 (10 points): The above functions are written in the C++ syntax needed for the synchronization. If you look carefully then we can see that the functions **producer** and **consumer** have signatures suitable enough to be called in separate threads. Write the main function below which initializes the semaphores, then starts the functions **producer** and **consumer** in separate threads and waits for them to join.

```
1  sem_t s1;
2  sem_t s2;
3
4  sem_t* sem_1=NULL;
5  sem_t* sem_2=NULL;
6
7
8  int main ()
9  {
10
11      sem_1=&s1;
12      sem_2=&s2;
13
14      pthread_t tid1, tid2;
15
16      if (sem_init(sem_1, 0, 1) == -1)
17      {
18          cout << "semaphore creation error";
19          exit(-1);
20      }
21
22      if (sem_init(sem_2, 0, 0) == -1)
23      {
24          cout << "semaphore creation error";
25          exit(-1);
26      }
27
28      if (pthread_create(&tid1, NULL, producer, NULL)!=0)
29      {
30          cout <<"thread creation error";
31          exit(-1);
32      }
33      if (pthread_create(&tid2, NULL, consumer, NULL)!=0)
34      {
35          cout <<"thread creation error";
36          exit(-1);
37      }
38
39      if (pthread_join(tid1, NULL)!=0)
40      {
41          cout <<"thread join error";
42          exit(-1);
43      }
44      if (pthread_join(tid2, NULL)!=0)
45      {
46          cout <<"thread join error";
47          exit(-1);
48      }
49
50 }
```

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

- | | |
|-----------------|-----------------|
| 1. Data Section | 3. Stack |
| 2. Code Section | 4. Files |

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

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

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

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

Question 7 (2 points): In a multitasking environment, if a process is continuously denied necessary resources, then the problem is called

- | | |
|----------------------|--------------|
| 1. deadlock | 3. inversion |
| 2. starvation | 4. aging |

Question 8 (2 points): Among the following, which function on files is counter productive in sequential storage mediums like tape drives

- | | |
|----------|----------------|
| 1. open | 3. seek |
| 2. close | 4. create |

Question 9 (2 points): Contiguous allocation of files may have following problems. Tick all correct.

- | | |
|---|----------------------------------|
| 1. Creation of file is slow | 3. Deleting a file is slow |
| 2. Enlarging the file size is slow | 4. Searching a file is difficult |

Question 10 (2 points): Named pipes can operate in a situation where the two processes reside on different machines

- | | |
|---------|-----------------|
| 1. True | 2. False |
|---------|-----------------|

Question 11 (2 points): We may use shared memory when processes reside on different machines, even without any extra driver or software package.

- | | |
|---------|-----------------|
| 1. True | 2. False |
|---------|-----------------|

Question 12 (2 points): Thread creation is a costly procedure. So in practice **thread pools** are used to allocate designated number of threads to a process and making thread creation faster.

Question 13 (2 points): User level threading libraries implement following threading model.

- | | |
|------------------------|---------------|
| 1. Many to Many | 3. One to one |
| 2. Many to one | |