Operating Systems
Spring 2011 Mid #2
Time 1½ Hour – Marks 50

Q1. Consider the following set of processes, with the length of the CPU burst time given in milliseconds. (10+5+5)

Process	Burst Time	Priority	Arrival time
P1	6	3	2
P2	7	2	3
P3	6	5	4
P4	4	1	4
P5	3	4	5

A. Consider the execution of these processes using SJF and Preemptive SJF Scheduling algorithms, draw Gantt charts for each process for each of the scheduling algorithms.
B. What is the response time of each process for each of the scheduling algorithms?
C. What is the turnaround time of each process for each of the scheduling algorithms?

Q2. What are the three basic conditions for the solution of a Critical Section problem? Define and explain each of them. (10)

Q3. The following code segments are two independent threads of the same program manipulating a circular queue tshaff. Is there any critical section code? If yes, then point out and elaborate the operation where mutual exclusion is required; and ensure mutual exclusion using semaphores. (10)

Operating Systems
Midterm II
Time 1½ Hour Marks 40

Q1. Five parallel threads of a program are acquiring data from some sources that we are not interested in and places it in a buffer dbuf. Ten threads are there to process data that is placed in a buffer. Following is a code piece of both the threads: **(20)**

```
void getData (void) {
...
while (1) {
    while (dhead == BUFSIZE);
    obj=readData();
    dbuf[dhead++]=obj;
    dhead%=BUFSIZE;
}
}
```

```
void processData(void) {
while (1) {
    while (dhead==dtail);
    //buffer is empty
    robj=dbuf[dtail++];
    dtail%=BUFSIZE;
    process(robj);
}
}
```

Identify the critical sections and busy waits in these threads and appropriately take care of them with the use of mutexes and semaphores. You should select the best possible place to insert these synchronization primitives, your marks will depend on your choices.

Q2. Give brief and complete answers to the following questions: **(5X4=20)**

a. What is the reason of using TLBs in a paging system?
b. Why do we make multilevel page tables?
c. What is inverted page table?
d. What is the function of D (dirty) and A (accessed) bits in a virtual memory system?

Q3. Following is a reference string of a program. How many page faults are going to occur and on which references if you use:

a) FCFS (FIFO) b) LRU

page replacement algorithms with a quota of three frames. (10)

1,2,3,1,4,3,1,5,6,3,1,2,3,7,6,3,2,1,2,3,6

Q2. What are the three basic conditions for the solution of a Critical Section problem? Define and explain each of them. (10)

Q3. The following code segments are two independent threads of the same program manipulating a circular queue txbuff. Is there any critical section code? If yes, then point out and elaborate the operation where mutual exclusion is required and ensure mutual exclusion using semaphores. (10)

```
void transmit(unsigned char c) {
  while ((txhead+1)==txtail);
  txbuff[txhead++]=c;
  txhead%=8;
}
```

```
void ser0(void) {
  if (TI) {
    TI=0;
    if (txhead!=txtail) {
      SBUF=txbuff[txtail++];
      txtail%=8;
      if (txhead==txtail) {
        txhead=txtail=0;
      }
    }
  }
```

Q4. Consider the following table: (10)

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical address generated against the following logical addresses: 0:430, 1:10, 2:500, 3:400, 4:112

GOOD LUCK

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Operating Systems	Course Code:	CS2006
	Degree Program:	BSCS	Semester:	Spring 2023
	Exam Duration:	60 Minutes	Total Marks:	45
	Paper Date:	10-April-2023	Weight	15%
	Section:	ALL Sections	Page(s):	3
	Exam Type:	Mid 2 Exam		

Student : Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions on the given answer sheet. Clearly mention your attempted question no. with the answers on the answer sheet. Avoid unnecessarily explanation. Kindly write your information in above mentioned space. Attach this question paper with your answer sheet.

CLOs	CLO-3	CLO-2	CLO-5	
Questions	Q-1	Q-2	Q-3	Total
Total Marks	20	10	15	**45**
Marks Obtained				

Question 01: (20 points) (CLO-3)
Suppose that there is a single common room in the university. When a female student is in the common room, other female students may enter, but no male student, and vice versa. A sign with a sliding marker on the door of each common room indicates which of the three possible states it is currently in:

- Empty
- Female Student present
- Male Student present

Write the following procedures in order to synchronize the above scenario:

1) **female_student_wants_to_enter**
2) **male_student_wants_to_enter**
3) **female_student_leaves**
4) **male_student_leaves**

You may use whatever counters, semaphores and synchronization techniques you like.

Question 02: (10 points) (CLO-2)
Assuming calls to all library routines succeed. What is output of the following code?

```
void *printer(void *arg) {
  char *p = (char *) arg;
  printf("%c", *p);
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p[5];
  for (int i = 0; i < 5; i++) {
    char c = 'a' + i;
    pthread_create(&p[i], NULL, printer, (void *) &c);
  }
  for (int i = 0; i < 5; i++)
    pthread_join(p[i], NULL);
  return 0;
}
```

Question 03: (15 points) (CLO-5)

Consider the following snapshot of a system:

	Allocation				**Max**				**Available**			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the **banker's algorithm:**

a) What is the content of the matrix **Need**?

b) Is the system in a **Safe State**?

c) If a request from process **P1** arrives for **(0, 4, 2, 0)**, can the request be granted immediately?

Only For Sections BCS-4F, 4G, 4H and 4J

Question 03: (15 points) (CLO-5)
Can you explain the differences between deadlock and starvation in concurrent programming? Please provide coding examples and real-world scenarios to illustrate these concepts.

National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	**Operating Systems**	**Course Code:**	**CS 205**
Program:	**Bachelors in Computer Science**	**Semester:**	**Spring 2019**
Duration:	**60 minutes**	**Total Marks:**	**30**
Paper Date:	**12th April 2019**	**Weight**	**15**
Section:	**ALL**	**Page(s):**	**3**
Exam Type:	**Mid 2**		

Student : Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Q1: Choose the correct answer. [10 points]

1. Identify the one which is not an advantage of threads
   a. Is lightweight
   b. Increases efficiency
   c. Allows scalability
   d. Enables multiprocessing on uniprocessor system

2. Threads within a process do not share
   a. Data
   b. Files
   c. Registers
   d. code

3. It is possible to achieve concurrency with one processor but it is not possible to achieve parallelism with one processor.
   a. True
   b. False

4. Which of the following multithreading models is efficient as well as avoids blocking issues
   a. One to many
   b. Many to many
   c. Many to one

5. The state where 1 process out of total 4 processes is unable to acquire CPU is known as:
   a. Deadlock
   b. Starvation
   c. Priority inversion
   d. Bounded Waiting

6. The `pthread_join(workerthread, NULL)` does the following:
   a. `workerThread` waits for calling thread
   b. calling thread waits for `workerThread`
   c. calling and `workerThread` execute independently
   d. Calling and `workerThread` implement mutual exclusion

7. Which of the following is not a condition for synchronization problems solution
   a. Relative speed
   b. Mutual Exclusion
   c. Progress
   d. Bounded waiting

8. Peterson solution is considered as the primitive solution to synchronization problems because
   a. It lacks synchronization capabilities
   b. Uses too much memory
   c. Only provides synchronization between two processes
   d. Does not allow *progress*

9. Which of the following is not a solution to synchronization problems
   a. Monitors
   b. Using `block()` and `wait()` operations
   c. Mutex
   d. Semaphore

10. Which of the following is a not an advantage of semaphores
    a. Solving synchronization problem
    b. Signaling
    c. Resource utilization management
    d. Efficient hardware utilization

Q 02: [a – 6 points] Consider the following concurrently executing processes. Synchronize them using semaphores to generate the following infinite string: ”yesyesyesyes...”. Your solution should use the minimum possible number of semaphores. Please add statements to the code below without modifying any existing statements.

Initialize: semaphore s1 = 1, s2 = 0, s3 = 0;

Process 1	Process 2	Process 3

while(true)
{
wait(s1);
cout << "y";
signal(s2);
}
	while(true)
	{
	wait(s2);
	cout << "e";
	signal(s3);
	}
		while(true)
		{
		wait(s3);
		cout << "s";
		signal(s1);
		}

Q 02: [b – 4 points] Consider the following set of semaphores and their initialization values:

Semaphore car-avail = 0, car-taken=0; car-filled=0;

Assume that the **implementation of semaphores uses waiting queues instead of busy waiting**.

Your job is to keep track of the number of processes waiting in the queue for each semaphore. Given the values above and the statements given in the table below, fill the fields (car-avail, car-taken, car-filled) with integer values such that the number of processes waiting in the waiting queues for each semaphore are demonstrated.

Note: For each column, fill the cells with integers where there is a change in value ONLY. Leave a “-” otherwise

Statements	car-avail	car-taken	car-filled
Wait (car-avail)	-1	0	0
Wait (car-avail)	-2	0	0
Signal(car-avail)	-1	0	0
Signal(car-avail)	0	0	0
Signal(car-taken)	0	1	0
Wait(car-filled)	0	1	-1
Wait(car-taken)	0	0	-1
Wait(car-taken)	0	-1	-1

Q 03 [4+6 points]: There are two code segments given in this question. Part A whose output has to be provided is given in the left box. Part B has to be answered in the right box containing the question.

Question statement Part B:
The following program finds the maximum value in an array in parallel. For simplicity, size of the array and number of threads are chosen as 100 and 4 respectively. Assume that the size of the array is perfectly divisible by

the number of threads. Also, assume that there is no compilation error. The code for #include statements and the init() function is not provided for brevity. The init() function initializes elements of the array to random values between 1 and 1000. The code is executing on a laptop with 8 idle cores.

Main requirements: The program must divide the work between the 4 threads and they must run in parallel.

Your job here is to find and fix a subtle benign bug in the code that is hindering the code from executing efficiently and meeting main requirements mentioned above. Just modify the code clearly in the box below. No need to rewrite. Hard-coded line modifications will be penalized. (6 marks)

(Part A) Provide output at the bottom

```
#include <pthread.h>
#include <stdio.h>
int value = 0;

void *runner(void *param); /* the thread */

int main(int argc, char *argv[]) {

  int pid;
  pthread_t tid;
  pid = fork();

  if (pid == 0) {

     pthread_create(&tid, NULL, runner, NULL);
     pthread_join(tid, NULL);
     printf("CHI: val = %d \n", value); /*LINE C*/

  } else if (pid > 0) {

     value -= 5;
     wait(NULL);
     printf("PAR: val = %d \n", value); /*LINE P*/

  }
}

void *runner(void *param) {
          value += 5;
          pthread_exit(0);
}
```

Write the output of the code above at **LINES C** and **P** in this box. (4 marks)

Output at LINE C = ____5_____

Output at LINE P = _____-5_____

(Part B)

```
#define SIZE 100
#define THREADS 4

int retval[THREADS];

int buffer[SIZE];
void* max_find (void*);
void init (int *buffer);

int main() {
   init (buffer);
   pthread_t tid[THREADS];

   for(int i = 0; i < THREADS; i++)
      pthread_create (&tid[i], NULL, max_find, (void*) i);

   for(int i = 0; i < THREADS; i++)
      pthread_join (tid[i], NULL);

   int max_val = retval[0];
   for(int i=1; i < THREADS; i++)
      if(retval[i] > max_val)
         max_val = retval[i];

   printf("The maximum value is %d\n\n", max_val);
   return 0;
}

void* max_find (void* args) {
   int myid = (int) args;
   int start = 0;   //int start = mynum * SIZE/THREADS;
   int end = SIZE;  //int end = start + SIZE/THREADS;
   int localmaxval = buffer[start];

   for(int i = start; i < end; i++)
      if(buffer[i] > localmaxval)
         localmaxval = buffer[i];
   retval[myid] = localmaxval;
}
```

National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	**Operating Systems**	**Course Code:**	**CS 205**
Program:	**Bachelors in Computer Science**	**Semester:**	**Spring 2019**
Duration:	**60 minutes**	**Total Marks:**	**30**
Paper Date:	**12th April 2019**	**Weight**	**15**
Section:	**ALL**	**Page(s):**	**3**
Exam Type:	**Mid 2**		

Student : Name:_____ Roll No._____ Section:_____

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Q1: Choose the correct answer. [10 points]

1. Identify the one which is not an advantage of threads
   a. Is lightweight
   b. Increases efficiency
   c. Allows scalability
   d. Enables multiprocessing on uniprocessor system

2. Threads within a process do not share
   a. Data
   b. Files
   c. Registers
   d. code

3. It is possible to achieve concurrency with one processor but it is not possible to achieve parallelism with one processor.
   a. True
   b. False

4. Which of the following multithreading models is efficient as well as avoids blocking issues
   a. One to many
   b. Many to many
   c. Many to one

5. The state where 1 process out of total 4 processes is unable to acquire CPU is known as:
   a. Deadlock
   b. Starvation
   c. Priority inversion
   d. Bounded Waiting

6. The `pthread_join(workerthread, NULL)` does the following:
   a. `workerThread` waits for calling thread
   b. calling thread waits for `workerThread`
   c. calling and `workerThread` execute independently
   d. Calling and `workerThread` implement mutual exclusion

7. Which of the following is not a condition for synchronization problems solution
   a. Relative speed
   b. Mutual Exclusion
   c. Progress
   d. Bounded waiting

8. Peterson solution is considered as the primitive solution to synchronization problems because
   a. It lacks synchronization capabilities
   b. Uses too much memory
   c. Only provides synchronization between two processes
   d. Does not allow *progress*

9. Which of the following is not a solution to synchronization problems
   a. Monitors
   b. Using `block()` and `wait()` operations
   c. Mutex
   d. Semaphore

10. Which of the following is a not an advantage of semaphores
    a. Solving synchronization problem
    b. Signaling
    c. Resource utilization management
    d. Efficient hardware utilization

Q 02: [a – 6 points] Consider the following concurrently executing processes. Synchronize them using semaphores to generate the following infinite string: ”yesyesyesyes...”. Your solution should use the minimum possible number of semaphores. Please add statements to the code below without modifying any existing statements.

Process 1	Process 2	Process 3

while(true)
{
 	
cout << "y";
 	
}
	while(true)
	{
	
	cout << "e";
	
	}
		while(true)
		{
		
		cout << "s";
		
		}

Q 02: [b – 4 points] Consider the following set of semaphores and their initialization values:

Semaphore car-avail = 0, car-taken=0; car-filled=0;

Assume that the **implementation of semaphores uses waiting queues instead of busy waiting**.

Your job is to keep track of the number of processes waiting in the queue for each semaphore. Given the values above and the statements given in the table below, fill the fields (car-avail, car-taken, car-filled) with integer values such that the number of processes waiting in the waiting queues for each semaphore are demonstrated.

Note: For each column, fill the cells with integers where there is a change in value ONLY. Leave a “-” otherwise

Statements	car-avail	car-taken	car-filled
Wait (car-avail)			
Wait (car-avail)			
Signal(car-avail)			
Signal(car-avail)			
Signal(car-taken)			
Wait(car-filled)			
Wait(car-taken)			
Wait(car-taken)			

Q 03 [4+6 points]: There are two code segments given in this question. Part A whose output has to be provided is given in the left box. Part B has to be answered in the right box containing the question.

Question statement Part B:
The following program finds the maximum value in an array in parallel. For simplicity, size of the array and number of threads are chosen as 100 and 4 respectively. Assume that the size of the array is perfectly divisible by

the number of threads. Also, assume that there is no compilation error. The code for #include statements and the init() function is not provided for brevity. The init() function initializes elements of the array to random values between 1 and 1000. The code is executing on a laptop with 8 idle cores.

Main requirements: The program must divide the work between the 4 threads and they must run in parallel.

Your job here is to find and fix a subtle benign bug in the code that is hindering the code from executing efficiently and meeting main requirements mentioned above. Just modify the code clearly in the box below. No need to rewrite. Hard-coded line modifications will be penalized. (6 marks)

(Part A) Provide output at the bottom

```
#include <pthread.h>
#include <stdio.h>
int value = 0;

void *runner(void *param); /* the thread */

int main(int argc, char *argv[]) {

  int pid;
  pthread_t tid;
  pid = fork();

  if (pid == 0) {

     pthread_create(&tid, NULL, runner, NULL);
     pthread_join(tid, NULL);
     printf("CHI: val = %d \n", value); /*LINE C*/

  } else if (pid > 0) {

     value -= 5;
     wait(NULL);
     printf("PAR: val = %d \n", value); /*LINE P*/

  }
}

void *runner(void *param) {
          value += 5;
          pthread_exit(0);
}
```

Write the output of the code above at **LINES C** and **P** in this box. (4 marks)

Output at LINE C = _____

Output at LINE P = _____

(Part B)

```
#define SIZE 100
#define THREADS 4

int retval[THREADS];

int buffer[SIZE];
void* max_find (void*);
void init (int *buffer);

int main() {
   init (buffer);
   pthread_t tid[THREADS];

   for(int i = 0; i < THREADS; i++)
      pthread_create (&tid[i], NULL, max_find, (void*) i);

   for(int i = 0; i < THREADS; i++)
      pthread_join (tid[i], NULL);

   int max_val = retval[0];
   for(int i=1; i < THREADS; i++)
      if(retval[i] > max_val)
         max_val = retval[i];

   printf("The maximum value is %d\n\n", max_val);
   return 0;
}

void* max_find (void* args) {
   int myid = (int) args;
   int start = 0;
   int end = SIZE;
   int localmaxval = buffer[start];

   for(int i = start; i < end; i++)
      if(buffer[i] > localmaxval)
         localmaxval = buffer[i];
   retval[myid] = localmaxval;
}
```

Operating System - Mid-term II

Question1 [Pts:20 (10+10)]

a. Explain precisely how the situation of a complete traffic jam in a four-way crossing/chowke technically merits being a deadlock. Explain this by showing how four conditions of the deadlock are held in this situation.
Answer:

1. Mutual exclusion condition applies, since only one vehicle can be on a section of the street at a time.
2. Hold-and-wait condition applies, since each vehicle is occupying a section of the street, and waiting to move on to the next section of the street.
3. No-preemptive condition applies, since a section of the street that is a section of the street that is occupied by a vehicle cannot be taken away from it.
4. Circular wait condition applies, since each vehicle is waiting on the next vehicle to move. That is, each vehicle in the traffic is waiting for a section of street held by the next vehicle in the traffic.

b. Explain the difference between binary semaphore and mutex.

The mutex is similar to the principles of the binary semaphore with one significant difference: the principle of ownership. Ownership is the simple concept that when a task locks (acquires) a mutex only it can unlock (release) it. If a task tries to unlock a mutex it hasn't locked (thus doesn't own) then an error condition is encountered and, most importantly, the mutex is not unlocked. If the mutual exclusion object doesn't have ownership then, irrelevant of what it is called, it is not a mutex.

Question 2[Pts:15]

Give a solution for the problem of metro bus service in Lahore. There is a one-way narrow bridge on ravi where metro buses coming from opposite directions cannot cross. This means that if a bus enters from south i.e. Lahore (let's call it southbus) then till the time it crosses the bridge and the bridge is clear, no bus from north i.e. Shahadara (let's call it northbus) may enter it, and vice versa. Also, another southbus must be allowed to enter if another southbus is already using the bridge and vice versa. The traffic controller has contacted you for your repute in OS concepts to give an amicable solution. Using your synchronization

knowledge, you are required to implement a starvation free solution. Use the following template skeleton to complete give the

Southbus(){	Northbus(){
//approaching bridge entrance	//approaching bridge entrance
//your synchronization logic goes here	//your synchronization logic goes here
Pass the bridge code.	Pass the bridge code.
// exiting the bridge	// exiting the bridge
// and your synchronization logic goes here	// and your synchronization logic goes here
}	}

Question 3[Pts: 15]

You are required to implement an algorithm of Coke machine which is a shared buffer

Two types of users

- Producer: Restocks the coke machine
- Consumer: Removes coke from the machine

Requirements

- Only a single person can access the machine at any time.
- If the machine is out of coke, wait until coke is restocked.
- If machine is full, wait for consumers to drink coke prior to restocking.
- If the machine is not full, producer can produce as many cokes as he likes (NOT more than machine's capacity)
- If the machine is not empty, consumer can consume as many cokes as they likes (until the machine is empty)

Solution:

```
shared binary semaphore mutex = 1;
shared counting semaphore empty = MAX;
shared counting semaphore full = 0;
shared anytype buffer[MAX];
shared int in, out, count;
```

PRODUCER :

```
anytype item;

repeat {

        /* produce something */
        item = produce();

        /* wait for an empty space */
        wait(empty);

        /* store the item */
        wait(mutex);
        buffer[in] = item;
        in = in + 1 mod MAX;
        count = count + 1;
        signal(mutex);

        /* report the new full slot */
        signal(full);

} until done;
```

CONSUMER:

```
anytype item;

repeat {

        /* wait for a stored item */
        wait(full);

        /* remove the item */
        wait(mutex);
        item = buffer[out];
        out = out + 1 mod MAX;
        count = count - 1;
        signal(mutex);

        /* report the new empty slot */
        signal(empty);

        /* consume it */
        consume(item);

} until done;
```

Operating Systems (CS2006)

Sessional-II Exam

Date: November 5th 2024

Course Instructor(s)

Mr. Razi Uddin
Mr. Mubashar Hussain
Ms. Rubab Anam
Ms. Namra Absar

Total Time (Hrs): 1
Total Marks: 35
Total Questions: 2

[redacted]

Roll No [redacted] Section

[redacted]
Student Signature

Do not write below this line

Attempt all the questions.

Follow the order of the questions as given. You must show all relevant work, reasoning, and steps as specified in the question statements to receive full credit.

CLO 3: Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc.

Q1: *[Marks: 20]*

A CPU scheduling system needs to handle six processes, where some processes are scheduled based on **preemptive Priority Scheduling** (where 1 is the highest priority) and others using **Shortest Remaining Time First (SRTF)**. The system always gives preference to **Priority Scheduling** for high-priority tasks, but if two processes have the same priority, the **SRTF** rule applies.

Process	Arrival Time	Burst Time	Priority
P1	0	5	2
P2	1	3	1
P3	2	8	4
P4	3	6	3
P5	4	2	1
P6	6	4	2

TAT = $\frac{66}{6}$ = 11

W = $\frac{38}{6}$ = 6.3333

Requirements:

1. **Draw a Gantt chart** showing the order of execution of these processes according to the scheduling rules.
2. Calculate the following:

- Average **Turnaround Time** for all processes.
- Average **Waiting Time** for all processes.

CLO 3: Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc.

Q2: [Marks: 10+5=15]

A)

In a busy restaurant, serving an order requires both a chef and a waiter. An order can only be completed if both roles are present.
There are two processes: one for chefs and one for waiters. The kitchen has only one serving station where orders are picked up. An order can be fulfilled if both a chef and a waiter are available. If a waiter arrives at the station and a chef has already prepared the order, the waiter picks it up and serves it, and vice versa. However, if a waiter arrives and no chef is available, they'll wait at the serving station until a chef arrives with the prepared order, and the same goes if a chef arrives first.
Additionally, no two chefs or two waiters can be at the serving station simultaneously; each order requires exactly one chef and one waiter working together. Implement these rules using **semaphores**.

Note: To receive full credit, explicitly declare and initialize all semaphores and variables.

Process 1 (Chef)	Process 2 (waiters)
Serving station (Critical Section)	Serving station (Critical Section)

B)

```
#define BUFFER_SIZE 10
typedef struct
{
  ...
} item;
item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

The code for the producer process is:

```
while(1)
{
   /*Produce an item in nextProduced*/
   while(counter == BUFFER_SIZE); /*do nothing*/
   buffer[in]=nextProduced;
   in=(in+1)%BUFFER_SIZE;
   counter++;
}
```

The code for the consumer process is:

```
while(1)
{
    while(counter==0); //do nothing
    nextConsumed=buffer[out];
    out=(out+1)%BUFFER_SIZE;
    counter--;
    /*Consume the item in nextConsumed*/
}
```

The code above addresses the famous Producer-Consumer problem and is intended to synchronize the producer and consumer processes:

- Identify a potential problem with this code. Explain the cause of this issue and the specific conditions under which it might occur.

National University of Computer and Emerging Sciences
Lahore Campus

Operating Systems (CS2006)

Date: April 4st 2024

Course Instructor(s)

Mr. Razi Uddin
Mr. Mubashar Hussain
Ms. Rubab Anam
Ms. Namra Absar

Sessional-II Exam

Total Time (Hrs):	1
Total Marks:	38
Total Questions:	3

Roll No _____ Section _____ Student Signature _____

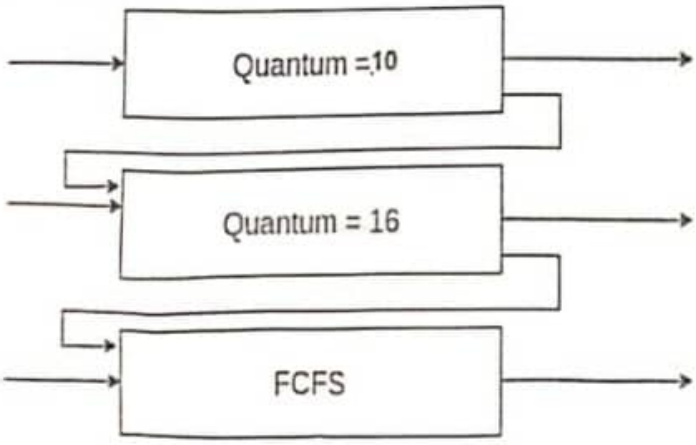
Do not write below this line

Attempt all the questions.

Q1: [CLO-3] **[Marks: 15]**

Consider a multilevel feedback queue scheduling with three queues, numbered as Q1,Q2,Q3.

- The scheduler first executes processes in Q1, which is given a time quantum of 10 milli-seconds.
- If a process does not finish within this time, it is moved to the tail of the Q2.
- The scheduler executes processes in Q2 only when Q1 is empty.
- The process at the head of the Q2 is given a quantum of 16 milli-seconds.
- If it does not complete, it is preempted and is put into Q3.
- Processes in Q3 are run on an FCFS basis, only when Q1 and Q2 are empty.
- A process that arrives for queue 2 will preempt a process in queue 3. A process in queue 2 will in turn be preempted by a process arriving for queue 1.
- If a process does not use up its quantum in queue 2 due to preemption by queue 1, it will keep its current queuing level and be put into the end of the queue. Then, it can still get the same amount of quantum (not remaining quantum) next time when it is picked.



The following set of processes, with the arrival times and the length of the CPU-burst times given in milliseconds, have to be scheduled using this Multilevel Feedback Queue Scheduler:

Processes	Arrival time	Burst time
P1	0	17
P2	12	25
P3	28	8
P4	36	32
P5	46	18

a) Draw a Gantt chart illustrating the execution of these processes.
b) Calculate the average waiting time and the average turnaround time for the scheduling.

Q2: [CLO-2] **[Marks: 8]**

A process has 3 threads, T1, T2, and T3, and its code does not contain any exec*() commands. T1 opens 3 files and T2 creates a pipe. After these actions, T1 executes a fork() command and T2 executes a fork() whose child immediately calls execvp() to execute a single-threaded program. T1 closes the three files and then terminates. Note that all processes use Pthreads.

a. How many different processes are running? Why? **[Marks: 2]**
b. How many different programs are being executed? Why? **[Marks: 2]**
c. How many open file descriptors are there? Why? **[Marks: 3]**
d. Thread T3 makes a system call that causes all threads in its process to terminate. Which system call could it be? **[Marks: 1]**

Q3: [CLO-3] **[Marks: 10+5=15]**

A) Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below. Synchronize them using Semaphores so that it always lead to an output string with **"000110001100011"**.

Process P	Process Q

while(1)
{
 	
cout<<"0";
 	
}
	while(1)
	{
	
	cout<<"1";
	
	}

B) Consider multi-threaded system in which two threads running this fragment of code simultaneously, s1, s2, s3 and s4 are shared semaphores, all are initiliaze to 1. While all other variables are automatic, that is each thread has a local copy of a and b that it modifies, initial values are a=2 and b=0 in Thread1 and a=0 and b=2 in Thread2.

P (or **wait**) is used to acquire a resource.
V (or **signal**) is used to release a resource.

Consider the following program fragment:

```
if(a > 0)
        P(s1);
else
        P(s2);
b++;
P(s3);
if(b < 0 && a <= 0)
        P(s1);
else if(b >= 0 && a > 0)
        P(s2);
else
        P(s4);
a++;
V(s4);
V(s3);
V(s2);
V(s1);
```

1. After running this fragment of code simultaneously, can there be a deadlock? Why, or why not?
2. In above case, state all the possible values of a, b and shared semaphores to justify part 1, for both threads

BEST OF LUCK!

Solution

National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	Operating Systems	Course Code:	CS 2006
Degree Program:	BS (CS/ SE/ DS)	Semester:	Fall 2023
Exam Duration:	60 Minutes	Total Marks:	30
Paper Date:	11-Nov-2023	Weight	15%
Section:	ALL	Page(s):	5
Exam Type:	Mid-2		

Student : Name:_____ Roll No._____ Section:_____

Instruction/Notes: Avoid unnecessary explanation. Kindly write your information in the above mentioned space. Read ALL questions **carefully** and answer accordingly.

Question 1: What will be the output of the given code? **(5 Marks)**

```
#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void *printme(void *ip)
{
  int *i;
  i = (int *) ip;
  printf("Hi.  I'm thread %d\n", *i);
  pid_t pid = fork();
  printf("Hello with  %d\n", *i);
  return NULL;
}

int main()
{
  int i, vals[4];
  pthread_t tids[4];
  void *retval;

  for (i = 0; i < 4; i++) {
    vals[i] = i;
    pthread_create(tids+i, NULL, printme, vals+i);
    pthread_join(tids[i], &retval);
    printf("Joined with tid %d\n", i);
  }
return 0;
}
```

Answer:

Hi. Im thread 0
Hello with 0
Hello with 0
joined with tid 0

Hi. Im thread 1
Hello with 1
Hello with 1
joined with tid 1

Hi. Im thread 2
Hello with 2
Hello with 2
joined with tid 2

Hi. Im thread 3
Hello with 3
Hello with 3
joined with tid 3

0295

min = 2
Avg = 5
max = 10

Question 2: **(10 Marks)**

Late-Night Cake. A group of students are studying for an exam. The students can study only while eating cake. Each student executes the following loop:

```
while (true) {
        pick up a piece of cake;
        study while eating the cake;
}
```

If a student finds that the cake is gone, the student goes to sleep until another cake arrives. The first student to discover that the group is out of cake phones Layers at Johar town to order another cake before going to sleep. Each cake has **C** slices. Write code to synchronize the student threads and the cake delivery thread. Your solution should avoid deadlock and phone Layers (i.e., wake up the delivery thread) exactly once each time a cake is exhausted. No piece of cake may be consumed by more than one student.

10 Also write the initial values of any variable and semaphore used in the code.

Hint: You need to write only two functions. One is for students and other is for Layers

3 binary semaphore

```
mutex = 1
order-pizza = 0
deliver-pizza = 0
bool first = True, have-pizza = F
```

```
cheezious()
{
do
{
wait(mutex);
wait(order-pizza);
makePizza();
slices = S;
first = true;
signal(deliver-pizza);
signal(mutex);
} while(1);
}
```

```
int slices = S;
students()
{ do
{
wait(mutex);
while(!have-pizza)
{
if(slices > 0)
{
slices--;
have-pizza = True;
}
else
{
if(First)
{ signal(order-pizza);
first = false;
}
wait(deliver-pizza);
}
signal(mutex);
study(); //sleep()
have-pizza = false;
```

C.S

C.S

S1

S2

context switch

2

0295

Question 3: **(10 Marks)**

Assume that there are 5 processes, P0 through P4, and 4 types of resources. At T0 we have the following system state:

Total resources (A=3 , B=17 , C=16 , D=12)

Process	Max	Allocation
	A, B, C, D	A, B, C, D
P0	0 2 1 0	0 1 1 0
P1	1 6 5 2	1 2 3 1
P2	2 3 6 6	1 3 6 5
P3	0 6 5 2	0 6 3 2
P4	0 6 5 6	0 0 1 4

Suppose the system is in a safe state, can the following requests be granted by using Banker's Algorithm, why or why not? Calculate any matrix or vector if required. Please also run the safety algorithm on each request if necessary.

(If any of the request is granted the next request will be checked on the **new state of the system**)

a. P1 requests (2,1,1,0)

b. P1 requests (0,2,1,0)

c. P3 requests (1,0,2,0)

(a) P1 request (2,1,1,0)

⇒ Is request1 ≤ Need1 ??

(2,1,1,0) ≤ (0,4,2,1) false request denied

(b) P1 request (0,2,1,0)

⇒ Is request1 ≤ Need1 ??

(0,2,1,0) ≤ (0,4,2,1) True

⇒ Is request1 ≤ Available ??

(0,2,1,0) ≤ (1,5,2,0) True.

Now apply safety Algorithm to find safe sequence.

Processes	Max A	B	C	D	Allocation A	B	C	D	Need A	B	C	D	Available A	B	C	D
P0	0	2	1	0	0	1	1	0	0	1	0	0	1	5	2	0
P1	1	6	5	2	1	2	3	1	0	4	2	1	1	6	3	0
P2	2	3	6	6	1	3	6	5	1	0	0	1	1	12	6	2
P3	0	6	5	2	0	6	3	2	0	8	2	0	1	12	7	6
P4	0	6	5	6	0	0	1	4	0	6	4	2	2	14	10	7

(2, 1, 1, 0)

P0, P3, P4, P1, P2

(P0, P1, P2, P3, P4)

Safe sequence can be many.

(c) P3 requests (1, 0, 2, 0)

=> is $request_3 \leq Need\,3$??

(1, 0, 2, 0) $\leq$ (0, 0, 2, 0) false

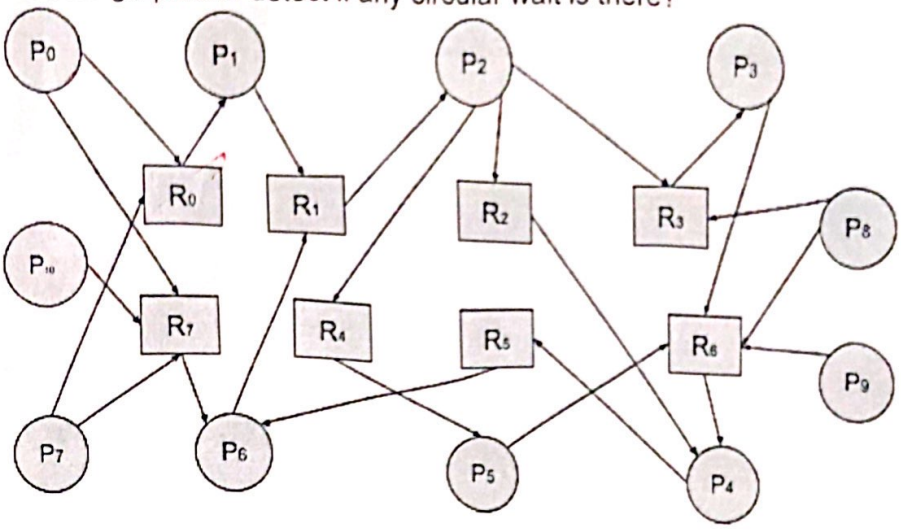
request denied.

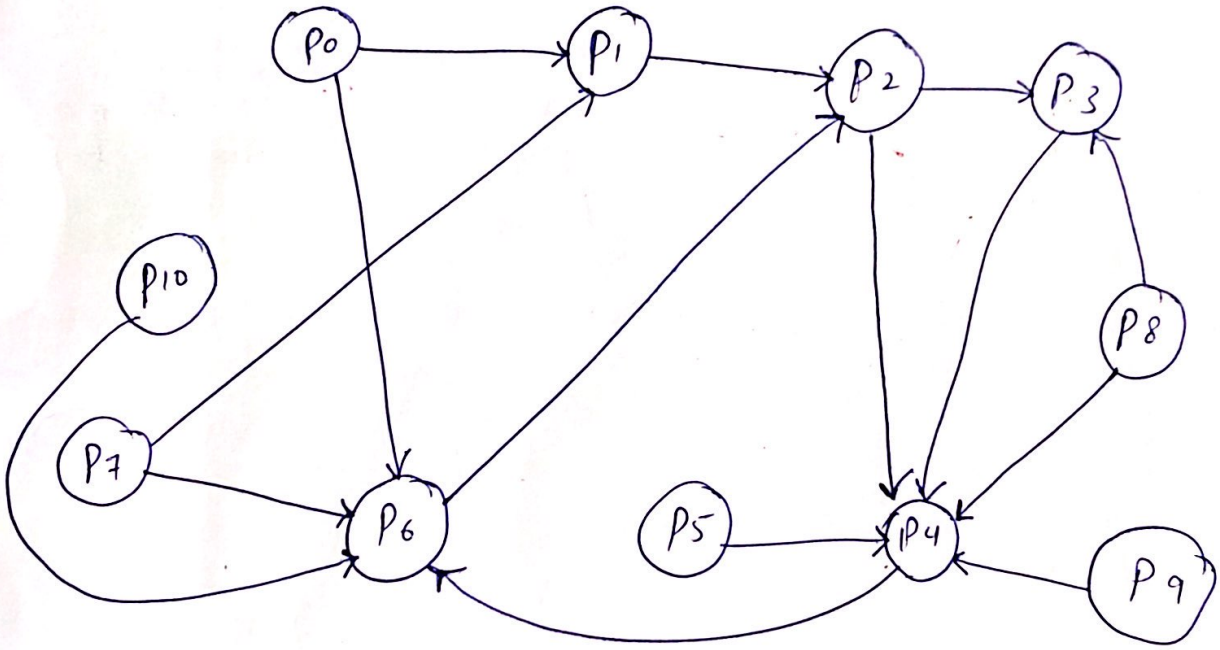
But there exist a safe sequence after running the Bankers algorithm.

(P0, P3, P4, P1, P2)

Question 4: **(5 Marks)**

Create a wait-for-graph and detect if any circular wait is there?





Yes circular wait is there.

National University of Computer and Emerging Sciences, Lahore Campus

Course:	Operating System	Course Code:	CS-205
Program:	BS(Computer Science)	Semester:	Fall 2017
Duration:	1 hour	Total Marks:	50
Paper Date:	2nd November, 2017	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): Given below are two tables. The first table shows the processes coming **in order** (a process with lower number comes earlier than the one with greater number). Using **best fit** algorithm assign each process some hole given in the following table. Also calculate the remaining space.

Process#	Required Memory (MBs)
1	20
2	43
3	21
4	77
5	6
6	11
7	15
8	105
9	62
10	90

Hole#	Hole Size (MBs)	Allotted Process#	Remaining space
1	83		
2	50		
3	100		
4	25		
5	30		
6	10		
7	70		
8	15		
9	17		
10	110		

Question 2 (4 points): If you use **worst fit** algorithm instead, then which hole will be assigned to first **two** processes

- Process 1 → Hole#=_____
- Process 2 → Hole#=_____

Question 3 (6 points): Extract page numbers and offsets from the following logical addresses. The page size is 1000 bytes.

- 2101 → Page#=_____ Offset=_____
- 5215 → Page#=_____ Offset=_____
- 102 → Page#=_____ Offset=_____

Question 4 (15 points): Using the below given functions you have to implement a function `handlePageFault`. It is called when a page fault occurs. Meaning, when a page is not found loaded into the memory, it load it and fixes the page table. Use following functions without the knowledge of their definition. Comments describe their functionality. The page table is a **two level page table** stored without caching. **Hint:** read the declarations carefully!

```
int getFirstLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the associated first level page number.

int getSecondLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
     and returns the associated second level page number.

int getCombinedPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the value of bits associated to the page number (first and second level).

int* getFirstLevelPageTable(int PID); // takes the process ID as input and returns the pointer
     to its first level page table.

int loadPageFromBackingStore(int combinedPageNumber, int PID); // takes the page number and
    process ID as inputs and returns the frame number on which it loaded the page.
```

```
void handlePageFault( int logicalAddress, int PID)//the faulty address and the process ID are
    the parameters to the function.
{









}
```

Question 5 (15 points): Following is a proposed solution for readers writers problem. Although the solution fulfills some requirements, but it does violate "The Bounded Wait" property. Which means that in this case the writer may have to wait indefinitely. Remove that problem by inserting new semaphore(s). For 100% marks propose a solution which gives a better chance to writers to enter into the critical section.

Code for Writers	Code for Readers
roomEmpty=0,mutex=0, readers=0 // all variables are shared	
1: *roomEmpty.wait*() 2: Write(*Rsc*) 3: *roomEmpty.post*()	1: *mutex.wait*() 2: *readers*++ 3: **if** *readers* == 1 **:** 4: *roomEmpty.wait*() 5: *mutex.post*() 6: Read(*Rsc*) 7: *mutex.wait*() 8: *readers*−− 9: **if** *readers* == 0 **:** 10: *roomEmpty.post*() 11: *mutex.post*()

National University of Computer and Emerging Sciences, Lahore Campus

Course:	Operating System	Course Code:	CS-205
Program:	BS(Computer Science)	Semester:	Fall 2017
Duration:	1 hour	Total Marks:	50
Paper Date:	2nd November, 2017	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): Given below are two tables. The first table shows the processes coming **in order** (a process with lower number comes earlier than the one with greater number). Using **best fit** algorithm assign each process some hole given in the following table. Also calculate the remaining space.

Process#	Required Memory (MBs)
1	20
2	43
3	21
4	77
5	6
6	11
7	15
8	105
9	62
10	90

Hole#	Hole Size (MBs)	Allotted Process#	Remaining space
1	83	4,5	6,0
2	50	2	7
3	100	10	10
4	25	1	5
5	30	3	9
6	10		
7	70	9	8
8	15	6	4
9	17	7	2
10	110	8	5

Question 2 (4 points): If you use **worst fit** algorithm instead, then which hole will be assigned to first **two** processes

- Process 1 → Hole#= 10
- Process 2 → Hole#= 3

Question 3 (6 points): Extract page numbers and offsets from the following logical addresses. The page size is 1000 bytes.

- 2101 → Page#=2 Offset=101
- 5215 → Page#=5 Offset=215
- 102 → Page#=0 Offset=102

Question 4 (15 points): Using the below given functions you have to implement a function `handlePageFault`. It is called when a page fault occurs. Meaning, when a page is not found loaded into the memory, it load it and fixes the page table. Use following functions without the knowledge of their definition. Comments describe their functionality. The page table is a **two level page table** stored without caching. **Hint:** read the declarations carefully!

```
int getFirstLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the associated first level page number.

int getSecondLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
     and returns the associated second level page number.

int getCombinedPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the value of bits associated to the page number (first and second level).

int* getFirstLevelPageTable(int PID); // takes the process ID as input and returns the pointer
     to its first level page table.

int loadPageFromBackingStore(int combinedPageNumber, int PID); // takes the page number and
    process ID as inputs and returns the frame number on which it loaded the page.
```

```
void handlePageFault( int logicalAddress, int PID)//the faulty address and the process ID are
    the parameters to the function.
{

        int firstP = getFirstLevelPageNumber(logicalAddress);
        int secondP = getSecondLevelPageNumber(logicalAddress);
        int combinedP = getCombinedPageNumber(logicalAddress);

        int* firstPT = getFirstLevelPageTable(PID);
        int* secondPT = firstPT[firstP];

        int frame = loadPageFromBackingStore(combinedP, PID);
        secondPT[secondP] = frame;

}
```

Question 5 (15 points): Following is a proposed solution for readers writers problem. Although the solution fulfills some requirements, but it does violate "The Bounded Wait" property. Which means that in this case the writer may have to wait indefinitely. Remove that problem by inserting new semaphore(s). For 100% marks propose a solution which gives a better chance to writers to enter into the critical section.

Code for Writers	Code for Readers
roomEmpty=0,mutex=0, readers=0 // all variables are shared	
1: *roomEmpty.wait*() 2: Write(*Rsc*) 3: *roomEmpty.post*()	1: *mutex.wait*() 2: *readers*++ 3: **if** *readers* == 1 **:** 4: *roomEmpty.wait*() 5: *mutex.post*() 6: Read(*Rsc*) 7: *mutex.wait*() 8: *readers*−− 9: **if** *readers* == 0 **:** 10: *roomEmpty.post*() 11: *mutex.post*()

Code for Writers	Code for Readers
roomEmpty=0,mutex=0, readers=0 sem = 0// all variables are shared	
1: *sem.wait*() 2: *roomEmpty.wait*() 3: Write(*Rsc*) 4: *sem.post*() 5: *roomEmpty.post*()	1: *sem.wait*() 2: *sem.post*() 3: *mutex.wait*() 4: *readers*++ 5: **if** *readers* == 1 **:** 6: *roomEmpty.wait*() 7: *mutex.post*() 8: Read(*Rsc*) 9: *mutex.wait*() 10: *readers*−− 11: **if** *readers* == 0 **:** 12: *roomEmpty.post*() 13: *mutex.post*()

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 parition.
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.
{









}
```

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.

sem_1=1,sem_2=0, buffer // among shared variables **buffer** is an infinite list of elements , rest are semaphores

Producer

```
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 }
```

Consumer

```
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 }
```

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 _____

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. Use the correct C++ syntax.

```
1  int main ()
2  {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28 }
```

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

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

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

1. Creation of file is slow
2. Enlarging the file size is slow
3. Deleting a file 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 _____ 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
2. Many to one
3. One to one

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 parition.
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

```
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 }
```

```
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 }
```

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 << "smepahore creation error";
19                  exit(-1);
20          }
21
22          if (sem_init(sem_2, 0, 0) == -1)
23          {
24                  cout << "smepahore 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
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 (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 8 (2 points): Among the following, which function on files is counter productive in sequential storage mediums like tape drives

1. open
2. close
3. **seek**
4. create

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

1. Creation of file is slow
2. **Enlarging the file size is slow**
3. Deleting a file 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**
2. **Many to one**
3. One to one

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-220
	Program:	BS(Computer Science)	Semester:	Fall 2020
	Duration:	1.5 hour	Total Marks:	45
	Paper Date:	25^{th} November, 2020	Weight:	15%
	Section:	ALL	Page(s):	4
	Exam:	Mid-2		

Instructions/Notes: Answer questions on the question paper. Attempt all questions. Programmable calculators are not allowed. 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**.

Name:_____ **Roll No.:**_____ **Section:**_____

Question 1 (2 points): The major concern with race conditions is (tick the correct option)

(1) Inefficient use of resources
(2) Integrity of data
(3) Loss of data
(4) No optimal solutions available.

(1) Inefficient use of resources
(2) **Integrity of data**
(3) Loss of data
(4) No optimal solutions available.

Question 2 (2 points): Which of the following is NOT the name of a Linux file system structure (tick the correct option)

(1) Dentry
(2) Master File table
(3) Inode
(4) Superblock

(1) Dentry
(2) **Master File table**
(3) Inode
(4) Superblock

Question 3 (2 points): The following is NOT an advantage of Semaphore (tick the correct option)

(1) Resource management
(2) Sequencing execution
(3) Ensuring mutual exclusion
(4) Avoids deadlocks

(1) Resource management
(2) Sequencing execution
(3) Ensuring mutual exclusion
(4) **Avoids deadlocks**

Question 4 (2 points): Linked allocation method in file system suffers from (tick the correct option)

(1) Performance and reliability issue
(2) External fragmentation and performance issue
(3) Internal fragmentation and external fragmentation
(4) External fragmentation and reliability issue.

(1) **Performance and reliability issue**
(2) External fragmentation and performance issue
(3) Internal fragmentation and external fragmentation
(4) External fragmentation and reliability issue.

Question 5 (2 points): Contiguous allocation method in file system suffers from (tick the correct option)

(1) Performance of reading the file
(2) Performance of saving the file
(3) Performance of expanding the file
(4) Data integrity

(1) Performance of reading a file
(2) Performance of saving a file
(3) **Performance of expanding a file**
(4) Data integrity

Question 6 (10 points): Assume that you are in a google meet question-answer session with your teacher. During this session, only one person can speak at a time, either the teacher or a student. Since there are many students, any one of the students can ask a question. The teacher accepts only one question at a time and speaks only when a question is asked. However, as soon as the question is completed, the teacher has to answer it. No other student can ask a question until the first question is answered. Remember, it is the students who will start the session by asking a question from the teacher. You are required to force this sequence of question-answer session using semaphore(s). In order to write your solution, you may use the following semaphores:

1. Semaphore *mutex_speak*: both the teacher and students will use this semaphore to speak; either to answer or ask the question respectively
2. Semaphore *question*: One of the students will use this semaphore to ask a question.
3. Semaphore *answer*: A teacher will enable himself to answer a question using this semaphore.

Your answer should contain the prototype of the code using the following syntax: "wait(speak); signal (avail_question)" etc. Furthermore, you are also required to initialize the semaphores.

Question 7 (5 points): The following numbered statements show the sequence of opening and reading a newly created file on a file system. Your task is to order the statements in the correct sequence.

1. The open() system calls searches the system-wide open-file table for the file if it is already in use.
2. The application program calls the logical file system to open a file.
3. The physical block numbers for the requested logical block numbers are to calculated.
4. The directory structure provides the pointer to the inode
5. The required physical blocks are loaded into memory.

Solution: 2, 1, 4, 3, 5.

Question 8 (10 points): Considering a file having a size of 20 KB. Using index allocation method, a block size of 256 bytes, and a pointer of 4 bytes, answer the following questions

1. How many data blocks will be required for storing this file in the secondary storage?
2. How many second level index blocks will be required to complete the mapping of this file?
3. If you have to read the $8,220^{th}$ byte, which second level index block will you use?
4. If you have to read the $8,220^{th}$ byte, which index of second level index block will you use?
5. What offset in the data block will be used to read the $8,220^{th}$ byte?

Solution:

1. $20480/256 = 80$ data blocks
2. $80/64 = 1.25$, meaning 2 index block.

3. $8220/(256*64) = 0.5$, meaning 0^{th} index block
4. First $8220 \quad mod \quad (256*64) = 8220$. Then, $8220/256 = 32.1$, meaning 32nd index.
5. $8220 mod 256 = 28$.

Question 9 (10 points): You are given two vectors a and b of equal length. You need to sum the two vectors element wise and save the result into another vector c. Note that at the end of the operation the length of a, b and c will be same. Besides the main thread, you can create only two additional threads. Write the code to do the task in a most efficient way. You do not need to initialize vectors a, b and c. Assume that their values are already populated, as shown in the code below.

```
// Better solutions are possible, but following would suffice
#define LENGTH xxxx

struct arr_limits{
        int start;
        int end;
};

int* a= NULL;
int* b= NULL;
int* c= NULL;

#define handle_error(msg) do { perror(msg); exit(EXIT_FAILURE); } while (0)

void* sum_array(*void params)
{
        arr_limits* limits = (arr_limits*) params;
        for (int i = arr_limits->start; i < arr_limits->end; ++i)
                c[i] = a[i] + b[i]
        pthread_exit(NULL);
}
int main()
{
        a = populate_a();
        b = populate_b();
        c = populate_c();

        arr_limits limit_1 ;
        limits_1.start = 0;
        limits_1.end = LENGTH/2;

        arr_limits limit_2 ;
        limits_2.start = LENGTH/2 + 1;
        limits_2.end = LENGTH;


        int s;
        pthread_attr_t attr;
        s = pthread_attr_init(&attr);
        if (s != 0)
           handle_error_en(s, "pthread_attr_init");
        pthread_t tid1 = 0;
        pthread_t tid2 = 0;

        s = pthread_create(&tid1, &attr, sum_array, &limits_1);
        if (s != 0)
           handle_error_en(s, "pthread_create");

        s = pthread_create(&tid2, &attr, sum_array, &limits_2);
        if (s != 0)
           handle_error_en(s, "pthread_create");

        s = pthread_join(tid1, NULL);
        if (s != 0)
           handle_error_en(s, "pthread_join");

        s = pthread_join(tid2, NULL);
        if (s != 0)
           handle_error_en(s, "pthread_join");

}
```