National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Operating Systems	Course Code:	CS2006
	Degree Program:	BS (CS / SE / DS)	Semester:	Spring 2023
	Exam Duration:	180 Minutes	Total Marks:	90
	Paper Date:	08-June-2023	Weight	45%
	Section:	ALL Sections	Page(s):	11
	Exam Type:	Final Exam		

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

Instruction/Notes: Avoid unnecessarily explanation. Kindly write your information in the above mentioned space. Rough sheet is also attached on the last page. No extra sheet is allowed.

CLOs	CLO-3	CLO-3	CLO-5	CLO-2	CLO-5	CLO-5	
Questions	Q-1	Q-2	Q-3	Q-4	Q-5	Q-6	Total
Total Marks	10	20	15	10	15	20	**90**
Marks Obtained							

Question 01: (10 points) (CLO-3)

Javed, Yusuf, and Babar go to a restaurant at a busy time of the day. The waiter apologetically explains that the restaurant can provide only two pairs of spoons (for a total of four spoons) to be shared among the three people. Yusuf proposes that all four spoons be placed in an empty glass at the center of the table and that each diner should obey the following protocol:

```
semaphore spoon = 4;

while (!had_enough_to_eat())
{
        wait(spoon);
        wait(spoon);
        eat();
        signal(spoon);
        signal(spoon);
}
```

(a) Can this dining plan lead to a deadlock? YES or NO. Explain your answer.

Answer:

(b) Suppose now that instead of three there will be an arbitrary number of *D* diners. Furthermore, each diner *d* = 1...*D* may require a different number of S_d spoons to eat. For example, it is possible that one of the diners is an octopus, who for some reason refuses to begin eating before acquiring *octopus* = 8 spoons. For example, Javed, Yusuf, Babar, and one octopus would result in C = 14. Each diner's eating protocol will be as displayed below:

```
int s;
int num_spoons = my_spoon_requirement();
while (!had_enough_to_eat())
{
        for (s = 0; s < num_spoons; s++)
        {
                wait (spoon); /* May block. */
        }
        eat ( );
        for (s = 0; s < num_spoons; s++)
        {
                signal (spoon); /* Does not block. */
        }
}
```

What is the smallest number of spoons (in terms of D and S_d) needed to ensure that deadlock cannot occur? Explain your answer.

Answer:

Question 02: (20 points) (CLO-3)

Consider a Multilevel Feedback Queue Scheduler having three queues numbered from 1 to 3 (see Fig. A). The processes are scheduled as follows:

- A new process enters queue 1 which is served using Round Robin (RR). When it gains CPU, process receives 8 milliseconds. If it does not finish in 8 milliseconds, process is moved to the end of queue 2.
- If queue 1 is empty, the processes at queue 2 are served using RR and receives 16 milliseconds. If it does not complete, it is preempted and moved to queue 3.
- Processes in queue 3 are run on a First Come First Serve (FCFS) basis, but are run only when queues 1 and 2 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:

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

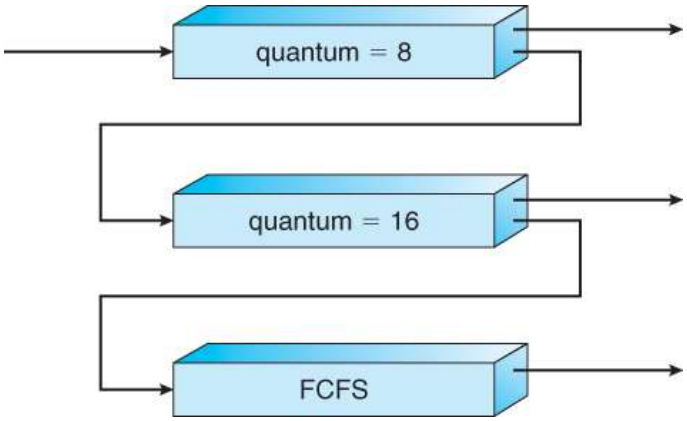


Figure A: Multilevel feedback queue

(a) Draw a Gantt chart illustrating the execution of these processes.

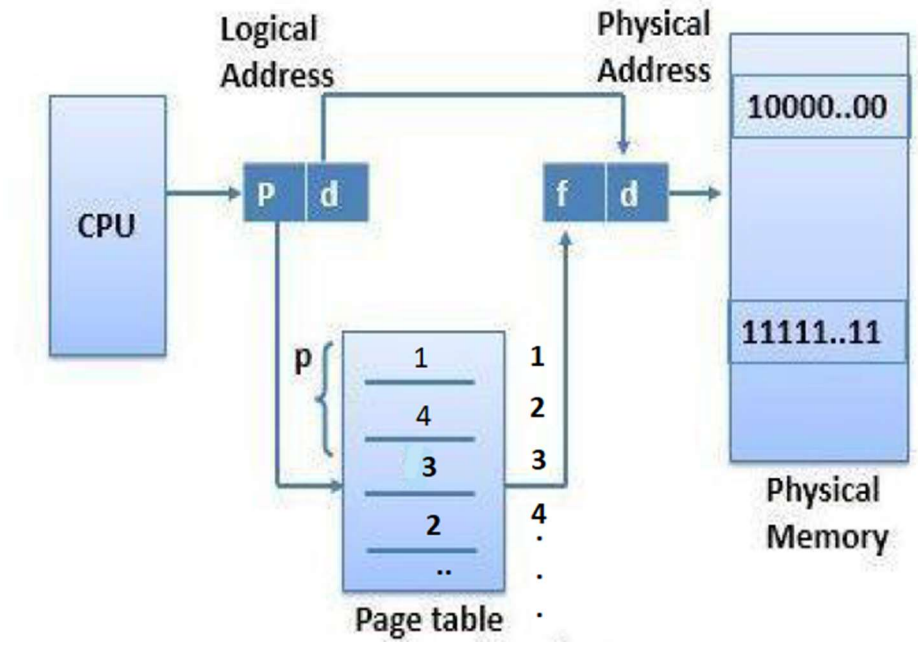
(b) Calculate the number of context switches for the processes.

(c) Calculate the average waiting time and the average turnaround time for the scheduling.

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

Consider a system with **20** bits logical address and **4 MBs** main memory size and page size is **4 KBs**.

1200 is logical address



A. Convert 1200 into its binary representation

B. No. of bits needed for p (page number)

C. No. of bits needed for f (frame number)

D. No. of bits needed for d (offset)

E. Convert logical address into physical address (you have to provide its binary as well as decimal representation).

Question 04: (10 points) (CLO-2)
Write the output of the following code.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main( ) {
   int n = 5;
   pid_t pid;

   for (int i = 0; i < n; i++) {
      pid = fork( );

      if (pid == 0) {
         if (i % 2 == 0) {
            fork( );
            printf ("Child process %d\n", i);
         } else {
            printf ("Child process %d\n", i);
            fork( );
         }
         break;
      } else if (pid > 0) {
         wait (NULL);
         printf ("Parent process\n");
      } else {
         printf ("Fork failed\n");
         return 1;
      }
   }

   return 0;
}
```

Answer:

Question 05: (15 points) (CLO-5)
A process has four page frames allocated to it. (All the following numbers are decimal, and everything is numbered starting from zero). The time of the last loading of a page into each page frame, the time of last access to the page in each page frame, the virtual page number in each page frame, and the referenced (R) and modified (M) bits for each page frame are as shown (the times are in clock ticks from the process start at time zero to the event -- not the number of ticks since the event to the present).

Virtual page number	Page frame	Time loaded	Time referenced	R bit	M bit
2	0	60	164	1	1
1	1	30	166	1	0
0	2	150	162	0	1
3	3	20	163	1	1

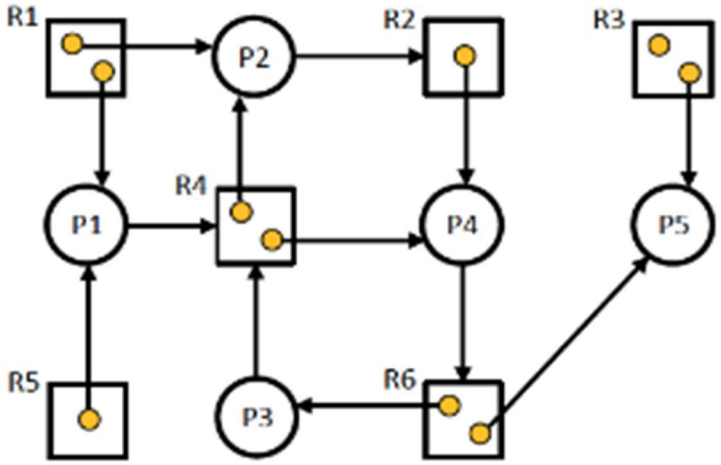
A page fault to virtual page 4 has occurred. Which page frame will have its contents replaced for each of the following memory management policies? Explain why in each case.

a) FIFO (first-in-first-out)

b) LRU (least recently used)

c) Optimal page replacement algorithm

Question 06: (20 points) (CLO-5)
Consider the following Resource Allocation Graph (RAG):



Do the following problems:

(a) Convert it to the matrix representation (i.e., Allocation, Request and Available)

(b) Is there a deadlock? If there is a deadlock, which processes are involved? Otherwise, provide safe sequence.

(Rough Sheet)

0261

National University of Computer and Emerging Sciences

Lahore Campus

Operating Systems (CS2006)

Date: DEC 23th 2024

Course Instructor(s)

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

Final Exam

Total Time (Hrs):	**3**
Total Marks:	**95**
Total Questions:	**8**

22i-2505 Roll No

SE-5A Section

Student Signature

Do not write below this line

Attempt all the questions in sequence.

[CLO-1] Describe services provided by the modern Operating Systems

Q1: **[Marks:5]**

How operating system use hardware mechanisms like the Memory Management Unit (MMU) to convert logical addresses to physical addresses, ensuring memory protection and efficient resource management. Give an example.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q2: **[Marks:5]**

Consider the following C code:

```
int main() {
    int x = 3, y = 2;

    if ((x + y) % 2 == 1) {     // Condition 1
        fork();                 // Fork A
    }
    for (int i = 0; i < x; i++) { // Loop 1
        if (fork() == 0) {      // Fork B
            break;
        }
    }
    if (x * y > 5 && fork()) { // Condition 2 and Fork C
```

```
        fork();                    // Fork D
    }
    return 0;
}
```

Draw the complete fork tree generated by this program. Label each process with its parent-child relationship (e.g., P, C1, C2, etc.). Clearly indicate how the loop and conditions affect process creation.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q5: Warehouse Stock Management with Threads and Semaphores [15 Marks]

Requirements:

1. **Warehouse Initialization:**

- Define an array of NUM_SECTIONS warehouse sections. Each section will have an initial stock of INITIAL_STOCK items (e.g., 100 items per section).
- Use an array of semaphores, where each semaphore corresponds to a warehouse section, to ensure thread-safe access to the stock.
- Skeleton code is given below.

```
#define NUM_SECTIONS 5
#define INITIAL_STOCK 100

int warehouse_stock[NUM_SECTIONS];   // Array to store stock in each section
sem_t section_lock[NUM_SECTIONS];    // Array of semaphores for synchronization

void initializeWarehouse()           // Initialize each semaphore
{
//Write your code here
}
```

2. **Transaction Functions**

- Implement two functions: **addStock()** and **removeStock()**.
- These functions will:
  1. Accept a section number and transaction amount as arguments.
  2. Use semaphores to ensure only one thread modifies a section's stock at a time.
  3. Ensure **removeStock()** checks for sufficient stock before decrementing.

```
void addStock(int section, int amount) {
//Write your code here
}
void removeStock(int section, int amount) {
//Write your code here
}
```

3. **Worker Threads**

- Create multiple threads (e.g., **NUM_WORKERS = 5**), where each thread represents a worker.
- Each thread will:

1. Randomly select a warehouse section.
2. Randomly choose to add or remove stock (50% probability for each).
3. Perform the operation using **addStock()** or **removeStock()** and print the transaction details.

```
#define NUM_WORKERS 5

void* worker(void* arg) {
//Write your code here
}
```

4. Synchronization

- Use semaphores to ensure that only one thread can access a warehouse section's stock at a time.
- Semaphores ensure there is no race condition or inconsistent state.

Basic Code Structure for Main Function:

```
int main()
{

pthread_t workers[NUM_WORKERS];

int worker_ids[NUM_WORKERS];

// Initialize warehouse

// Create worker threads

// Print final stock

}
```

Sample Output:

```
Added 7 to Section 2. Current Stock: 107
Removed 4 from Section 3. Current Stock: 96
Removed 3 from Section 0. Current Stock: 97
Added 8 to Section 4. Current Stock: 108
...

Final Stock in Warehouse Sections:
Section 0: 102 items
Section 1: 100 items
Section 2: 105 items
Section 3: 98 items
Section 4: 110 items
```

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

Q4: **[Marks: 20]**

A system implements a Multi-Level Feedback Queue (MLFQ) scheduler with the following characteristics:

- **Queue 1** (Highest Priority): Time quantum = 3 ms, Round Robin.
- **Queue 2** (Medium Priority): Time quantum = 6 ms, Round Robin.
- **Queue 3** (Lowest Priority): FCFS (First-Come, First-Served).

At time **t = 0 ms**, five processes arrive in the system with the following burst times and priority boost intervals:

Process	Burst Time (ms)	Arrival Time (ms)	Priority Boost Interval (ms)
P1	10	0	20
P2	15	0	30
P3	8	0	16
P4	20	0	45
P5	17	0	24

Rules:

1. All processes start in **Queue 1**.
2. If a process does not finish within its time quantum, it moves to the next lower-priority queue.
3. After every **Priority Boost Interval** (calculated from the arrival time of the process), the process is **immediately moved and placed at the end of Queue 1** (if the process is not using CPU). **[e.g. for P1, 1st booster at 20 , 2nd at 40 , 3rd at 60 and so on. Same goes for others]**
4. If a process is currently executing (using CPU) in **Queue 2** or **Queue 3** when its priority boost interval is reached:
    - The process will **not surrender its current time slice** in Queue 2 but will be boosted to Queue 1 right after its time slice finishes.
    - The process will be **interrupted mid-execution in Queue 3 (FCFS)** and moved to Queue 1 at the end of the queue 1.
5. Processes in the same queue are scheduled based on the respective queue's scheduling policy. Context switching time is ignored.

Tasks:

1. Draw the Gantt chart for the schedule.
2. Calculate the turnaround time (TAT) and waiting time (WT) for each process and average.

National University of Computer and Emerging Sciences
Lahore Campus

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

OR

Scenario **[15 Marks]**

In a factory, there is a shared machine that is used for producing goods. The machine can either be operated by workers to produce the goods or undergo maintenance by engineers.

- **Production Workers:** These workers operate the machine to produce goods. They only use the machine to create products, without modifying its internal structure.
- **Engineers:** These workers perform maintenance on the machine, which includes checking its components, fixing issues, or upgrading its software or mechanical parts. During maintenance, the machine cannot be used for production.

Here are the rules for the factory system:

1. **Multiple Production Workers Rule:** If there are several workers using the machine at the same time, they can all work without issue, as long as no engineers are working on the machine.

2. **Maintenance Rule:** If an engineer is performing maintenance on the machine, no one else can use it. This means that no workers (production or maintenance i.e. other engineers) are allowed to access the machine until the engineer finishes their work.

3. **Priority for Engineers:** If an engineer is waiting to perform maintenance on the machine, they must be given priority. If there are any workers currently using the machine, they must finish their work, and no new workers can start operating the machine until the engineer begins the maintenance.

4. **No New Workers When an Engineer is Waiting:** If an engineer is waiting for the machine, no new workers are allowed to start using the machine, even if some workers are already working on it. Once all current workers finish, the engineer can perform maintenance.

Task:
Write a solution for the system using synchronization mechanism (**semaphores**) to manage access to the shared machine ensuring that engineers are given priority and that production workers can continue using the machine if no engineers are waiting. **To receive full credit, you are required to explicitly initialize all semaphores and variables in your solution.**

Production Workers	Engineers

Worker() {
 	
<CRITICAL Section> //Production started
 	
}
	Engineer() {
	
	<CRITICAL Section> // Maintenance started
	}

[CLO-4] Deploy OS tools related related to Virtualization and Containers

Q6: **[7+2+2+4=15 Marks]**

Part A:

Consider a single-level paging scheme where:

- The virtual address space is 512MB.
- The page table entry size is 8 bytes.

What is the minimum page size such that the entire page table fits in one page? For the page table to fit within a single page, the following condition must be satisfied:

Page Table Size ≤ Page Size

Part B:

Consider a memory system with the following properties:

- The address length is 30 bits.
- The memory is 2-byte addressable (each address refers to 2 bytes of memory).

What will be the size of the Memory?

Part C:

A computer system has 40-bit virtual address space with a page size of 16K, and 8 bytes per page table entry.

1. How many pages are in the virtual address space?
2. What is the maximum size of the addressable memory in this system? **(Hint:** you can ignore protection bits)

[CLO-4] Deploy OS tools related related to Virtualization and Containers

Q7: **[10 Marks]**

You are given a following **page reference string** for a single process:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 4, 5, 3, 6, 7, 5

Each page in the reference string has a **criticality factor**, which determines its **cost** during a page fault. The **criticality factor** of a page can be one of the following:

- **Critical pages (C)**: Have a **cost of 5 units** per page fault.
- **Non-critical pages (N)**: Have a **cost of 2 units** per page fault.

Page	Criticality Factor
7	C
0	N
1	N
2	C
3	N
4	C
5	N
6	C

Memory starts with 4 empty frames, Your task is to apply **FIFO** and **LRU** Page replacement Algorithms and calculate the total number of page faults and the total cost for each algorithm.

[CLO-5] Understand the dead locks and memory management.

Q8: **[2+3+3+2=10 Marks]**

A resource allocation system that uses the Banker's algorithm for 4 resource types (A, B, C, D) and 5 users (P0, P1, P2, P3 P4) is currently in the following state.

Processes	Allocation (A B C D)	Max (A B C D)	Available (A B C D)
P_0	2001	4212	3321
P_1	3121	5252	
P_2	2103	3216	
P_3	1312	4244	
P_4	1432	3665	

Using Banker's algorithm, answer the following questions:

a. How many resources of type A, B, C and D are there in the system?
b. What are the contents of the need matrix?
c. Is the system in a safe state? Why?
d. If the request from P4 arrives for (0, 0, 2, 0), can the request be granted immediately?

National University of Computer and Emerging Sciences 0046

Islamabad Campus

Operating Systems (CS2006)

Final Exam

Date: May 22, 2024

Total Time (Hrs): 3
Total Marks: 100
Total Questions: 7

Course Instructor(s):
Dr. M. Faisal Cheema, Dr. Adnan Tariq, Ms. Maryam Shahbaz, Ms. Rabail Zahid, Mr. M. Aadil-ur-Rehman

Roll No **Course Section** **Student Signature**

IMPORTANT: Attempt Question 1, 2 on Answer Sheet, and remaining Question on Question Paper. If you do not follow the instructions, your questions will not be marked.

Attempt all the questions.

Question 1: (12 Marks)
[Attempt this Question on the Answer Sheet]

Danyal, Sohaib, and Fahad plant seeds continuously. Danyal digs the holes. Sohaib then places a seed in each hole. Fahad then fills the hole up. There are several synchronization constraints:

1. Sohaib cannot plant a seed unless at least one empty hole exists, but Sohaib does not care how far Danyal gets ahead of Sohaib.
2. Fahad cannot fill a hole unless at least one hole exists in which Sohaib has planted a seed, but the hole has not yet been filled. Fahad does not care how far Sohaib gets ahead of Fahad.
3. Fahad does care that Danyal does not get more than MAX holes ahead of Fahad. Thus, if there are MAX unfilled holes, Danyal has to wait.
4. There is only one shovel with which both Danyal and Fahad need to dig and fill the holes, respectively.

Write the pseudocode for the 3 processes which represent Danyal, Sohaib and Fahad using semaphores as the synchronization mechanism. **[3 + 3+ 3 Marks]**
Make sure to initialize the semaphores correctly **[3 Marks]**

Question 2: (10 Marks)
[Attempt this Question on the Answer Sheet]

Canada and US are separate threads executing their respective procedures shown below. The code below is intended to cause them to forever take turns exchanging insults through the shared variable X in strict alternation. The Sleep() routine blocks the calling thread, and the Wakeup() routine unblocks a specific thread if that thread is blocked.

// Initializing Semaphores
creatableHoles = MAX // Danyal's sem
emptyHoles = 0 // Sohaib's sem
fillableHoles = 0 // Fahad's sem
shovel = 1 // binary sem that Danyal & Fahad

Danyal()
    wait(creatableHoles)
    while(!shovel);
    shovel = 0 // Crit. sec
    signal(emptyHoles) // Digging Hole
    shovel = 1 // end of C.S

Sohaib()
    wait(emptyHoles)
    signal(fillableHoles)

Fahad()
    wait(fillableHoles)
    while(!shovel)
    shovel = 0
    signal(creatableHoles)
    shovel = 1

ok at these 3
sleep();
x = showInsult
wakeup(US Th

threads can
to miniscule
cannot be
in both US a
sequentially.
It is very
runs wake
run both
the US The
resulting
time

b) S

Look at these 3 lines:

sleep();	x = shoutinsult(x);
x = shoutinsult(x);	wakeup(ThreadCanada);
wakeup(USThread);	sleep();

As threads can execute at different speeds due to miniscule changes in the CPU's speed, we cannot be 100% sure that the lines in both US and Canada codes will execute sequentially.
It is very likely that after the USC) thread runs wakeup(ThreadCanada), the Canada() thread may run both ShoutInsult() and wakeup(USThread) before the USThread can go to sleep (or vice versa), resulting in both threads sleeping at the same time and causing a deadlock.

b)
```
Semaphore s=1;
void Canada(){
        while(1){
            wait(s);
            x=ShoutInsult(x);
            signal(s);
        }
}
void USC){
        while (1){
            wait(s);
```

7.5

Q/Part No.

```
X = ShoutInsult(X);
signal(s);
    }
}
```

```
void
Canada ()
{
  while (1) {
    Sleep();
    X = ShoutInsult(X);
    Wakeup(USThread);
  }
}
```

```
void
US()
{
  while (1) {
    X = ShoutInsult(X);
    Wakeup(ThreadCanada);
    Sleep();
  }
}
```

a. The code shown above exhibits a well-known synchronization flaw. Outline a scenario in which this code would fail, and the outcome of that scenario. What is this flaw called? **[4 Marks]**

b. Show how to fix the problem using semaphores, replacing the Sleep() and Wakeup() calls with semaphore Wait (P/down) and Signal (V/up) operations. Note: Disabling interrupts is not an option. **[6 Marks]**

Question 3: (15 marks)

[Write your answer on the Question Paper only in the provided space below]

Consider the following page reference string:

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

Now assume that the total number of frames in main memory are 5. Fill the tables below using the FIFO,OPTIMAL and LRU page replacement algorithms for the given memory by filling in the table with page numbers. The top row is the string of memory references and each row contains the page numbers that resides in each physical frame F1-F5 after each memory reference.

Note: 1) If two or more frames are empty, the lower frame number is filled first. 2) If more than one page can be replaced according to the given policy, the page in the lower frame number will be replaced.

FIFO: //Assuming that lower frame number means FSCF1 (it doesn't make a difference running time).

	1	2	3	4	2	1	5	6	2	1	3	7	6	3	2	1	2	3	6
F1							5	5	5	5	5	5	5	5	2	2	2	2	2
F2				4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3
F3			3	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7
F4		2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1
F5	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	6	6
	F	F	F	F	H	H	F	F	H	F	H	F	H	F	F	H	H	H	H

LRU:

	1	2	3	4	2	1	5	6	2	1	3	7	6	3	2	1	2	3	6
F1							5	5	5	5	5	7	7	7	7	7	7	7	7
F2				4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
F3			3	3	3	3	3	6	6	6	6	6	6	6	6	6	6	6	6
F4		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	F	F	F	F	H	H	F	F	H	H	F	F	H	H	H	H	H	H	H

OPTIMAL:

	1	2	3	4	2	1	5	6	2	1	3	7	6	3	2	1	2	3	6
F1							5	~~5~~	~~5~~	~~5~~	~~5~~	7	7	7	7	7	7	7	7
F2				4	4	4	4	~~6~~	~~6~~	~~6~~	~~6~~	6	6	6	6	6	6	6	6
F3			3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
F4		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	F	F	F	F	H	H	F	F	H	H	H	F	H	H	H	H	H	H	H

Here, Replaced 6 in F2 with 6 because its the lower ~~order~~ page frame according to my assumption.

	Page Fault Ratio	Page Hit Ratio
FIFO	10/19=52.6%.	9/19=47.3%.
LRU	8/19=42.1%	11/19=57.9%.
OPTIMAL	7/19= 36.8%.	12/19=63.1%.

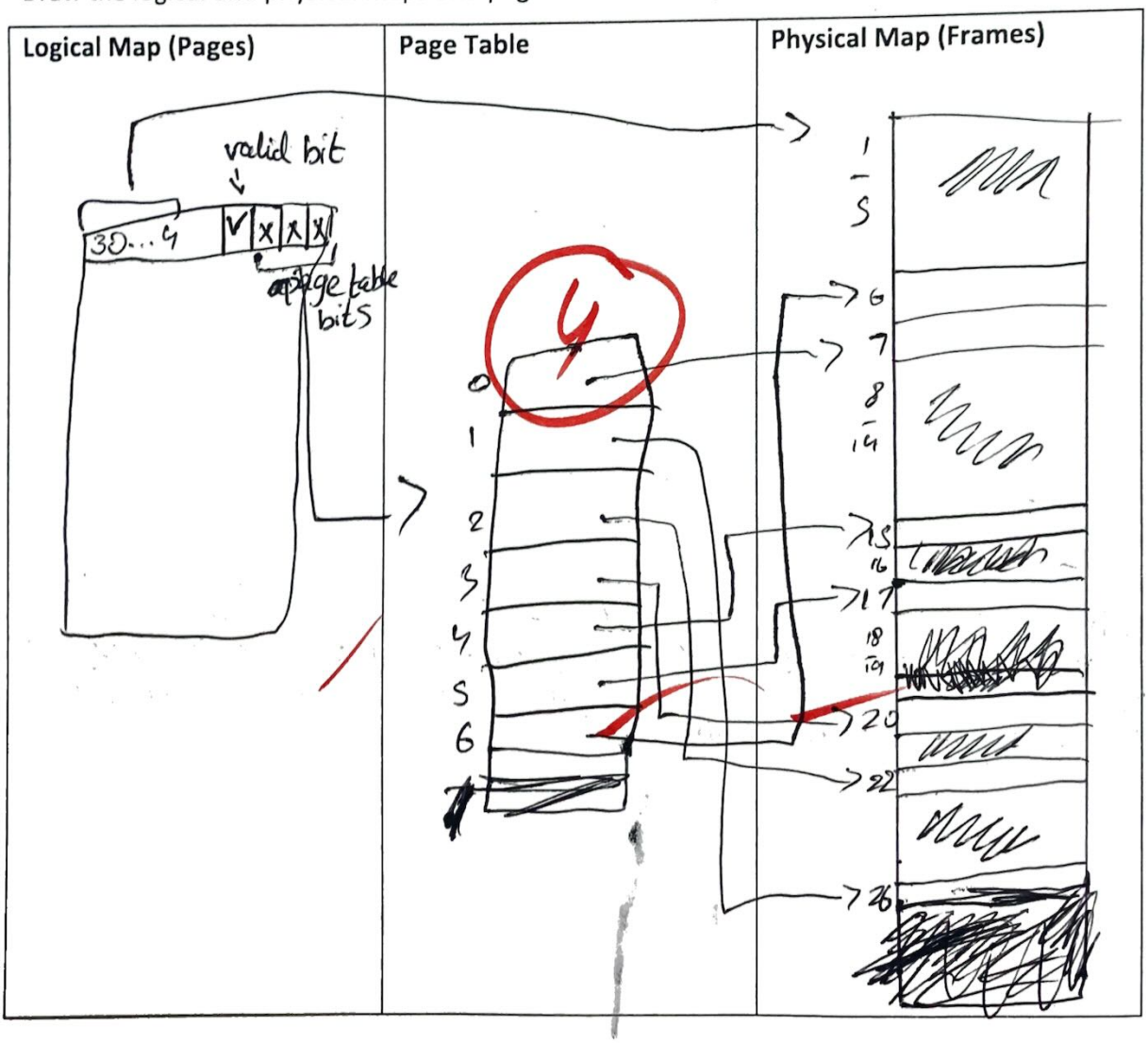
15

Question 4: (10 Marks)

[Write your answer on the Question Paper only in the provided space below]

Consider a user program of logical address of size 7 pages and page size is 4 bytes. The physical address contains 30 frames. The user program consists of 26 instructions a, b, c, . . . y, z . Each instruction takes 1 byte. Assume at that time the free frames are 7, 26, 22, 20, 15, 17 and 6. The first page will be kept at frame 7, the second page will be kept at 26, and so on.

Draw the logical and physical maps and page tables in the space. **[4 marks]**



The physical address of **j** [1 mark]	0000000 0000000 0001 0000000 000
The physical address of **o** [1 mark]	0000000000001 0000000 0000000
The physical address of **s** [1 mark]	0000000 0000000000000 000000
The physical address of **x** [1 mark]	0010 0000000 0000000000000
Amount of internal fragmentation *(in bytes)* **[2 marks]** Solution: 2 Bytes	6 bytes

Process #
P0
P3
P4
P1
P2
P3
P4
P8
P9
P8
P11

Question 5: (15 Marks)
[Write your answer on the Question Paper only in the provided space below]

The *Bakery Algorithm* for "n process mutual exclusion" is shown below.

```
boolean choosing[ n ];
int  number[ n ];
1  do {
2       choosing[ i ] = true;
3       number[ i ] = max(number[0], number[1], ..., number [n – 1])+1;
4       choosing[ i ] = false;
5
6       for (j = 0; j < n; j++) {
7               while (choosing[ j ]) ;
8               while ( (number[ j ] != 0) && (number[ j ], j ) <  (number[ i ], i ) )) ;
9       }
11              critical section
13       number[ i ] = 0;
14
15              remainder section } while (1);
```

A. We know that Bakery Algorithm does not guarantee that two processes do not receive the same token. In case of a tie, the process with the lowest ID is served first. Keeping this in mind, what is the purpose of **choosing** in lines 2 , 4 and 7? If we remove lines 2,4 and 7, will the code work fine. If so how? **[3 marks]**

Choosing ensures that we don't attempt to access the number of a specific process while it is ~~se~~calculating it. If we removed this, then processes could simultaneously be in the critical section as number[j] may be 0 ~~before~~ when process i checks.

B. Suppose there are 14 processes i.e. Process 0 – 14. Some processes require the critical section. The order of execution in which they require critical section is:

P0,P3,P4,P1,P2,P3,P4,P8,P9,P8,P11,P6

Fill the status of the circular Queue after each process requests entry into critical section, i.e. after execution of line 4 in the above code: **[6 marks]**

Process #	Queue
P0	(1,0,0,0,0,0,0,0,0,0,0,0,0,0)
P3	(1,0,0,2,0,0,0,0,0,0,0,0,0,2)
P4	(1,0,0,0,3,0,0,0,0,0,0,0,0,2)
P1	(1,4,0,2,3,0,0,0,0,0,0,0,0,2)
P2	(1,4,5,2,3,0,0,0,0,0,0,0,0,2)
P3	(1,4,5,6,3,0,0,0,0,0,0,0,0,2)
P4	(1,4,5,6,7,0,0,0,0,0,0,0,0,2)
P8	(1,4,5,6,7,0,0,0,8,0,0,0,0,2)
P9	(1,4,5,6,7,0,0,0,8,9,0,0,0,2)
P8	(1,4,5,6,7,0,0,0,10,9,0,0,0,2)
P11	(1,4,5,6,7,0,0,0,10,9,0,11,0,2)
P6	(1,4,5,6,7,0,12,0,10,9,0,11,0,2)

C. At some point, the process numbers and their corresponding ticket numbers are shown.

Process #:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Ticket#:	**7**	**0**	**0**	**5**	**0**	**3**	**0**	**0**	**0**	**6**	**0**	**3**	**0**	**0**

What is the meaning of Ticket# = 0? **[1 mark]**

That the process does not want access to critical section. (1)

D. Considering the information contained in Part C, replace the blank with a proper process number: **[5 marks]**

Process # 0 waits for process # 9 ✓
Process # 3 waits for process # 5 ×
Process # 5 waits for process # ~~None~~ 11 ×
Process # 9 waits for process # 3 ✓
Process # 11 waits for process # ~~None~~ ×

(2)

Question 6: (8 Marks)

[Write your answer on the Question Paper only in the provided space below]

Given the code segment provided, which creates multiple child processes, assume that the operating system maintains a variable NEXT_PROCESS_ID initialized to 100. Each time a new process is created, it is assigned the value of NEXT_PROCESS_ID as its process ID. The NEXT_PROCESS_ID is then incremented to prepare for the next process creation request. There are no compilation or execution errors in this code.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define BUFFER_SIZE 25
```

```
int main () {

  char msg[BUFFER_SIZE] = "Welcome";
  pid_t pid = fork ();

  if (pid > 0) {
  strcpy (msg, "Welcome to OS course");
  printf ("Parent process waiting for child termination \n"); wait (NULL);
  printf ("Parent Terminating \n"); }
  else {
  printf ("Message: %s \n", msg);
  pid_t pid1 = fork ();

   strcpy (msg, "OS course");
   pid_t pid2 = fork ();

   if (pid2 == 0) {
          strcpy (msg, "Adv OS course");
          printf ("Child Process called \n");
   }
   else {
           wait (NULL);
           printf ("Message: %s \n", msg);
   }
    if(pid1 > 0) {
          wait(NULL);
    }
   }
   return 0;
  }
  return 0;
```

Child Process called
Message: OS course
Message: OS course
Child Process called
Parent Terminating

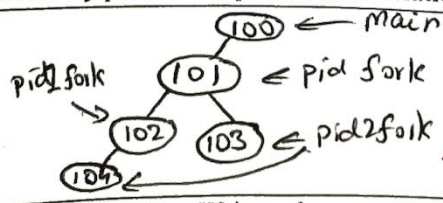
child
2 created
2 outputs
4 created
101, 102
103

ignoring these because it IS a compilation error.

a) How many *new* processes are created *(do not count the initial main() process)*? **[2 Marks]**

4

b) Create a **process tree** illustrating each process node along with its corresponding process ID. The tree should distinctly present the parent-child relationships for all processes. **[3 Marks]**



c) Show output of the code below. Write only one sequence if you feel that multiple sequences can be printed. **[3 Marks]**

~~Welcome~~ Parent process waiting for child termination
Message: Welcome

Child Process called
Message: OS Course
Message: OS Course
Child Process called
Parent Terminating

8

Question 7: (30 Marks)
[Write your answer on the Question Paper only in the provided space below]

Suppose that four processes, P1, P2, P3 and P4, running simultaneously in a multiprocessor system consisting of 3 **processors**. Following is the data about the processes and scheduling policies.

- P1 has three threads. P2 has four threads. P3 has five threads. P4 has three threads.
- All threads in all the processes are CPU-bound, i.e., they never block for I/O.
- There is a strong processor affinity and no load balancing policy in place. i.e. once a thread is tied to a processor, it completes its execution with the same processor.
- There is kernel level thread scheduling in place.
- Queue 1 serves Processor 1, Queue 2 serves Processor 2 and Queue 3 serves Processor 3.
- All the processors are running Round Robin scheduling. Following are the time quantum for each queue.
  - Queue 1 time quantum (q) = 1 ms
  - Queue 2 time quantum (q) = 3 ms
  - Queue 3 time quantum (q) = 2 ms
- All the threads have been loaded in their respective Queues and the current status of the queues at time interval 10 is as shown below (P1T1 means Process 1 – Thread 1 and so on):

Queue 1

P1T1	P3T2	P4T2	P3T5	P2T4	Processor 1

Queue 2

P3T1	P4T1	P1T3	P2T1	Processor 2

Queue 3

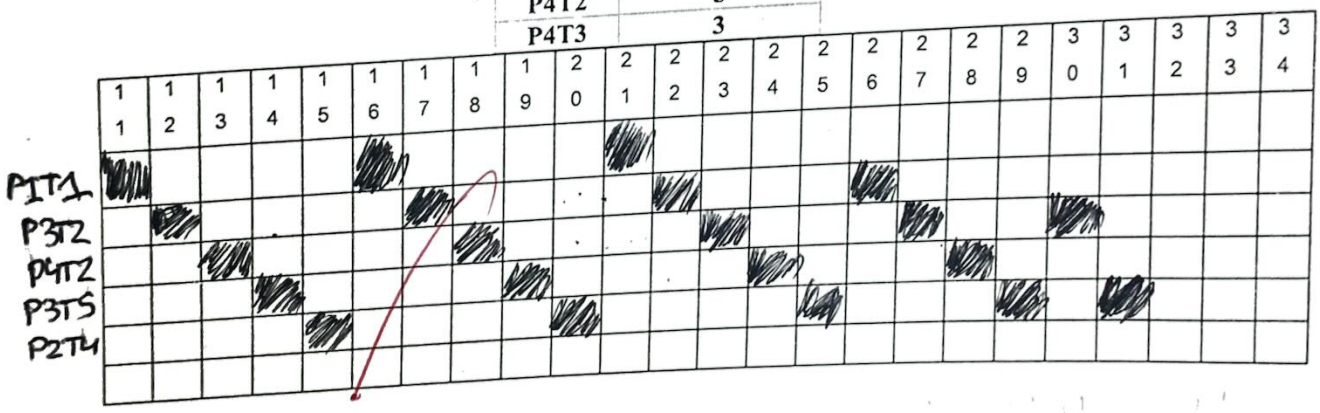
P4T3	P3T3	P2T2	P2T3	P1T2	P3T4	Processor 3

There is currently no process in the processor at time interval **11** i.e. the last process in each of the processor has been context switched out and dispatcher can schedule the next thread immediately at time **11**. The context switch (dispatch latency) cost is not given, so assume it as zero.

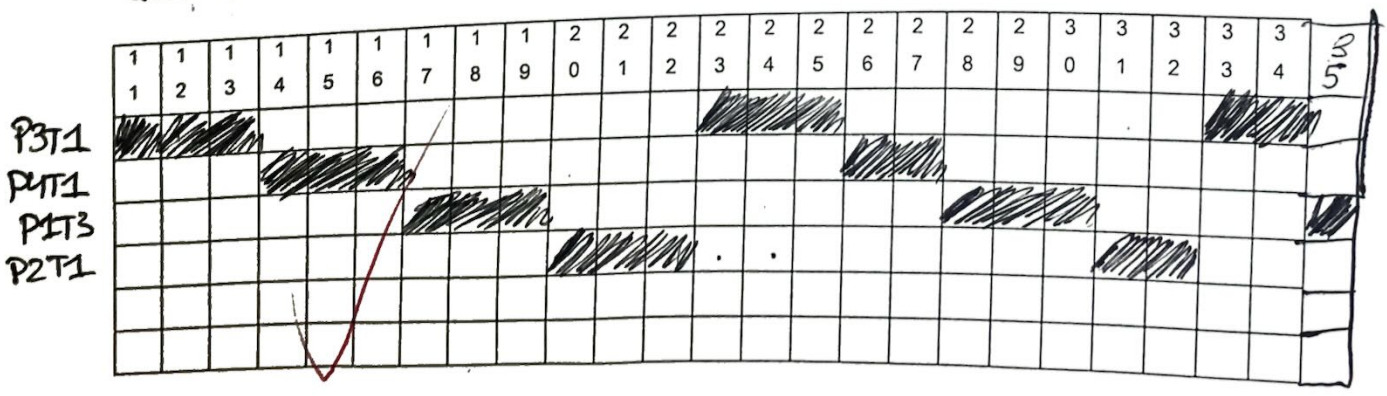
Fill in the following Gantt charts below for each Processor to answer the questions given at the end:

Threads	Remaining CPU Time (ms)
P1T1	3
P1T2	5
P1T3	7
P2T1	5
P2T2	4
P2T3	4
P2T4	5
P3T1	8
P3T2	4
P3T3	5
P3T4	3
P3T5	4
P4T1	5
P4T2	5
P4T3	3

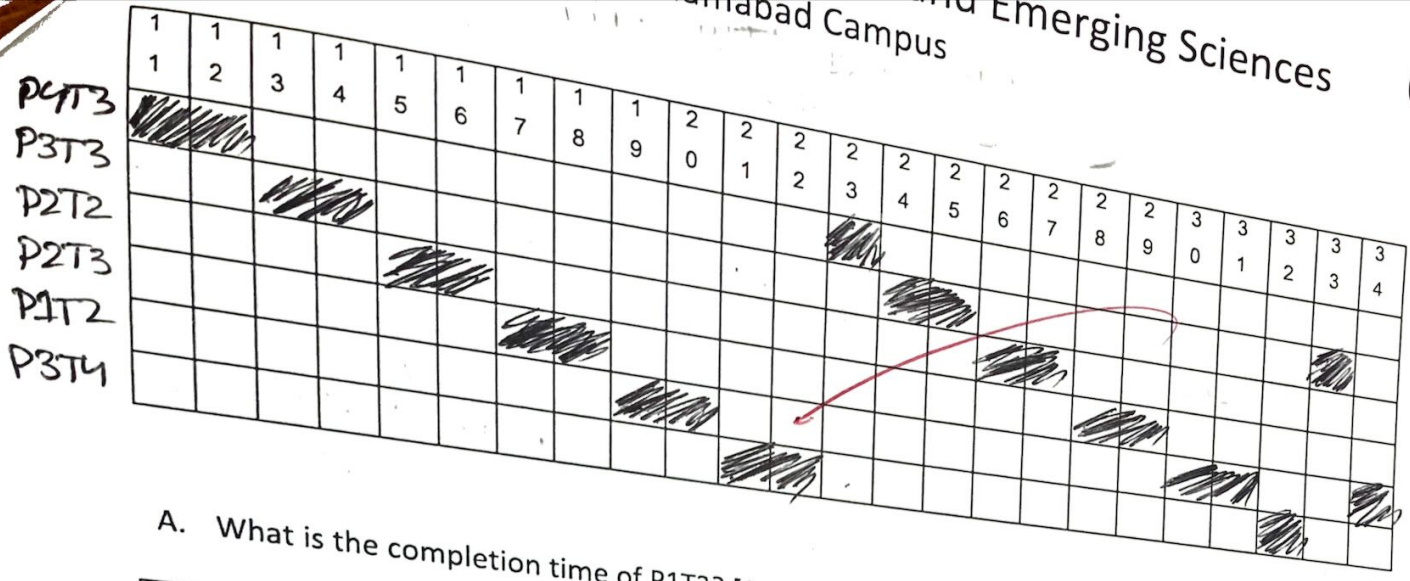
Queue 1 / Processor 1 – Gantt Chart **[3 marks]**



Queue 2 / Processor 2 – Gantt Chart **[3 marks]**



Queue 3 / Processor 3 – Gantt Chart **[3 marks]**



A. What is the completion time of P1T3? **[1 mark]**

T= 35

B. What is the completion time of P1T2? **[1 mark]**

T=34

C. What is the completion time of P2T4? **[1 mark]**

T=31

D. What is the completion time of P3T2? **[1 mark]**

T=26

E. What is the completion time of P3T3? **[1 mark]**

T=33

F. What is the completion time of P4T2? **[1 mark]**

T=30

G. What is the average waiting time of Process P1 after interval 10 (i.e. starting from 11) ? **[2 marks]**

Avg waiting time of P1=(waiting time of P1+w.t of P2+w.t of P3)/3=(8+18+19)/3=45/3=15

H. What is the average waiting time of Process P2 after interval 10 (i.e. starting from 11) ? **[2 marks]**

= (16+17+13+15)/4 = 15.25

I. What is the average waiting time of Process P3 after interval 10 (i.e. starting from 11) ? **[2 marks]**

=(12+14+16+18+19)/5 = 15.8

J. What is the average waiting time of Process P4 after interval 10 (i.e. starting from 11) ? **[2 marks]**

=(15+12+10)=37/3= 12.3

K. What percentage of all processor's time will be spent running Process P1 during interval 11-19? [1 mark]

Proc.1 out of 9 spends 2 quantums, Proc.2 spends 3 out of 9, Proc.3 spends 1 of 9. Total = $\frac{2+3+1}{27}$ = 22.2%.

L. What percentage of all processor's time will be spent running Process P1 during interval 15-24? [1 mark]

Proc.1: 2 of 10 intervals, Proc2: 3 of 10, Proc.3: 2 of 10. total = $\frac{2+3+2}{30}$ = 23.3%.

M. What percentage of all processor's time will be spent running Process P2 during interval 15-24? [1 mark]

Proc.1: $\frac{2}{10}$, Proc.2: $\frac{3}{10}$, Proc.3: $\frac{4}{10}$ $(\frac{2+2}{10})$ => $\frac{2+3+4}{30}$ = 30%.

N. What percentage of all processor's time will be spent running Process P3 during interval 15-24? [1 mark]

Proc1: $\frac{4}{10}$, Proc2: $\frac{2}{10}$, Proc3 $\frac{3}{10}$ => $\frac{4+2+3}{30}$ = $\frac{9}{30}$ = 30%.

O. What percentage of all processor's time will be spent running Process P3 during interval 20-29? [1 mark]

Proc 1: $\frac{4}{10}$, Proc2: $\frac{3}{10}$, Proc 3: $\frac{4}{10}$ => $\frac{4+3+4}{30}$ = $\frac{11}{30}$ = 36.6%

P. What percentage of all processor's time will be spent running Process P4 during interval 20-29? [1 mark]

Proc1: $\frac{2}{10}$, Proc 2: $\frac{2}{10}$, Proc 3: $\frac{1}{10}$ => $\frac{2+2+1}{30}$ = $\frac{5}{30}$ = 16.6%.

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2016
	Duration:	2½ hours	Total Marks:	75
	Paper Date:	22 December, 2016	Weight:	45%
	Section:	A-B	Page(s):	2
	Exam:	Final	Roll No:	

Instructions/Notes: Fill the answers in the first 10 pages of your answer sheet.

Question 1 (5 points): What is the purpose of command line interpreter? Why is it not always part of the kernel? Give the most important reason.

Question 2 (5 points): What is micro kernel architecture? List at least **three** benefits.

Question 3 (5 points): In Linux the open-file table is used to maintain information about files that are currently open. Should the operating system maintain a separate table for each user or maintain just one table that contains references to files that are currently being accessed by all users? If two users try to access the same file, how operating system manages the situation?

Question 4 (10 points): There are three processes running on a machine, call them P_1, P_2 and P_3. There are three resources on that machine, call them R_1, R_2 and R_3. The processes are scheduled according to Round Robin scheduling, where the quantum size is Q. Each process runs for 15 quanta and then terminates. Each process tries to acquire one resource in each quantum, in following manner.

- P_1: R_1, R_2, R_3
- P_2: R_2, R_3, R_1
- P_3: R_3, R_1, R_2

For example, process P_1 tries to acquire resource R_1 in first quantum, R_2 in second quantum and R_3 in third. While process P_3 acquires R_3 in first quantum, R_1 in second quantum and R_2 in third quantum. Similarly process P_2 starts from R_2 and proceeds to R_3 and R_1. No process releases the resource until its last quantum (quantum number 15). Explain, will there be any deadlock or not? If no, then why? If yes, then when the deadlock will occur? If the deadlock occurs, then propose a technique which prevents the deadlock. Draw diagrams for help.

Question 5 (10 points): Custom File System (CFS) uses indexed allocation for the files. Indexing is done at a single level. You have to implement a function **fetchByte()** to fetch a byte of a file from CFS. The block size in CFS is **P** bytes. The implementation may only contain the pseudo code. The parameters of the function are following

1. Number of the block where the index is stored, call it **INB**.
2. Logical byte number of the file whose data we have to fetch, call it **BN**.

Question 6 (10 points): Following is an inverted page table on a machine. A process whose ID is 1 accesses the following byte numbers, translate these addresses into physical addresses. The size of one page is **1000** bytes.

- 2345
- 87
- 4315
- 123

PID	Page #	Frame #
1	2	12
2	5	4
3	3	6
1	4	7
2	3	2
2	7	9
4	1	13
1	0	15
2	6	10

Question 7 (10 points): Suppose there is a dual core computer machine installed at a manufacturing unit. designing a custom OS for that machine. The machine has a large Translation Look Aside Buffer (TLB). The store 4×10^6 unique entries in it. The key for an entry can be made composite, for example composite key of Page number. There can be only two processes at a time running on that machine. The memory requirement process will not exceed more than $4GB$. Memory page size on the machine is $4K$ bytes.
Under the given circumstances provide the best technique of memory management which exploits the large size You have to answer the following questions **with reason**

- What will be the size of a process page table?
- Which type of page table should you use? given that you have a very large TLB.
- Should you consider to divide the page table into smaller parts or not?

Question 8 (10 points): Following is code for producer-consumer problem. There is a mistake in this code, you **identify** the mistake and then **fix** it. The best solution will be the one which only repositions a single statement. what is the problem in this code?

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

```
while(true)
{
 item = produce()
 sem_1.wait()
 buffer.add(item)
 sem_2.signal()
 sem_1.signal()
}
```

```
while(true)
{
 sem_1.wait()
 sem_2.wait()
 item = buffer.get()
 sem_1.signal()
 item.process()
}
```

Question 9 (10 points): In the above code the **buffer** object can store infinite number of elements. Modify th code so that it can work for a **buffer** which can only store N number of elements.

Date: May 6, 2016 **Marks:** 65 **Time:** 150 min

Instructions:

1. Submit the objective part before starting this exam.
2. Try to solve question 1 on page 1, question 2 on page 2, and so on. Only the first 10 pages of the answer sheet will be marked.

Question 1 (10 marks)

As you know, the files are stored in the form of blocks. Every block of a file has a logical block number. The first block has the logical block number 0, the second has no 1, the third equal to 2, and so on and so forth. Assume a file named "outline.pdf" is stored in a FAT based filesystem, andits first block is stored at physical block no5.(Means its 0th logical block number maps to physical block number 5).The rest of the file-blocks can be reached by looking at the following file allocation table:

6	EOF	7	8	9	2	4	0	1	3
0	1	2	3	4	5	6	7	8	9

Now, translate the following logical block numbers into the corresponding physical block numbers:

(a) 9 (b) 6 (c) 2

Question 2 (5 + 5 marks)

(a) Show execution of the Optimal page replacement algorithm on the following page-reference string, assuming four frames:

1 2 3 4 1 5 6 7 8 1 2 3 4

(b) Repeat part (a) using the LRU algorithm.

Question 3 (5 + 5 marks)

Question 2 (5 + 5 marks)

(a) Show execution of the Optimal page replacement algorithm on the following page-reference string, assuming four frames:

1 2 3 4 1 5 6 7 8 1 2 3 4

(b) Repeat part (a) using the LRU algorithm.

Question 3 (5 + 5 marks)

Consider the following page table of a process P:

Frame#	50	20	70	10	90	30	80	40	60	55
Page#	0	1	2	3	4	5	6	7	8	9

Now, convert the following logical addresses into the corresponding physical addresses, assuming a page size of 1000 bytes (not 1024):

(a) 3500 (b) 7400

Operating Systems
Final Exam, Spring 2016

Date: May 6, 2016 **Marks:** 65 **Time:** 150 min

Question 4 (5 marks)

Consider the following formulas to compute process priority in a system:

Formula A: priority = (CPU used in last two seconds)
Formula B: priority = 1 / (CPU used in last two seconds)

Assume there is an inverse relationship between priority and its value; for example 10 is higher than 20 ... Which of the two formulas is better? Give reason for your answer.

Question 5 (10 marks)

In the following applications/programs, where shall we use threads, and where shall we use processes? Give reasons for your answers. Be precise and concise.

a) Searching or sorting a large array
b) A program that executes different applications during its execution
c) An internet browser that can load multiple web pages in its different tabs

Question 6 (5+5 marks)

(a) The following three threads (note that there are exactly 3 threads, no more no less) generate a string containing characters a, b and c in an arbitrary order. Following are few examples:

abcbacccbaab, ababababcbcbcbaaaabbbc, ccccccbabababababababacccacb

a) Searching or sorting a large array
b) A program that executes different applications during its execution
c) An internet browser that can load multiple web pages in its different tabs

Question 6 (5+5 marks)

(a) The following three threads (note that there are exactly 3 threads, no more no less) generate a string containing characters a, b and c in an arbitrary order. Following are few examples:

abcbacccccbaab, abababcbcbcbaaabbbc, ccccccbabababababababacccacb

`// thread 1`
`while (1){`
`cout<< "a";`
`}`
	`// thread 2`
	`while (1){`
	`cout<< "b";`
	`}`
		`// thread 3`
		`while (1){`
		`cout<< "c";`
		`}`

Synchronize the threads using semaphores so that the printed string is a concatenation of the substring "cba". Here are few examples:

cba, cbacba, cbacbacba, ...

(b) Now again synchronize the threads in part (a) such that instead of infinite concatenation of substring "cba" an infinite concatenation of substring "cbbba" is generated ... Note that you are not allowed to add or modify any **cout** statement.

as possible; it shall not wait for termination of the first program.

Question 7 (10 marks) **(For Section A and B Only)**

(a) Following table shows the arrival time of four processes.

Process Name	Arrival Time
P1	1
P2	5
P3	7
P4	9

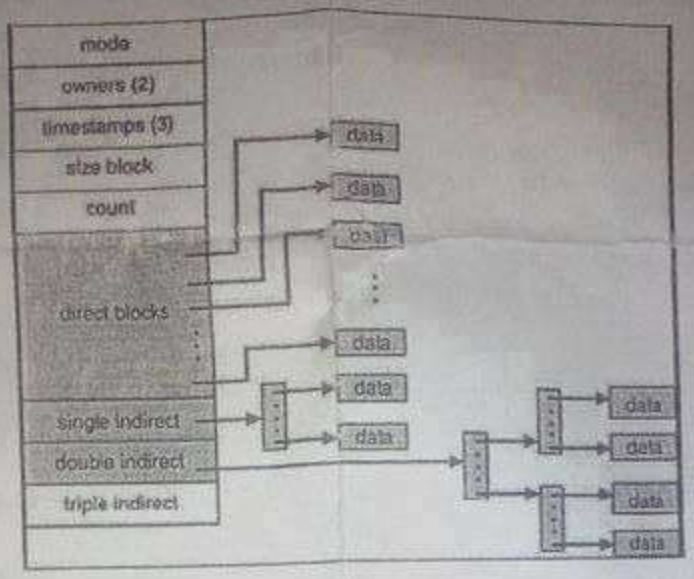
We assume that there are 3 resources **R1**, **R2**and **R3**, which are free initially. Following table shows the requests from processes, fill on the right side of following table, the results of these requests using wait and die algorithm.

Request	State of Resources
	R1, R2, R3 are free
P1 requests R1	R1 is assigned to P1
P2 requests R2	
P1 releases R1	
P3 requests R1	
P4 requests R3	
P4 requests R1	
P1 requests R3	

Date: Dec 24, 2013 Marks: 100 Time: 3 hours

Question 5 (15 marks)

Consider an operating system using Indexed Allocationfor storage of files:



Give a C/C++ function to compute the physical block number of a file from a given logical block number in such a system. Assume the INODE is read into the following C structure and there is a function `read(long block)` which can read any data block on a volume. Also assume that the block size is 4k bytes.

```
structINode {
        // ...
        long block[12];  // addresses (numbers) of the first 12 data blocks
        long indirect;   // address of the single indirect block
        longdblInd;      // address of the double indirect block
        longtplInd;      // address of the triple indirect block
};
```

Following is the prototype of your function. It takes a file's inode and a logical block number, and returns the physical block number: `long map(INodeinode, longlogBlk)`

Question 6 (15 marks)

Consider an operating system that recalculates the process priorities once per second using the following formula:

priority = (recent CPU usage / 2) + 60 (the higher the number, the higher the priority)

a) What would happen to an I/O bound process, and what would happen to a CPU bound process?

b) Is this treatment good or bad in an interactive environment? Why or why not?

c) Can you improve the above algorithm? If yes how would you do that?

Question 3 (10 marks)

Consider a computer system that uses XYZ processor. This processor supports virtual memory with a page size of 16 bytes and has instructions of size 4 bytes. The address bus of XYZ processor is 16 bits, while the data bus is 32 bits (i.e., all memory reads and writes are in 4 bytes).

A process P1 running on this computer system executes the following code:

```
Load  R1  [0x001C]   /* Load the value at address 0x001C into Register R1 */
Load  R2  [0x0074]   /* Load the value at address 0x0074 into Register R2 */
Add   R1  R2         /* Add the values of Registers R1 and R2 and store the result in R1 */
Store R1  [0x0038]   /* Store the value in register R1 at address 0x0038 */
```

Figure 3(a) shows the page table of process P1 and Figure 3(b) shows the state of the physical memory before execution of the above code. What would be the state of the Physical memory after execution of the above code. You need to show the state by drawing the Figure 3(c) with (only) new values in your answer book.

	Frame No.	Valid/Invalid
0	0	Valid
1	3	Valid
2	0	Invalid
3	1	Valid
4	0	Invalid
5	0	Invalid
6	0	Invalid
7	2	Valid

(a)

Physical Address	
0x0000	Process P1 Code
0x0004	
0x0008	
0x000C	
0x0010	0x90000018
0x0014	0x00000055
0x0018	0x00000024
0x001C	0x00000000
0x0020	0x00000098
0x0024	0x00000005
0x0028	0x00002004
0x002C	0x00000007
0x0030	0x00030045
0x0034	0x00000054
0x0038	0x00000036
0x003C	0x00000004

(b)

Physical Address	
0x0000	Process P1 Code
0x0004	
0x0008	
0x000C	
0x0010	
0x0014	
0x0018	
0x001C	
0x0020	
0x0024	
0x0028	
0x002C	
0x0030	
0x0034	
0x0038	
0x003C	

(c)

Figure 3.(a) Page Table of process P1. (b) Memory state before execution of the above code. (c) Memory state after the execution of the above code (you need to fill-in the memory state).

Operating Systems
Final Exam, Spring 2015

Date: June 11, 2015 **Marks:** 60 **Time:** 3 hrs

Question 4 (15 marks)

Consider a signal where vehicles move in four directions: north, south, east and west. At a time, the vehicles move either north/south or east/west; turns are not allowed. Following are four functions: one for each direction. The vehicles (threads) moving north execute the function moveNorth, the vehicles moving south execute the function moveSouth,and so on. Synchronize the threads using semaphore(s). (Do not worry about starvation; assume breaks come in the traffic regularly after some time.

// Type 1	// Type 2	//Type 3	//Type 4
void moveNorth() {	void moveSouth() {	void moveEast() {	void moveWest() {
...	...	...	...
// cross signal	// cross signal	// cross signal	// cross signal
...	...	...	...
}	}	}	}

Question 5 (15 marks)

Consider a file system using a variation of the Indexed Allocation. The first n-1 pointers of the first index block point directly to the data blocks. The last pointer, however, points to another index block (second index). Similarly the second index contains n-1 direct pointers, and the last pointer points to the next index block (third index), and so on.

Assuming a block size of one KB and a pointer size of four bytes, write a C/C++ function to compute the physical-block-number for a given logical-block-number of a file. Following is the function prototype:

int find(int index_block, long logical_block)

Here, index_block holds the address of the first index block of the file, and the logical_block contains the given logical-block-number. The function returns the corresponding physical block number after computation.

Assume we have a function available to read a disk block as follows:

bool read(long block_num, char* buffer)

Here 'block_num' is the address of the block to read, and 'buffer' is the target memory buffer/array.

Operating Systems
Final Exam, Spring 2016

Date: May 6, 2016 **Marks:** 65 **Time:** 150 min

Question 7 (10 marks) **(For Section C and D Only)**

Write a C/C++ program that executes any given two programs using the Unix (Linux) Pipeline. The program takes names of any two existing programs as input. It then executes the first program, while redirecting its output to a pipe. Finally, it executes the second program with its input redirected to the same pipe.

Note that your design shall support as much concurrency (parallelism) as possible. The second program shall start as soon as possible; it shall not wait for termination of the first program.

Question 7 (10 marks) **(For Section A and B Only)**

(a) Following table shows the arrival time of four processes.

Process Name	Arrival Time
P1	1
P2	5
P3	7
P4	9

We assume that there are 3 resources R1, R2and R3, which are free initially. Following tab

Operating Systems
Final Exam
Time 3 Hours-Marks 60

Q1. Following is the situation of FAT when the user requested to store a file of 25678 bytes.

A. Mark how the file system is going to allocate space in the File Allocation Table for this file if the cluster size is 2k. Non zero entries are used clusters, the first entry is cluster 0 counting onwards. (10)

B. Mark how the file system is going to allocate clusters if it tries to do contiguous allocation to maximize access speed. (5)

C. How many files are stored on this hard disk? (5)

0 ⇨

X	X	X	EOF	0	0	X	X	EOF	EOF	X	X	0	0	X	X	X	EOF	X	X
X	X	X	0	0	0	0	0	EOF	EOF	X	X	X	EOF	0	0	0	X	X	X
EOF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	EOF
X	X	X	X	EOF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Q2. What is the main difference between Tree Structured and Acyclic Graph Directories? (5).

Q3. In a pure demand paging environment, CPU executes the first instruction of a program:

```
or [0346],0x55AA
```

Write step by step scenarios of how the page faults occur and how the Operating System responds to these faults. (10)

Q4. Define the following: (5X2=10)

a. Semaphore
b. Mutex
c. Memory Compaction
d. Dynamic Linking when required
e. Demand Paging

Q5. A UNIX file is represented by an inode. Assume that there are 12 direct block pointers, and a singly, doubly, and triply indirect pointer in each inode. The file system block size and disk sector size are both 8Kbyte. If the disk block pointer is 32 bits (4 bytes), with 8 bits to identify the physical disk and 24 bits to identify the physical block, then

- What is the maximum file size supported by this system?
- What is the maximum file system partition supported by this system?
- Assuming that "inode" is already in memory, how many disk accesses are required to access the byte at position 13,423,956?

(3X5=15)

Operating Systems

Final Exam

Time 3 Hours-Marks 90

Q1. Following is the situation of FAT after user has deleted a file. The deleted directory entry states that the size of the file was 25678 and its starting cluster was 34 (the one marked).

a. Assuming that nothing has been done afterwards, undelete the file and display the condition of FAT after un-deletion. (10)

b. If we know that the system gives priority to contiguous allocation, what would have been the first cluster? (5)

c. How many files are stored on this hard disk? (5).

d. Why is it not possible to have Acyclic Graph Directories in the FAT file system? (5)

0 ⇨

X	X	X	EOF	0	0	X	X	EOF	EOF	X	X	0	0	X	X	X	EOF	X	X
X	X	X	0	0	0	0	0	EOF	EOF	X	X	X	EOF	0	0	0	X	X	X
EOF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	EOF
X	X	X	X	EOF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Q2.

a. If you are entrusted to design an MMU for AMD64 (OOPS!), what type of a paging system will you suggest for this machine which has a 64 bit address space? (5).

b. What is the main benefit of using multilevel page tables? (5)

c. Discuss some merits and de-merits of multitasking and multithreading in the context of paging. (5)

d. We have two systems running LINUX. One that uses paging and the other one do not have an MMU so it cannot use paging. Can you elaborate the difference of behavior of the `fork()` system call in both the systems. (10)

e. Kernel has two types of memory pools, paged and non-paged. Can you name one part of the kernel code which is mandatory to be kept in the non-paged pool? (5)

Q3.

a. Generally, mutexes and semaphores are created in the kernel space and there are system commands to use them. Give the one good reason (in a single sentence) that they are not created in the process space. (5)

b. What is the main difference in the usage of a Mutex and a Semaphore? (5)

Q4. Write short notes on the following: (5X5=25)

1. TLB
2. Address Binding
3. Dynamic Linking
4. INODE
5. Master Boot Record

☠☠☠☠☠ GOOD LUCK ☠☠☠☠☠

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(){

//approaching bridge entrance

//your synchronization logic goes here

Pass the bridge code.

// exiting the bridge

// and your synchronization logic goes here

}
```

```
Northbus(){

//approaching bridge entrance

//your synchronization logic goes here

Pass the bridge code.

// exiting the bridge

// and your synchronization logic goes here

}
```

```
class ProcessQ{ // it is the list of all processes in ready queue
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.
{
    addToQ(leavingProcess);
    return removeFromQ();
}
```

```
ProcessQ readyQ = getReadyQ();
readyQ.addToQ(leavingProcess);
return readyQ.removeFromQ();
```

8/

Question 3 (10 points): A node senses that the leader has died in the ring. It starts the election procedure. You are NOT writing the code for the node which initiates the election, rather you are writing the code for the node which is only receiving the election message. Implement the function 'onReceiveMessage'.
We assume that no node will fail AFTER the election has started. There may be nodes which have failed before the election, which we do not know of. Also there are at least two nodes in the ring, one which has started the election, the other which is calling the below given function. Of course, more than two nodes are also possible.

```
class ElectionMessage;  // A class representing an election message. Its definition is not
    needed here.
int getMyPID(); // returns the pid of the current process.
int getBiggerProcessID(int pid);// returns the id of the process in the ring which is bigger
    than the 'pid'.
void addPidToMesssage(ElectionMessage& m, int pid);// adds the 'pid' to the election message.
void sendMessage(ElectionMessage& m, int target); //sends an election message to the process
    identified by the parameter 'target'.
bool waitForReply();// always called after 'sendMessage'. It returns true if the
    acknowledgement is received, otherwise it waits for a while, then it returns false.
```

```
void onReceiveMessage(ElectionMessage m) // Participates in the election in a ring.
{
  int myPID = getMyPID();
  addPidToMessage(m, myPID);
  int biggerPID = getBiggerProcessID(myPID);
  sendMessage(m, biggerPID);
  while(waitForReply() == false)
  {
    biggerPID = getBiggerProcessID(biggerPID);
    sendMessage(m, biggerPID);
  }//end of while loop
}//end of function
```

10

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:	180 minutes	Total Marks:	55
	Paper Date:	14th May, 2019	Weight	45
	Section:	ALL	Page(s):	9
	Exam Type:	Final		

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

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

Question 1: ***[12 marks]***

Please write your answer in the table given below. Answers outside the table will not be marked.

Question no.	Your answer
1.1	a
1.2	b
1.3	b
1.4	F
1.5	T
1.6	F
1.7	F
1.8	a,b
1.9	c,d
1.10	a
1.11	c
1.12	a

1.1 Factors effecting the context-switch time are
- a. Register sets available at the hardware layer
- b. Number of processes in Ready Queue
- c. Format of FCB
- d. Size of physical memory

1.2 Consider calling fork() system call on Unix based operating system, which of the followings will be true after successful execution of fork
- a. Instruction following the fork() system call will be executed in the parent process before the child process gets CPU
- b. Parent and child processes will be running concurrently
- c. Parent and child processes share the data, hence modification to a variable will be visible to both processes
- d. Parent process cannot terminate until all the child processes terminate

1.3 A parent process terminates without calling wait() for its child processes. What possible scenarios can happen?
- a. Zombie Processes count increases
- b. Creation of Orphan processes
- c. Termination of all child processes
- d. Init process invoked to clear all zombie and orphan process

1.4 The OS provides the illusion to each thread that it has its own address space.True/False?

1.5 Threads belonging to the same process can access the same TLB entries.True/false?

1.6 Peterson's algorithm uses the atomic instructions to provide mutual exclusion for two threads.True/False?

1.7 File update operation is considered to be slow because multiple copies of the same FCB are maintained in memory and it takes time to maintain consistency among these duplicates. True/false?

1.8 Which of the following structure(s) must be stored with a file system persistently

a. Boot Control block
b. FCBs
c. System-wide open file table
d. Mount table

1.9 Which of the following structures are maintained in memory for proper file system execution

a. Directory Structure
b. Volume Control Block
c. Mount table
d. Per process open-file table

1.10 Which of the following allocation methods can cause external fragmentation

a. Contiguous Allocation
b. Linked Allocation
c. Indexed Allocation
d. None of the above

1.11 Consider the code snippet given below (assume irrelevant details have been omitted).

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
inti;
for (i = 0; i< 200; i++) {
        balance++;
}
printf("Balance is %d\n", balance);
        return NULL;
}
int main(intargc, char *argv[])
        pthread_t p1, p2, p3;
        pthread_create(&p1, NULL, mythread, "A");
        pthread_join(p1, NULL);
        pthread_create(&p2, NULL, mythread, "B");
        pthread_join(p2, NULL);
        pthread_create(&p3, NULL, mythread, "C");
        pthread_join(p3, NULL);
        printf("Final Balance is %d\n", balance);
}
```

What is the output of the code when thread p2 prints "`Balance is %d\n`"

a. Duetoraceconditions,"balance"mayhavedifferentvaluesondifferentrunsoftheprogram.
b. 200
c. 400
d. 600
e. None of the above

1.12 Consider the code snippet given below(assume irrelevant details have been omitted).

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
inti;
for (i = 0; i< 200; i++) {
        balance++;
}
printf("Balance is %d\n", balance);
        return NULL;
}
int main(intargc, char *argv[])
```

```
        pthread_t p1, p2, p3;
        pthread_create(&p1, NULL, mythread, "A");
        pthread_create(&p2, NULL, mythread, "B");
        pthread_create(&p3, NULL, mythread, "C");
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        pthread_join(p3, NULL);
        printf("Final Balance is %d\n", balance);
}
```

What is the output of the thread p3 when it executes the statement "`Balance is %d\n`"

a. Due to race conditions, "balance" may have different values on different runs of the program.
b. 200
c. 400
d. 600
e. None of the above

Question 2: ***[4 marks]***

Consider the following code snippet that creates multiple processes. Assume that the sleep(x) method puts the process in the waiting queue for x seconds.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid t pid;

pid = fork();
if (pid == 0) { /* Lets call this process 1 */
sleep(5);
}
else {
pid_t pid1;
pid1 = fork();
if(pid1 == 0)
{ /* Lets call this process 2 */
        Sleep(20);
}
else
{
sleep(10);
Wait(Null);
}
return 0;
}
```

a. Which process(es) becomes the Zombie process? __P1_____

   For how much time? _____5 secs_____

b. Which process(es) becomes the Orphan process? ___P2_____

   For how much time? _____ Until Init process calls the wait _____

Question 3: ***[6 marks]***

Consider the following set of processes, with the length of the CPU burst, initial priority and arrival time. A larger priority number implies a higher priority. I/O burst are not required by these processes.

	CPU Burst	Initial Priority	Arrival time
P1	11	2	1
P2	2	1	2
P3	5	2	3
P4	11	3	4
P5	2	3	5

Draw a ***Gantt Chart*** for the above mentioned processes using the following scheduling algorithm.
"Scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 5. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 1 is added to its time quantum, and its priority level is incremented. When a process blocks before using its entire time quantum, its time quantum is reduced by 1, but its priority remains the same."
Use the following table to show your gantt chart. Describe the running process inside the cell while mentioning the scheduling time below it as shown for the P1.

Option 1: Non-Pre-emptive Scheduler (100 % correct solution – deduct 1 mark for using non-pre-emptive while the question asked for pre-emptive solution)

P1	P4	P4	P5	P1	P3	P2	
1	6	11	17	19	25	30	32

Option 2: Non-Pre-emptive Scheduler (P5 scheduled before P4 – deduct 1 or 1.5 marks)

P1	P4	P5	P4	P1	P3	P2	
1	6	11	13	19	25	30	32

Option 3: Pre-emptive Scheduler (100% correct solution – 3 variants)

P1	P4	P4	P5	P1	P1	P3	P2	
1	4	9	15	17	21	25	30	32

P1	P4	P4	P5	P3	P1	P1	P2	
1	4	9	15	17	22	26	30	32

P1	P4	P4	P5	P1	P3	P1	P2	
1	4	9	15	17	21	26	30	32

Option 4: Pre-emptive Scheduler (P5 scheduled before P4 – deduct 1 to 1.5 marks)

P1	P4	P5	P4	P1	P1	P3	P2	
1	4	9	11	17	21	25	30	32

Rough Work:

Question 4: *[6 marks]*

A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary1. When a savage wants toeat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot.

Any number of **savage threads** run the following code:

```
while(true) {
        getServingFromPot();
        eat (); }
```

And one cook thread runs this code:

```
while(true){
        putServingsInPot(M); }
```

The synchronization constraints are:

- Savages cannot invoke getServingFromPot if the pot is empty.
- The cook can invoke putServingsInPot only if the pot is empty.

Puzzle: Add code for the savages and the cook that satisfies the synchronization constraints.

Hint: you need to use the following variables to complete the write-up of this puzzle:

- Servings = 0; //variable to hold the number of servings left
- Mutex = Semaphore(1); // you know what this is for
- emptyPot = Semaphore(0); //indicates that the pot is empty
- fullPot = Semaphore(0); //indicates that the pot is full

Your code goes below:

Cook Thread:	Savages Thread:

while(True){
 	
putServingsInPot(M);
 	
}
 	
	While (true){
	
	GetServingFromPot();
	
	Eat();
	
	}

Dining Savages solution (cook)

```
while True:
    emptyPot.wait()
    putServingsInPot(M)
    fullPot.signal()
```

Dining Savages solution (savage)

```
while True:
    mutex.wait()
        if servings == 0:
            emptyPot.signal()
            fullPot.wait()
            servings = M
        servings -= 1
        getServingFromPot()
    mutex.signal()

    eat()
```

Question 5: ***[6 marks]***

Consider the following code for a simple Stack:

```
class Stack {
private:
        int* a;    // array for stack
        int max; // max size of array
        int top;   // stack top

Semaphore full = 0;
Semaphore empty = MAX;
Semaphore mutex = 1;

public:
        Stack(int m) {
                a = new int[m];
                max = m;
                top = 0;
        }
void push(int x) {
                wait(empty);
                wait(mutex);
                a[top] = x;
                ++top;
                signal(mutex);
                signal(full);
        }
        int pop() {
                wait(full);
```

```
            wait(mutex);
            inttmp = top;
            --top;
            signal(mutex);
            signal(empty);
            return a[tmp];
        }
};
```

You can see from the code that a process does busy waiting if it calls push() when the stack is full, or it calls pop() when the stack is empty.

Consider running the functions push and pop ***concurrently***, ***synchronize*** the code using semaphores with the following requirements

- Eliminate the busy waiting
- Push should not be allowed to put item into stack if the stack if full.
- Pop should not be allowed to pop an item if the stack is empty

Just modify the code clearly in the box above. No need to rewrite.

Question 6: *[8 marks]*

Assume an architecture where every virtual address is of 16-bits and it is translated to a physical address using a 2-level paging hierarchy. Each virtual address has the format:

3-bits	3-bits	10-bits
Page Dir(outer page table) offset	Page Table (inner page table) Offset	Page Offset

Each page-table (inner page-table) entry and page-directory (outer page table) entry is of 12-bits. Both have the same format given below:

6-bits	3-bits	1-bit	1-bit	1-bit
Base address of Page-Table/Page	Reserved	Read	Write	Valid

Read and Write bits, when set, indicate that reading and writing is enabled respectively. If the valid bit is set, it indicates that the page is in physical memory otherwise it is not present. Three bits are reserved and are ignored for all practical purposes.

Note: Give answers in bytes, KB, or MB when asked about memory size.	
a. What is the size of a single page table? **(1 mark)**	12 bits x 2^3 = 12 bytes.
b. What is the size of a single physical page? **(1 mark)**	2^10 = 1 KB
c. What is the maximum amount of virtual memory available to any process? **(1 mark)**	2^16 = 64 KB
d. What is the size of a physical address in number of bits?**(1 mark)**	16 bits
e. What is the maximum physical memory that can be utilized by a user process in this system? **(1 mark)**	64 KB
Using the same architecture, you are given the following state of page directory (outer page table) and page tables (inner page table) for a particular process as shown in Figure 1. . <u>All values are in hex</u>. Addresses increase from bottom to top of each table. **f.** What is the total amount of physical memory currently being used by the process?**(3 mark)**	12 KB

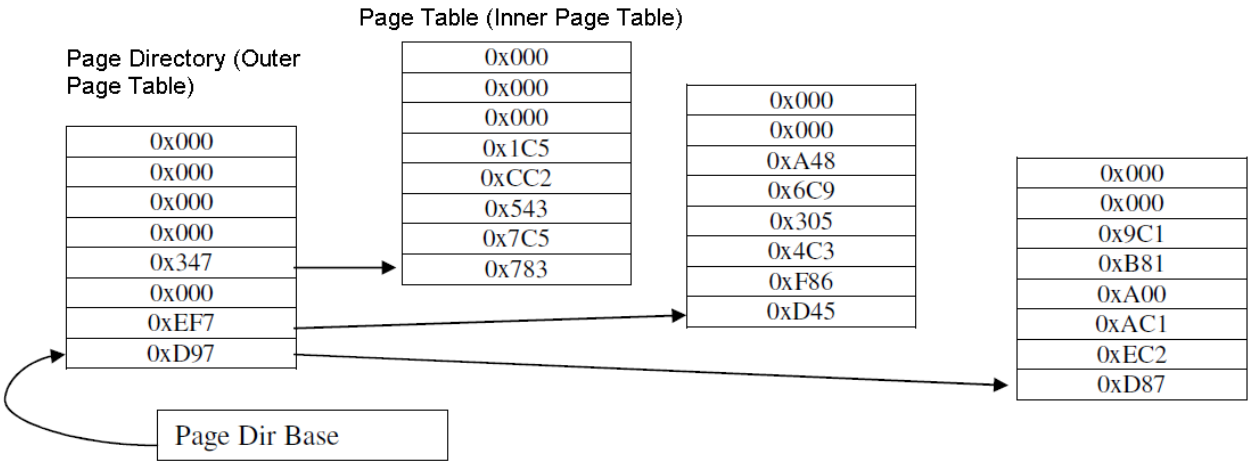


Figure 1: Page Directory (Outer Page Table) and Page Table (Inner Page Table) State for a Process

Question 7: *[7 marks]*

Consider the following code:

```
constint size = 4194304;          //4 MB
char* buf = (char*) malloc(size);       //Allocate 4MB virtual memory
char* end = buf + size;
char* p;
for(p = buf; p < end; p += 8192)
       *p = random();      //Write a random value into the location pointed-to by p
```

The malloc() library call allocates a 4 MB chunk of virtual memory from the heap portion of a process and returns the virtual address of the first byte (which is stored in **buf**). Assume that NONE of this 4MB virtual memory is mapped to physical memory (i.e. as the virtual pages are accessed they are mapped to physical memory and page table entries are setup). Also assume that the OS and hardware are using paging with a page size of 4KB. Assume that you have enough physical memory, so that you never need to replace a page. Please note that a char consists of one byte.

<table>
<tr><td>a. Will the OS be able to run this code without actually using 4MB of physical memory for buf? Answer “yes” or “no”. (1 mark)</td><td colspan="2">Yes</td></tr>
<tr><td>b. How much physical memory (for buf array) will actually be in use once this loop ends? Provide answer in MB’s. (1 mark)</td><td colspan="2">512 * 4 KB = 2 MB</td></tr>
<tr><td>c. How many page faults does the “for loop” cause?(1 mark)</td><td colspan="2">512 page faults</td></tr>
<tr><td rowspan="2">d. Assume now that the page size in your hardware is 8KB instead of 4KB. Write the total page faults and total physical memory used (in MBs) in the right box. (2 marks)</td><td>Total Page Faults</td><td>512</td></tr>
<tr><td>Total Physical Memory Usage (in MB’s)</td><td>4MB</td></tr>
<tr><td rowspan="3">e. Assume that we change the “for loop” in the above code to the following. Fill the table in the right. (2 marks)
for(p = buf + 2048; p < end; p += 4096)
*p = random();</td><td colspan="2">Total Page Faults</td></tr>
<tr><td>Page-size = 4 KB</td><td>1024</td></tr>
<tr><td>Page-size = 8 KB</td><td>512</td></tr>
</table>

Question 8: ***[6 marks]***

Consider a file system on a disk that has both logical and physical block sizes of 1KB(1024 bytes). Assume that the FCBs of each file are already in memory, however, every other relevant data structure must be read from the disk.

a. If we are currently at logical block 8 and want to access logical block 4, how many physical blocks must be read from the disk for each of the following methods?

1. Contiguous Allocation Method
   Answer: _____1_____

2. Linked Allocation method
   Answer: _____4_____

3. Indexed Allocation method
   Answer: _____1_____

b. Consider a file system that used Combined Indexed Allocation method with following details

1. Index table contains 12 direct block addresses
2. Index table contains single, double and triple indirect block addresses
3. Disk blocks are 1 KB in size
4. Pointer to a disk block requires 4 bytes

What is the maximum size of a file that can be stored in this file system? Show your complete calculations below. [Answer should be in Kilobyte units. No need for unit conversions].

Maximum file Size : _____

Calculations:

w = Data stored through direct block addresses: 12 * 1KB

1 block can contain block addresses = 1 KB / 4B = 2 ^ 8 = 256 addresses

x = Data stored through single indirect block addresses:

256 * 1 KB

y = Data stored through double indirect block addresses:

256 * 256 * 1KB

z = Data stored through double indirect block addresses:

256 * 256 * 256 * 1KB

Maximum File size = w + x + y + z

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:	**180 minutes**	**Total Marks:**	**55**
Paper Date:	**14th May, 2019**	**Weight**	**45**
Section:	**ALL**	**Page(s):**	**9**
Exam Type:	**Final**		

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

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

Question 1: ***[12 marks]***

Please write your answer in the table given below. Answers outside the table will not be marked. Note that there can be more than one correct options. *Mention all of them* to be eligible for marks.

Question no.	Your answer
1.1	
1.2	
1.3	
1.4	
1.5	
1.6	
1.7	
1.8	
1.9	
1.10	
1.11	
1.12	

1.1 Factor(s) effecting the context-switch time are
- a. Register sets available at the hardware layer
- b. Number of processes in Ready Queue
- c. Format of FCB
- d. Size of physical memory

1.2 Consider calling fork() system call on Unix based operating system, which of the followings will be true after successful execution of fork
- a. Instruction following the fork() system call will be executed in the parent process before the child process gets CPU
- b. Parent and child processes will be running concurrently
- c. Parent and child processes share the data, hence modification to a variable will be visible to both processes
- d. Parent process cannot terminate until all the child processes terminate

1.3 A parent process terminates without calling wait() for its child processes. What possible scenarios can happen?
- a. Zombie Processes count increases
- b. Creation of Orphan processes
- c. Termination of all child processes
- d. Init process invoked to clear all zombie and orphan process

1.4 The OS provides the illusion to each thread that it has its own address space.True/False?

1.5 Threads belonging to the same process can access the same TLB entries.True/false?

1.6 Peterson's algorithm uses the atomic instructions to provide mutual exclusion for two threads.True/False?

1.7 File update operation is considered to be slow because multiple copies of the same FCB are maintained in memory and it takes time to maintain consistency among these duplicates. True/false?

1.8 Which of the following structure(s) must be stored with a file system persistently

a. Boot Control block
b. FCBs
c. System-wide open file table
d. Mount table

1.9 Which of the following structures are maintained in memory for proper file system execution

a. Directory Structure
b. Volume Control Block
c. Mount table
d. Per process open-file table

1.10 Which of the following allocation methods can cause external fragmentation

a. Contiguous Allocation
b. Linked Allocation
c. Indexed Allocation
d. None of the above

1.11 Consider the code snippet given below (assume irrelevant details have been omitted). Also, ignore compilation errors, if any.

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
inti;
for (i = 0; i< 200; i++) {
        balance++;
}
printf("Balance is %d\n", balance);
        return NULL;
}
int main(intargc, char *argv[])
        pthread_t p1, p2, p3;
        pthread_create(&p1, NULL, mythread, "A");
        pthread_join(p1, NULL);
        pthread_create(&p2, NULL, mythread, "B");
        pthread_join(p2, NULL);
        pthread_create(&p3, NULL, mythread, "C");
        pthread_join(p3, NULL);
        printf("Final Balance is %d\n", balance);
}
```

What is the output of the code when thread p2 prints "Balance is %d\n"

a. Duetoraceconditions,"balance"mayhavedifferentvaluesondifferentrunsoftheprogram.
b. 200
c. 400
d. 600

1.12 Consider the code snippet given below(assume irrelevant details have been omitted). Also, ignore compilation errors, if any.

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
inti;
for (i = 0; i< 200; i++) {
        balance++;
}
printf("Balance is %d\n", balance);
```

```
        return NULL;
}
int main(intargc, char *argv[])
        pthread_t p1, p2, p3;
        pthread_create(&p1, NULL, mythread, "A");
        pthread_create(&p2, NULL, mythread, "B");
        pthread_create(&p3, NULL, mythread, "C");
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        pthread_join(p3, NULL);
        printf("Final Balance is %d\n", balance);
}
```

What is the output of the thread p3 when it executes the statement `"Balance is %d\n"`

a. Due to race conditions, "balance" may have different values on different runs of the program.
b. 200
c. 400
d. 600

Question 2: ***[4 marks]***

Consider the following code snippet that creates multiple processes. Assume that the sleep(x) method puts the process in the waiting queue for x seconds.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
       pid t pid;
       pid = fork();
       if (pid == 0) { /* Lets call this process 1 */
            sleep(5);
       }
       else {
              pid_t pid1;
              pid1 = fork();
              if(pid1 == 0)
              { /* Lets call this process 2 */
                   Sleep(20);
              }
              else
              {
                   sleep(10);
                   Wait(Null);
              }
    }
    return 0;
}
```

a. Which process(es) becomes the Zombie process? _____

   For how much time? _____

b. Which process(es) becomes the Orphan process? _____

   For how much time? _____

Question 3: ***[6 marks]***

Consider the following set of processes, with the length of the CPU burst, initial priority and arrival time. A larger priority number implies a higher priority. I/O burst are not required by these processes.

	CPU Burst	Initial Priority	Arrival time
P1	11	2	1
P2	2	1	2
P3	5	2	3
P4	11	3	4
P5	2	3	5

Draw a ***Gantt Chart*** for the above mentioned processes using the following scheduling algorithm.

"Scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 5. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 1 is added to its time quantum, and its priority level is incremented. When a process blocks before using its entire time quantum, its time quantum is reduced by 1, but its priority remains the same."

Use the following table to show your gantt chart. Describe the running process inside the cell while mentioning the scheduling time below it as shown for the P1.

Please note that the number of cells do not imply the total context switches.

P1									

0

Rough Work:

Question 4: *[6 marks]*

A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot.

Any number of **savage threads** run the following code:

```
while(true) {
        getServingFromPot();
        eat ();
        }
```

And one cook thread runs this code:

```
while(true){
        putServingsInPot(M); }
```

The synchronization constraints are:

- Savages cannot invoke getServingFromPot if the pot is empty.
- The cook can invoke putServingsInPot only if the pot is empty.

Puzzle: Add code for the savages and the cook that satisfies the synchronization constraints.

Hint: you need to initialize and use the following variables to complete the write-up of this puzzle:

- int Servings; //variable to hold the number of servings left
- Semaphore Mutex; // you know what this is for
- Semaphore emptyPot; //indicates that the pot is empty
- Semaphore fullPot; //indicates that the pot is full

Your code goes below:

Cook Thread:	Savages Thread:

while(True){
 	
putServingsInPot(M);
 	
}
 	
	While (true){
	
	GetServingFromPot();
	
	eat ();
	
	}

Question 5: ***[6 marks]***

Consider the following code for a simple Stack:

```
class Stack {
private:
        int* a;    // array for stack
        int max; // max size of array
        int top;   // stack top




public:
        Stack(int m) {
                a = new int[m];
                max = m;
                top = 0;
        }
        void push(int x) {


                while (top == max);   // if stack is full then wait
                a[top] = x;
                ++top;



        }
        int pop() {


                while (top == 0);   // if stack is empty then wait
                inttmp = top;
                --top;



                return a[tmp];
        }
};
```

You can see from the code that a process does busy waiting if it calls push() when the stack is full, or it calls pop() when the stack is empty.

Consider running the functions push and pop ***concurrently***, ***synchronize*** the code using semaphores with the following requirements

- Eliminate the busy waiting
- Push should not be allowed to put item into stack if the stack if full.
- Pop should not be allowed to pop an item if the stack is empty

Just modify the code clearly in the box above. No need to rewrite.

Question 6: *[8 marks]*

Assume an architecture where every virtual address is of 16-bits and it is translated to a physical address using a 2-level paging hierarchy. Each virtual address has the format:

3-bits	3-bits	10-bits
Page Dir(outer page table) offset	Page Table (inner page table) Offset	Page Offset

Each page-table (inner page-table) entry and page-directory (outer page table) entry is of 12-bits. Both have the same format given below:

6-bits	3-bits	1-bit	1-bit	1-bit
Base address of Page-Table/Page	Reserved	Read	Write	Valid

Read and Write bits, when set, indicate that reading and writing is enabled respectively. If the valid bit is set, it indicates that the page is in physical memory otherwise it is not present. Three bits are reserved and are ignored for all practical purposes.

Note: Give answers in bytes, KB, or MB when asked about memory size.	
a. What is the size of a single page table? **(1 mark)**	
b. What is the size of a single physical page? **(1 mark)**	
c. What is the maximum amount of virtual memory available to any process? **(1 mark)**	
d. What is the size of a physical address in number of bits?**(1 mark)**	
e. What is the maximum physical memory that can be utilized by a user process in this system? **(1 mark)**	
Using the same architecture, you are given the following state of page directory (outer page table) and page tables (inner page table) for a particular process as shown in Figure 1. <u>All values are in hex</u>. Addresses increase from bottom to top of each table. **f.** What is the total amount of physical memory currently being used by the process?**(3 mark)**	

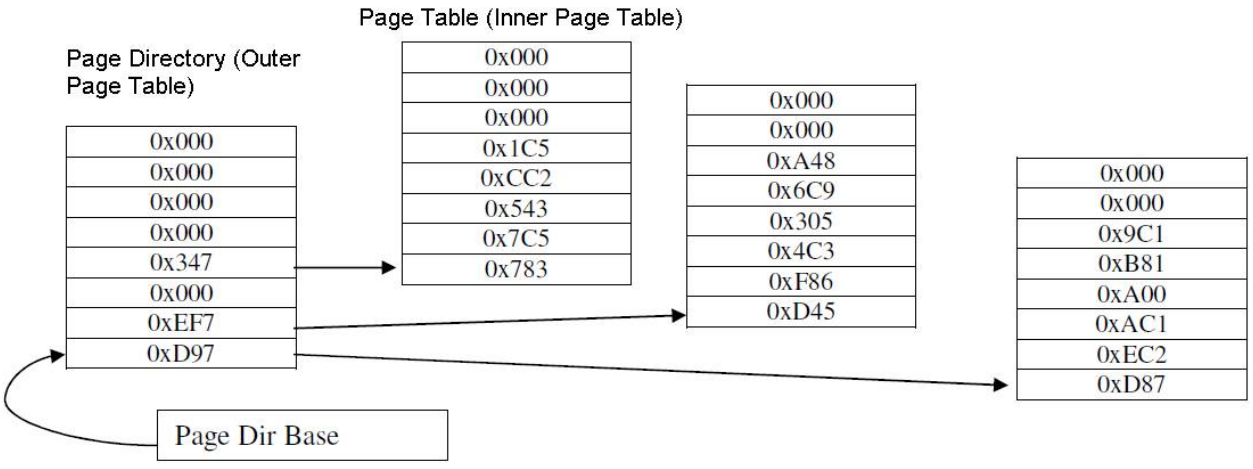


Figure 1: Page Directory (Outer Page Table) and Page Table (Inner Page Table) State for a Process

Question 7: *[7 marks]*

Consider the following code:

```
constint size = 4194304;          //4 MB
char* buf = (char*) malloc(size);      //Allocate 4MB virtual memory
char* end = buf + size;
char* p;
for(p = buf; p < end; p += 8192)
      *p = random();      //Write a random value into the location pointed-to by p
```

The malloc() library call allocates a 4 MB chunk of virtual memory from the heap portion of a process and returns the virtual address of the first byte (which is stored in **buf**). Assume that NONE of this 4MB virtual memory is mapped to physical memory (i.e. as the virtual pages are accessed they are mapped to physical memory and page table entries are setup). Also assume that the OS and hardware are using paging with a page size of 4KB. Assume that you have enough physical memory, so that you never need to replace a page. Please note that a char consists of one byte.

<table>
<tr><td>a. Will the OS be able to run this code without actually using 4MB of physical memory for buf? Answer “yes” or “no”. (1 mark)</td><td colspan="2"></td></tr>
<tr><td>b. How much physical memory (for buf array) will actually be in use once this loop ends? Provide answer in MB’s. (1 mark)</td><td colspan="2"></td></tr>
<tr><td>c. How many page faults does the “for loop” cause?(1 mark)</td><td colspan="2"></td></tr>
<tr><td rowspan="2">d. Assume now that the page size in your hardware is 8KB instead of 4KB. Write the total page faults and total physical memory used (in MBs) in the right box. (2 marks)</td><td>Total Page Faults</td><td></td></tr>
<tr><td>Total Physical Memory Usage (in MB’s)</td><td></td></tr>
<tr><td rowspan="3">e. Assume that we change the “for loop” in the above code to the following. Fill the table in the right. (2 marks)
for(p = buf + 2048; p < end; p += 4096)
*p = random();</td><td colspan="2">Total Page Faults</td></tr>
<tr><td>Page-size = 4 KB</td><td></td></tr>
<tr><td>Page-size = 8 KB</td><td></td></tr>
</table>

Question 8: ***[6 marks]***

Consider a file system on a disk that has both logical and physical block sizes of 1KB(1024 bytes). Assume that the FCBs of each file are already in memory, however, every other relevant data structure must be read from the disk.

a. If we are currently at logical block 8 and want to access logical block 4, how many physical blocks must be read from the disk for each of the following methods?

1. Contiguous Allocation Method
   Answer: _____
2. Linked Allocation method
   Answer: _____
3. Indexed Allocation method
   Answer: _____

b. Consider a file system that used Combined Indexed Allocation method with following details

1. Index table contains 12 direct block addresses
2. Index table contains single, double and triple indirect block addresses
3. Disk blocks are 1 KB in size
4. Pointer to a disk block requires 4 bytes

What is the maximum size of a file that can be stored in this file system? Show your complete calculations below. [Answer should be in Kilobyte units. No need for unit conversions].

Maximum file Size : _____

Calculations:

National University of Computer and Emerging Sciences, Lahore Campus

Course:	Operating Systems	Course Code:	CS 2006
Program:	BS(Computer Science)	Semester:	Spring 2022
Date:	08-June-2022	Total Marks:	70
Exam Type:	Final	Page(s):	9
Time:	180 minutes	Weightage:	45
Sections:	ALL		

Roll Number:_____ **Section:**_____

Instructions:

1. Attempt all questions
2. You may use extra sheets for working, but do not attach them.
3. Write the final answer in the space provided for it.
4. Exchange of calculators or other items is not allowed.

Question	Marks	Marks Obtained	CLO
1	10		CLO1, CLO2, CLO3, CLO4
2	15		CLO1, CLO2, CLO3, CLO4
3	10		CLO3
4	10		CLO4
5	10		CLO2
6	10		CLO3
7	5		CLO4, CLO5

Question 1: Choose the correct option. [10 marks]

1. Which of the following is true for many-to-one thread mapping?
   a. It requires fewer resources in the kernel space.
   b. It does not block other threads of the process if one thread in the same process makes a blocking call.
   c. It can provide multithreading in an operating system that has no native support for multithreading.
   d. All above statements are true
   e. Only a and c are true

2. Which of the following is false about threads?
   a. Threads share instructions.
   b. Threads share stack.
   c. Threads share global variables.
   d. Threads share process id.

3. On a system that has a single CPU, which of the following scenarios may cause the ready queue to be empty?
   a. All processes are I/O bound.
   b. All processes are CPU bound.
   c. 50% of the processes are CPU bound. The remaining processes are I/O bound.
   d. Both a and b

4. Which of the following statements is true?
   a. Keeping time quantum large in round robin scheduling can make the scheduling more like first-come-first-served algorithm.
   b. Keeping the time quantum small will decrease the number of context switches.
   c. Keeping the time quantum large will decrease the number of context switches.
   d. Both a and b
   e. Both a and c

5. Which of the following statements is true?
   a. The size of Logical address space is always equal to the size of physical address space.
   b. The size of logical address space may be larger than the size of physical address space.
   c. The size of logical address space is always less than the size of physical address space.
   d. All statements are false.

6. The time it takes for a process to begin executing for the first time is known as:
   a. turnaround time
   b. waiting time
   c. dispatch latency
   d. response time

7. Which of the following system calls is a blocking call?
   a. read
   b. write
   c. pipe

d. none of these

8. Copy-on-write is a technique in which:
   a. parent and child processes share all pages. Any changes in a page by one process are visible to other process.
   b. parent and child processes don't share any page.
   c. parent and child processes initially share pages. However, any changes in a page cause the page to be copied and placed in separate memory, i.e., the modified page is no longer shared.
   d. parent and child share all pages. Any changes in a page are not allowed.
   e. none of the above

9. A zombie process is a process that has terminated and:
   a. its parent process has also terminated without applying the wait system call.
   b. the parent process is currently executing without applying the wait system call.
   c. It was created without any parent.
   d. none of these.

10. In ordinary pipes, communication between parent and child is possible because:
   a. the pipe's buffer is part of address space of both child and parent
   b. child inherits parent's file descriptor table
   c. the data is read from and written to a file present on hard disk.
   d. none of these

Question 2: Give short and precise answers to following questions: [15 marks]

1. Give two benefits of dynamically linked libraries. [2 + 2= 4 marks]

   a. It reduces the size of executable.
   b. It saves memory space since multiple processes can use the same memory frame(s) where the library code is placed.
   c. It can allow the process to use the latest version of the library.

2. What is the difference between short-term scheduler (CPU-scheduler) and long-term scheduler (job scheduler)? Which one of these two schedulers runs more often? [2 + 1= 3 marks]

   a. Short term scheduler selects one of the processes from the ready queue for execution.
   b. Long term scheduler decides which process to remove from the memory (and hence from the ready queue) to limit the number of processes that are executing concurrently.

   Short term scheduler runs more often.

3. What is meant by bounded wait in process synchronization? [2 marks]

Bounded wait means that a process must not wait infinitely before entering the critical section.

4. What is task parallelism? What is data parallelism? Give one example of both. [1+1+1+1=4 marks]

Task parallelism involves distributing task (threads) across multiple computing cores. Each thread is performing a unique operation. Different threads may be operating on the same data, or they may be operating on different data.

Data parallelism focuses on distributing subsets of the same data across multiple computing cores and performing the same operation (but on a different subset of data) on each core.

Example of Task parallelism: performing addition, subtraction and multiplication (on same data or on different data).

Example of Data Parallelism: Search on first half of the array and on the second half of the array using two threads.

5. Suppose we have an operating system that has two versions of mutex implementation. One implementation allows busy wait whilst the other implementation does not allow busy wait, i.e., the process that could not acquire the lock is put to sleep until the other process releases the lock. Now Under what circumstance, is it desirable to use the version of mutex that allows busy wait? [2 marks]

Mutex with busy wait is desirable in the case if the other process will use the critical section for a very short time.

Question 3: Consider the following processes, their arrival and CPU burst times. [5 + 5= 10 marks]

Process	Arrival Time	CPU Burst Time
P1	0	9
P2	1	7
P3	2	4

a. Draw Gantt chart of the above set of processes by applying shortest-remaining-time-first schedule.

b. What is the waiting time of each process?

P1=0+11=11
P2=0+4=4
P3=0

Question 4: Suppose that a machine has 48-bit virtual address and 32-bit physical address, and the memory is byte-addressable, i.e., each byte can be accessed individually using its physical address. The size of a page is 4KB, i.e. 4*1024 bytes. Now calculate the following: [10 marks]

1. Number of bits required for the page offset. [2 marks]

$log_2(4 * 1024) = 12$

2. Number of bits required for the physical page (also called frame) offset. [2 marks]

Same as page offset.

3. Number of bits required for the page number. [2 marks]

48-12=36

4. Number of bits required for the frame number. [2 marks]

32-12=20

5. Total number of pages that a process can have. [1 mark]

2^36=68719476736

6. The size of physical memory in MBs. Assume 1KB=1024B and 1MB=1024KB. [1 mark]

2^32= 4294967296B =4194304KB= 4096MB

Question 5: There is an executable program named "sort.out" that, whenever executed, asks a list of numbers from the user via the keyboard. The executable program then sorts the list and prints the numbers in sorted order. Your task is to write another program that calls the executable program sort.out in such a way that sort.out gets the list of numbers from a file nums.txt rather than from the user via keyboard. Also, after sorting, the sort.out program will print the numbers to another file sorted_nums.txt rather than on screen. After sort.out has finished execution, your program will print "Task Completed!". Write the program using C/C++ syntax. There is no need to write code to include the required libraries. [10 marks]

```
int main()
{
  int pid = fork();
  if (pid == 0)
  {
    int readFd = open("nums.txt", O_RDONLY, 0);
    int writeFd = open("sorted_nums.txt", O_WRONLY | O_CREAT, 0666);
    dup2(readFd, 0);
    dup2(writeFd, 1);
    close(readFd);
    close(writeFd);
    execlp("sort.out", "sort.out", NULL);
    cout << "execlp error" << endl;
  }
  else if (pid > 0)
  {
     wait(NULL);
     cout << "Task completed!" << endl;
  }
  else
    cout << "fork error" << endl;
}
```

Question 6: Suppose we have a gym in Lahore that allows men as well women. However, to enforce gender separation, the gym has the following policy: When a woman is in the gym, other women may enter the gym, but no man can enter the gym. Similarly, when a man is in the gym, other men may enter the gym, but no woman can enter the gym. Your task is to enforce this rule using semaphores. There must not be any busy wait in your solution. Give your solution in the form of pseudocode.[10 marks]

```
//Create any shared variables here.
semaphore w=1, m=1,entry=1; //binary semaphores
int noOfMen=0, noOfWomen=0;
```

Woman (process)	Man (process)

wait(w);
noOfWomen++;
if (noOfWomen==1)
  wait(entry);
signal(w);
 	
Use Gym() [Critical Section]
 	
wait(w);
noOfWomen--;
if (noOfWomen==0)
  signal(entry);
signal(w);
	wait(m);
	noOfMen++;
	if (noOfMen==1)
	  wait(entry);
	signal(m);
	
	Use Gym() [Critical Section]
	
	wait(m);
	noOfMen--;
	if (noOfMen==0)
	  signal(entry);
	signal(m);

Question 7: Show execution of the optimal page replacement algorithm on the following page reference string: [5 marks]

2 3 1 4 3 5 1 2 1 4

Assume there are only three frames in the RAM. Show contents of memory frames after each page access from the reference string. (Please note that the number of boxes below maybe less or more depending upon the question. It certainly does not mean you have to utilize exactly the given number of boxes.)

2	2	2	4	4	4	4	4	4	4														
	3	3	3	3	5	5	2	2	2														
		1	1	1	1	1	1	1	1														

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/time.h>
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/types.h>

struct param
{
    int* arr1;
    int* arr2;
    int* result;
    int size;
};
void* mergeTwo(void* arg)
{

    struct param* MyArg = (struct param*) arg;
    int i = 0, j = 0 , k = 0;

    for(; i < MyArg->size && j < MyArg->size;)
    {
        if(MyArg->arr1[i] > MyArg->arr2[j])
        {
            MyArg->result[k++] = MyArg->arr2[j++];
        }
        else
        {
             MyArg->result[k++] = MyArg->arr1[i++];
        }
    }
    while(i < MyArg->size)
    {
         MyArg->result[k++] = MyArg->arr1[i++];
    }
    while(j < MyArg->size)
    {
         MyArg->result[k++] = MyArg->arr2[j++];
    }
    pthread_exit(MyArg);
}
void mergeFour(int* arr1, int* arr2, int* arr3, int* arr4, int size)
{
    struct param arguments, arguments2;
    arguments.arr1 = malloc(size * sizeof(int));
    arguments.arr2 = malloc(size * sizeof(int));
    arguments.result = malloc(2 * size * sizeof(int));
    arguments.size = size;

    arguments2.arr1 = malloc(size * sizeof(int));
    arguments2.arr2 = malloc(size * sizeof(int));
    arguments2.result = malloc(2 * size * sizeof(int));
    arguments2.size = size;

    for(int i = 0 ; i < size; i++)
    {
        arguments.arr1[i] = arr1[i];
        arguments.arr2[i] = arr2[i];
    }

    for(int i = 0 ; i < size; i++)
    {
        arguments2.arr1[i] = arr3[i];
```

```
        arguments2.arr2[i] = arr4[i];
    }

    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, mergeTwo, &arguments);  //  merging 1st and 2nd array
    pthread_create(&thread2, NULL, mergeTwo, &arguments2);  //  merging 3rd and 4th array

    struct param* ans;
    struct param* ans2;

    pthread_join(thread1, (void**) &ans);
    pthread_join(thread2, (void**) &ans2);

    free(arguments.arr1); //  we only need 1 struct now for last step
    free(arguments.arr2);

    arguments.arr1 = malloc(size * 2 * sizeof(int));
    arguments.arr2 = malloc(size * 2 * sizeof(int));
    arguments.size = size * 2;

    for(int i = 0 ; i < size * 2; i++)
    {
        arguments.arr1[i] = ans->result[i];
         arguments.arr2[i] = ans2->result[i];

    }

    pthread_create(&thread1, NULL, mergeTwo, &arguments);

    pthread_join(thread1, (void*) ans);

    for(int i = 0 ; i < 20; i++)
    {
        printf("%d ", ans->result[i]);  //  final ans
    }
    printf("\n");


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

    int s = 5;

    int array1[5] = {1,2,4,5,7};
    int array2[5] = {3,6,8,9,10};
    int array3[5] = {21,22,23,24,25};
    int array4[5] = {11,12,13,14,26};

    mergeFour(array1, array2, array3, array4, s);


    return 0;

}
```

National University of Computer and Emerging Sciences, Lahore Campus

Course:	Operating System	Course Code:	CS-205
Program:	BS(Computer Science)	Semester:	Fall 2017
Duration:	3 hour	Total Marks:	50
Paper Date:	27^{th} December, 2017	Weight:	45%
Section:		Page(s):	3
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.

Question 1 (10 points): Although practically it is impossible to implement shortest job first algorithm, but if we had following class implementations, we could easily implement SJF. So lets do it. **Hint:** read the declarations carefully!

```
class List{ // it is the list of all processes ready to run.
public:
        bool addToList(int element); // adds a process described by 'element' to the list.
            Return value is not used here.
        bool removeFromList(int element); // removes a process described by 'element' from the
             list. Return value is not used here.
};
class Iterator{
public:
        Iterator(List list);// initializes the iterator with the list.
        int getNext();  // Used to iterate over the 'List' object, just like an iterator in
            STL. Returns -1 when reaches the end. Othewise returns the PID and moves next.
friend class List;
};
//-------------------------------------------------------------------------------------
double getProcessRemaningTime(int pid) // Practically this function is difficult to implement,
     but here it returns the time the process, described by the 'pid', will take in the next
    burst. Based on the return value of this fucntion we can make decisions.
```

```
int getNextProcessToRun( int leavingProcessID, List list) // First parameter is the process
    which is leaving the CPU. Second the list of ready processes. The function returns the ID
    of the process to run next.
{




}
```

Question 2 (10 points): Implement a function which takes the logical address and returns the physical address. Use the functions provided below.

```
int getPageNumber(int logicalAddress); // takes logical address and returns the associated
    page number.
int getFrameNumber(int pageNumber); // takes the pagenumber and returns the associated frame
    number.
int loadPageInMemory(int pageNumber); // loads a page from backing store into the physical
    memory, and return the framenumber where the page was loaded.
void setFrameNumber(int pagenumber, int framenumber); // sets the framenumber of the
    pagenumber. Also sets all relevant bits of the page table.
int replacePageByFrameNumber(int logicaladdress, int framenumber); // converts the
    logicaladdress into a physical address by replacing the page number by frame number.
```

```
int getPhysicalAddress(int logicalAddress)
{




}
```

Question 3 (10 points): Get the physical byte stored in a file which exists in a file system that uses single indexed table. Parameters are the logical address of the byte, and the file ID.

```
#define BLOCK_SIZE  xxxx; // tells how many bytes are there in one block
int getIndexBlockNumber(int fileID);// takes the file ID and returns the block where index
    table is stored.
int* loadIndexFromBlock(int blockNumber);// takes the block number and loads the index table
    in memory and returns its pointer.
byte* loadBytesFromBlock(int blockNumber); // takes the block number and loads raw bytes in
    that block in memory, and returns its address.
```

```
int getByte(int logicalByteNumber, int fileID)
{




}
```

Question 4 (10 points): Implement the optimal page replacement algorithm using following functions.

```
Class List; // the same class definition given in Question 1
Class Iterator; // the same class definition given in Question 1
//-------------------------------------------------------------------------------------
int getNextOccurence(int pageNumber); // returns the position of next occurence of the '
    pageNumber' in the reference string.
List getPageList(); // returns the list of all pages loaded in the memory.
```

```
int getPageToReplace()// returns the page number of the page which should be evicted from the
    physical memory
{




}
```

Question 5 (6 points): List any three conditions which need to be true for a deadlock to occur.

1.
2.
3.

Question 6 (2 points): In deadlock avoidance algorithms, deadlocks are possible structurally, but we keep a gaurd and do not let all those conditions to be true that can result into a deadlock.

1. True
2. False

Question 7 (2 points): In deadlock prevention algorithms, deadlocks are structurally not possible.

1. True
2. False

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	**Operating Systems**	**Course Code:**	**CS-205**
	Program:	**BS (CS)**	**Semester:**	**Spring 2018**
	Duration:	**Three hours**	**Total Marks:**	**60**
	Paper Date:	**18-May-2018**	**Weight**	
	Section:	**ALL**	**Page(s):**	**7**
	Exam Type:	**Final**		

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

Instruction/Notes: Write your answers on this question paper. Do not attach any answer sheet. However you may use an additional sheet for rough work.

Question 1 (10 marks)

Consider a variation of the Indexed allocation method: The start block has *N* pointers. The first *N-1* pointers point to data blocks, while the last pointer points to another index block. The second index block holds addresses of only data blocks; no more index blocks.

Now write a C/C++ function for address translation in such a variation. Use the following prototype:

```
int translate(int start, int log)
```

The function takes the physical address of the start block, and a logical block number, and computes/returns the corresponding physical block number.

Assume you have a system call to read a disk block into an array:

```
void read(int blockNo, int* array)
```

```
int translate(int start, int log) {
      int a[N];
      read(start, a);
      if (log < N-1)
            return a[log];

      int rem = log - (N-1);      // remaining
      read(a[N-1], a);
      return a[rem];
}
```

Question 2(5+5 marks)

a) Show execution of the LRU page replacement algorithm on the following page reference string:

1 2 3 1 4 5 3 2 1

Assume there are only three frames in the RAM. (Please note that the number of boxes below maybe less or more depending upon the question. It, certainly, does not mean you have to utilize exactly the given number of boxes.)

	1		1		1		1		1		3		3		3	
			2		2		4		4		4		2		2	
					3		3		5		5		5		1	

b) 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:

2 5 3 4 1

Question 3 (5+5)

a) Consider a paging system with a page size of 256. Assume a process running in this system has the following page table:

50	10	90
0	1	2

Now translate the following logical addresses into the corresponding physical addresses:

i) 700 ii) 450

i)	ii)
page = 700 / 256 = 2 frame = tbl[2] = 90 offset = 700 mod 256 = 188 phy add = 90 * 256 + 188 = 23,228	page = 450 / 256 = 1 frame = tbl[1] = 10 offset = 450 mod 256 = 194 phy add = 10 * 256 + 194 = 2,754

b) Consider a system with a memory access time of 100 ns, and a page-fault service time of 7 milliseconds (including the memory access time). Assuming a page fault rate of 10%, compute the effective-access-time. (help: 1 sec = 1000 ms = 10^9 ns)

EAT = 10/100 * 7,000,000 + 90/100 * 100

= 700,090 ns

Question 4 (10 marks)

Consider the following code for a simple Stack:

```
class Stack {
private:
      int* a;    // array for stack
      int max;   // max size of array
      int top;   // stack top
public:
      Stack(int m) {
            a = new int[m];  max = m;   top = 0;
      }
      void push(int x) {
            while (top == max)
                  ;     // if stack is full then wait
            a[top] = x;
            ++top;
      }
      int pop() {
            while (top == 0)
                  ;     // if stack is empty then wait
            int tmp = top;
            --top;
            return a[tmp];
      }
};
```

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.

```
Semaphore full = 0;
Semaphore empty = MAX;
Semaphore mutex = 1;

class Stack {
private:
      int* a;    // array for stack
      int max;   // max size of array
      int top;   // stack top
public:
      Stack(int m) {
            a = new int[m];
            max = m;
            top = 0;
      }
      void push(int x) {
            wait(empty);
            wait(mutex);
            a[top] = x;
            ++top;
            signal(mutex);
            signal(full);
      }
      int pop() {
            wait(full);
            wait(mutex);
            int tmp = top;
            --top;
            signal(mutex);
            signal(empty);
            return a[tmp];
      }
};
```

Question 5 (5+5 points)

a) 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;
}
```

2 3 5 4 6	2 5 3 4 6	2 3 5 6 4	2 5 3 6 4	2 3 4 5 6	2 5 6 3 4

b) Depending upon what technique will be suited best for the requirements given below, write "*shared memory*" or "*message passing*" in front of each of the following phrases:

i. matrix multiplication (shared)

ii. multiple programs running one after another (message)

iii. email (message)

iv. people meeting in a room (shared)

v. the IPC method that requires synchronization (shared)

Question 6 (10 points)
You have to write the code of the round-robin scheduling algorithm. You will implement the function **RoundRobin** whose skeleton is provided below. The function is called whenever a dispatch occurs, and it returns the process to be executed. It has one parameter called **inProc**. It is the pointer to the PCB of the process which was in execution before dispatch.

Below given is the definition of the process control block. Also, you have an object of type **Queue** globally defined as **readyQueue**. It stores the pointers to the Process Control Blocks which are ready to run. Use **readyQueue** and its functions as defined below to implement the function **RoundRobin**.

```
struct PCB{
        int PID; // PID of the process
        int remainingTime;// the number of cycles remaining for the process to complete.
...  // other elements
};

class Queue {
        int enqueu(PCB* proc); // adds an element of type PCB* into the queue
        PCB* dequeu(); // removes an element of type PCB* from the queue
        PCB* operator [] (const int index); // to enable the class to work like an array.
        int size(); //returns the number of elements inside the queue.
};
Queue readyQueue;
--------------------------------------------------------------------------------
PCB* RoundRobin( PCB* inProc) {

      if (inProc->remainingTime > 0)
            readyQueue.enqueu(inProc);

      return readyQueue.dequeu();
```

Operating Systems

Final Exam, Spring 2014

Date: May 16, 2014 **Marks:** 80 **Time: 3 hrs**

Question 1 (10+5+5 marks)

a) Show execution of the LRU Page replacement algorithm on the following page reference string. Assume only three frames are available, and these are initially empty. If the LRU algorithm is unable to guide you at some point then use FIFO.

1 2 3 1 4 5 6 7 5 8

This reference string says that a running process first needs page 1, then page 2, and so on.

b) Consider a paging system with a memory access time of 200 nano-seconds, and a TLB hit ratio of 90%. Calculate the effective access time.

c) Write down the steps in handling a page fault.

Question 2 (5+5+5+5)

a) How many processes would be created by the following code?

```
for (int i = 1; i <= 5; ++ i)
        fork();
```

b) Give all possible outputs of the following program:

```
int x = 0; // global
```

// Thread 1	// Thread 2	// Thread 3
x = x + 2;	x = x + 3;	cout << x;

c) Name the four necessary conditions for a deadlock.

d) Which of the following scheduling algorithms may suffer from starvation?

i. FIFO
ii. RR
iii. Priority Scheduling
iv. SJF
v. Multi-level Queue Scheduling

Question 3 (10+10 marks)

a) Consider a file stored in the following FAT with starting block 5:

3	6	0	7	8	9	2	4	eof	1
0	1	2	3	4	5	6	7	8	9

Now translate the following logical block numbers of the file into the physical block numbers:

(i) 0 (ii) 2 (iii) 5 (iv) 7 (v) 9

b) Consider a file system using Two-level Index with a block size of 4KB and a pointer size of 4 bytes. Write down a C/C++ function to compute the physical block number for a given logical block number of a particular file. Following is the function prototype:

```
int translate( int start, int logBlk )
```

The first parameter here points to the file index block and the second is the required logical block number. The function returns the corresponding physical block number after computation.

You may use the following system call to read a block: read(int blk, int* a). The first parameter here is the required physical block number and the second parameter is a pointer to the target array/buffer.

Question 4 (20 marks)

Consider a nursery where children eat sweets from a large pot. When the pot is empty the next interested child calls the chef, and the chef fills the pot again. This scenario can be modeled in a multi-threading environment as follows:

```
// child                    // chef
while (true)                while (true)
    getSweetFromPot             relax ...
    eat ...                     putSweetInPot
```

You need to synchronize the children and the chef using Semaphores. The "child" code is executed by multiple threads; on the other hand The "chef" code is

executed by a single thread. Assume the chef places N sweets in the pot at a time, and each child takes only a single sweet at one time.

Roll No_____ Fall 2012 Section_____

Operating Systems- (A, B, C)
FINAL-FALL 2012

Total Marks: **70**

Total

Time: 3 hours

Q# 1: [5+5+5+5 Marks]
Give Short answers to the following questions:

a) What kind of fragmentation is introduced in
- I) Paged Architecture
- II) Segmented Architecture

How are they removed in "Hierarchical page" ("Paged Segment" or Multilevel Paged) architecture. Explain the architecture for a 32 bit system.

b) A number of web browsers open each tab in a separate process, even though each tab essentially runs the same code. This adds to the bulk of the application; increases memory utilization and may slow down the machine. Can browsers use threads instead of processes? Explain?

c) Mr Herry and David both write code to perform the same functionality. Mr Hennry had taken the OS course with you. Now they both run their codes on different hardware systems with different OSes and do a benchmarking analysis of Page faults. Mr. David's program is producing 50% more page faults in comparison to you. Based on your knowledge of OS you can tell him why is that so?

Solution

```
1) Too small physical memory.
2) Page replace policy of particular OS is not good.
3) Code is haphazard.
```

Q# 2: [10 Marks]

—-----

Solution:

```
e.a. = 1 us + (0.20 * 1 us) + (0.02 * 20,000 us)
     = 401.2 us
```

Or, if you prefer

```
e.a. = (0.80 * 1 us) + (0.18 * 2 us) + (0.02 * 20,002 us)
     = .8 us + .36 us + 400.04 us
     = 401.2 us
```

Q# 3: [12 Marks]
Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of A:10, B:6, C:2, D:4, and E:8 minutes. Their (externally determined) priorities are A:3, B:5, C:2, D:1, and E:4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, draw the

gantt chart and determine the average waiting time. Ignore context switching overhead. For partial credit, you should list the finishing times:

i) Round-Robin (so assume a time slice of 1 unit time)
ii) Preemptive Priority scheduling.
iii) First-come, First-served
iv) Shortest Job First.

(a)Round Robin. Preemptive multiprogramming. Each job gets its fair share of CPU.

Initially all five processes are running, so they each get 1/5 of the CPU, which means they take 5 times longer to run. This means that it will take 10 mins for C (the shortest job) to complete.

After 10 mins each job will have used 2 mins of CPU time each, so A will have 8 mins left, B 4, D 2 and E will have 6. Since there are now 4 jobs they will each get 1/4 the CPU time, which means they will take four times as long to run. So, it will take 8 mins for D to complete. (So, D took 10 + 8 = 18 mins).

Now, A will have 6 mins left, B 2, and E 4.
Each process gets 1/3rd of the CPU. Hence B will complete execution in 6 minutes. (B took 10 + 8+ 6= 24 mins)

Now A will have 4 mins and E 2mins left. Each process gets 1/2 of the CPU, so E will finish executing in 4 mins. (E took 24 + 4= 28 mins)

Since A has only 2 mins left to run and it is the only job on the CPU, it will finish in to minutes with a total time of (28 + 2)=30 mins.

Averaging we get (10+18+24+28+30)/5 = 22 mins average turnaround.

(b)
Priority. Each job runs to completion without being preempted.

Since B has the highest priority it will run first, followed by E, followed by A, followed by C and finally D. There is no preemption, so B completes after 6 mins, E after 6 mins of waiting for B and 8 mins of processing (14 mins total), A after 14 mins of waiting for B and E, and 10 mins processing (24 mins total), C after 24 mins of waiting for B,E and A, and 2 mins processing (26 mins total)and D after 26 mins of waiting for B,E,A & C and 4 mins of processing (30 mins total). The average is (6+14+24+26+30)/5 = 100/5 = 20.

(c)

First Come First Served. Processes run to completion in the order of arrival.

A completes after 10 mins, B completes after (10 +6)=16 mins, C completes after (16 + 2)=18 mins, D completes after (18+4)=22 mins, and E completes after (22+8)=30 mins.

The average is : (10+16+18+22+30)/5 = 96/5 = 19.2

(d)

Shortest job first. Processes will run to completion in the order : C, D, B, E, A

C completes after 2 mins, D completes after (2+4)=6 mins, B completes after (6+6)=12 mins, E completes after (12+8)=20 mins, and A completes after (20+10)=30 mins.
The average is : (2+6+12+20+30)/5 = 70/5 = 14 mins

Q# 4: [8 Marks]

Five jobs are waiting to be run. Their expected running times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? State the scheduling algorithm that should be used AND the order in which the jobs should be run. (Your answer will depend on X).

Solution:

Shortest job first is the way to minimize average response time (we also accepted Shorted Remaining Time to Completion First).
0 < X <= 3: X, 3, 5, 6, 9.
3 < X <= 5: 3, X, 5, 6, 9.
5 < X <= 6: 3, 5, X, 6, 9.
6 < X <= 9: 3, 5, 6, X, 9.
X> 9: X, 3, 5, 6, 9, X.
.

Q# 5: [3+3+4 Marks]

A computer system has a 36-bit virtual address space with a page size of 8K, and 4 bytes per page table entry.
a) How many pages are in virtual address space?
b) What is the maximum size of addressable physical memory in this system?
c) If the average process size is 8GB, would you use a one level, two level or three level page table? Why?

Solution:

a. A 36 bit address can address 2^36 bytes in a byte addressable machine. Since the size of a page 8K bytes (2^13), the number of addressable pages is 2^36 / >2^13 = 2^23
b. With 4 byte entries in the page table we can reference 2^32 pages. Since each page is 2^13 B long, the maximum addressable physical memory size is 2^32 * 2^13 = 2^45 B (assuming no protection bits are used).
c. 8 GB = 2^33 B We need to analyze memory and time requirements of paging schemes in order to make a decision. Average process size is considered in the calculations below.

1 Level Paging

Since we have 2^23 pages in each virtual address space, and we use 4 bytes per page table entry, the size of the page table will be 2^23 * 2^2 = 2^25. This is 1/256 of the process' own memory space, so it is quite costly. (32 MB)

2 Level Paging

The address would be divided up as 12 | 11 | 13 since we want page table pages to fit into one page and we also want to divide the bits roughly equally.

Since the process' size is 8GB = 2^33 B, I assume what this means is that the total size of all the distinct pages that the process accesses is 2^33 B.

Hence, this process accesses 2^33 / 2^13 = 2^20 pages. The bottom level of the page table then holds 2^20 references. We know the size of each bottom level chunk of the page table is 2^11 entries. So we need 2^20 / 2^11 = 2^9 of those bottom level chunks.

The total size of the page table is then:

//size of the outer page table	//total size of the inner pages		
1 * 2^12 * 4	+ 2^9 * 2^11 * 4	=	2^20 * (2^-6 + 4) ~4MB

3 Level Paging

For 3 level paging we can divide up the address as follows:
8 | 8 | 7 | 13

Again using the same reasoning as above we need 2^20/2^7 = 2^13 level 3 page table chunks. Each level 2 page table chunk references 2^8 level 3 page table chunks. So we need 2^13/2^8 = 2^5 level-2 tables. And, of course, one level-1 table.

The total size of the page table is then:

//size of the outer page table	//total size of the level 2 tables	//total size of innermost tables	
1 * 2^8 * 4	2^5 * 2^8 *4	2^13 * 2^7 * 4	~4MB

As easily seen, 2-level and 3-level paging require much less space then level 1 paging scheme. And since our address space is not large enough, 3-level paging does not perform any better than 2 level paging. Due to the cost of memory accesses, choosing a 2 level paging scheme for this process is much more logical.

Q# 6: [10 Marks]

Suppose we are building printed circuit boards (PCBs) in a factory. This involves two steps: the first is to mount electrical components on a board and the second is to solder them. There are a number of Mount and Solder processes in our system. These processes can arrive in any order and their execution may be arbitrarily interleaved. The role of the Mount process is to carry out the mounting on a single board and it is followed by a Solder process. The Mount and Solder processes that work on the same board should only exit after the soldering of that board is complete. You are required to give the pseudo code for the Mount and Solder processes using semaphores. Make sure that you clearly represent the initialization values of your semaphores.

Variable initialization*:*

```
Semaphore mount = 0;

Semaphore board_complete = 0;
```

Mount Process

```
// start mounting

signal(mount);

wait(board_complete);
```

Solder Process

```
wait(mount);

//do soldering

signal(board_complete);
```

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	**Operating Systems**	**Course Code:**	**CS-205**
	Program:	**BS (CS)**	**Semester:**	**Spring 2018**
	Duration:	**Three hours**	**Total Marks:**	**60**
	Paper Date:	**18-May-2018**	**Weight**	
	Section:	**ALL**	**Page(s):**	**7**
	Exam Type:	**Final**		

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

Instruction/Notes: Write your answers on this question paper. Do not attach any answer sheet. However you may use an additional sheet for rough work.

Question 1 (10 marks)

Consider a variation of the Indexed allocation method: The start block has *N* pointers. The first *N-1* pointers point to data blocks, while the last pointer points to another index block. The second index block holds addresses of only data blocks; no more index blocks.

Now write a C/C++ function for address translation in such a variation. Use the following prototype:

```
int translate(int start, int log)
```

The function takes the physical address of the start block, and a logical block number, and computes/returns the corresponding physical block number.

Assume you have a system call to read a disk block into an array:

```
void read(int blockNo, int* array)
```

```
int translate(int start, int log) {
     int a[N];
     read(start, a);
     if (log < N-1)
          return a[log];

     int rem = log - (N-1);     // remaining
     read(a[N-1], a);
     return a[rem];
}
```

Question 2(5+5 marks)

a) Show execution of the LRU page replacement algorithm on the following page reference string:

1 2 3 1 4 5 3 2 1

Assume there are only three frames in the RAM. (Please note that the number of boxes below maybe less or more depending upon the question. It, certainly, does not mean you have to utilize exactly the given number of boxes.)

	1		1		1		1		1		3		3		3	
			2		2		4		4		4		2		2	
					3		3		5		5		5		1	

b) 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:

2 5 3 4 1

Question 3 (5+5)

a) Consider a paging system with a page size of 256. Assume a process running in this system has the following page table:

50	10	90
0	1	2

Now translate the following logical addresses into the corresponding physical addresses:

i) 700 ii) 450

i)	ii)
page = 700 / 256 = 2 frame = tbl[2] = 90 offset = 700 mod 256 = 188 phy add = 90 * 256 + 188 = 23,228	page = 450 / 256 = 1 frame = tbl[1] = 10 offset = 450 mod 256 = 194 phy add = 10 * 256 + 194 = 2,754

b) Consider a system with a memory access time of 100 ns, and a page-fault service time of 7 milliseconds (including the memory access time). Assuming a page fault rate of 10%, compute the effective-access-time. (help: 1 sec = 1000 ms = 10^9 ns)

EAT = 10/100 * 7,000,000 + 90/100 * 100

= 700,090 ns

Question 4 (10 marks)

Consider the following code for a simple Stack:

```
class Stack {
private:
      int* a;    // array for stack
      int max;   // max size of array
      int top;   // stack top
public:
      Stack(int m) {
            a = new int[m];  max = m;   top = 0;
      }
      void push(int x) {
            while (top == max)
                  ;    // if stack is full then wait
            a[top] = x;
            ++top;
      }
      int pop() {
            while (top == 0)
                  ;    // if stack is empty then wait
            int tmp = top;
            --top;
            return a[tmp];
      }
};
```

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.

```
Semaphore full = 0;
Semaphore empty = MAX;
Semaphore mutex = 1;

class Stack {
private:
      int* a;    // array for stack
      int max;   // max size of array
      int top;   // stack top
public:
      Stack(int m) {
            a = new int[m];
            max = m;
            top = 0;
      }
      void push(int x) {
            wait(empty);
            wait(mutex);
            a[top] = x;
            ++top;
            signal(mutex);
            signal(full);
      }
      int pop() {
            wait(full);
            wait(mutex);
            int tmp = top;
            --top;
            signal(mutex);
            signal(empty);
            return a[tmp];
      }
};
```

Question 5 (5+5 points)

a) 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;
}
```

2 3 5 4 6	2 5 3 4 6	2 3 5 6 4	2 5 3 6 4	2 3 4 5 6	2 5 6 3 4

b) Depending upon what technique will be suited best for the requirements given below, write "*shared memory*" or "*message passing*" in front of each of the following phrases:

i. matrix multiplication (shared)

ii. multiple programs running one after another (message)

iii. email (message)

iv. people meeting in a room (shared)

v. the IPC method that requires synchronization (shared)

Question 6 (10 points)
You have to write the code of the round-robin scheduling algorithm. You will implement the function **RoundRobin** whose skeleton is provided below. The function is called whenever a dispatch occurs, and it returns the process to be executed. It has one parameter called **inProc**. It is the pointer to the PCB of the process which was in execution before dispatch.

Below given is the definition of the process control block. Also, you have an object of type **Queue** globally defined as **readyQueue**. It stores the pointers to the Process Control Blocks which are ready to run. Use **readyQueue** and its functions as defined below to implement the function **RoundRobin**.

```
struct PCB{
        int PID; // PID of the process
        int remainingTime;// the number of cycles remaining for the process to complete.
...  // other elements
};

class Queue {
        int enqueu(PCB* proc); // adds an element of type PCB* into the queue
        PCB* dequeu(); // removes an element of type PCB* from the queue
        PCB* operator [] (const int index); // to enable the class to work like an array.
        int size(); //returns the number of elements inside the queue.
};
Queue readyQueue;
--------------------------------------------------------------------------------
PCB* RoundRobin( PCB* inProc) {

      if (inProc->remainingTime > 0)
            readyQueue.enqueu(inProc);

      return readyQueue.dequeu();
```