National University of Computer and Emerging Sciences
Chiniot-Faisalabad Campus

Operating System CS2006

Final Term Exam

Date: December 16th 2024

Course Instructor(s)

M. Haseeb Arshad, Mahzaib Younas, Aysha Liaqat, Rukhsana Zafar, Juhina Batool, Tahir Farooq

Total Time : 160 Min
Total Marks: 90
Total Questions: 8

_____ _____ _____
Roll No Section Student Signature

Vetted by: _____ **Signature:**_____.

Subjective

Subjective part is required to be distributed after collecting objective part.

CLO 4: Understand the dead locks and memory management **[8 + 5 = 13 Marks]**

Q2.

a. Calculate the internal fragmentation if the page size is 4 KB and the process size is 70,000 bytes

[1 mark for conversion + 2 mark for no of pages + 2 marks for fragmentation]

Page Size = 4KB = 4 x 1024 = 4096 bytes
Process size = 70,000 bytes

$$\text{No of pages} = \frac{70{,}000}{4096} = 17.08 \text{ s}$$

Internal Fragmentation

= 4096/100 = 40.96
=40.96 x .09 = 3.68
=3.6864 x 100 = 368.64

Total Internal Fragmentation

= 4096 – 368.64
= 3727.27 = 3728

b. Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?

2.5 mark for each algorithm(2.5 x 3 = 7.5) + 0.5 for efficiency]

First Fit

- Process 212K: Fits into the 500K partition.
  - Memory Partitions after allocation: 100K, 288K (500K-212K), 200K, 300K, 600K
- Process 417K: Fits into the 600K partition.
  - Memory Partitions after allocation: 100K, 288K, 200K, 300K, 174K (600K-417K)
- Process 112K: Fits into the 288K partition.
  - Memory Partitions after allocation: 100K, 176K (288K-112K), 200K, 300K, 174K
- Process 426K: Does not fit into any remaining partition.

Best Fit

- Process 212K: Fits into the 300K partition (best fit).
  - Memory Partitions after allocation: 100K, 500K, 200K, 88K (300K-212K), 600K
- Process 417K: Fits into the 500K partition (best fit).
  - Memory Partitions after allocation: 100K, 83K (500K-417K), 200K, 88K, 600K
- Process 112K: Fits into the 200K partition (best fit).
  - Memory Partitions after allocation: 100K, 83K, 88K (200K-112K), 88K, 600K
- Process 426K: Fits into the 600K partition.
  - Memory Partitions after allocation: 100K, 83K, 88K, 88K, 174K (600K-426K)

Worst Fit

- Process 212K: Fits into the 600K partition (worst fit).
  - Memory Partitions after allocation: 100K, 500K, 200K, 300K, 388K (600K-212K)
- Process 417K: Fits into the 500K partition (worst fit).
  - Memory Partitions after allocation: 100K, 83K (500K-417K), 200K, 300K, 388K
- Process 112K: Fits into the 388K partition (worst fit).
  - Memory Partitions after allocation: 100K, 83K, 200K, 300K, 276K (388K-112K)
- Process 426K: Does not fit into any remaining partition.

CLO 4: Understand the dead locks and memory management

Q 3: **[5 + 2 = 7 Marks]**

a. Explain the purpose of TLB. To find the effective memory-access time, we weight the case by its probability with 68 percent hit ratio if the access time of hit ratio is 20s and miss ratio is 1 min.

[2 mark for TLB + 3 mark for EAT]

The primary function of the TLB is to speed up the translation of virtual addresses to physical addresses. Without the TLB, every memory access would require a time-consuming page table lookup, which could significantly slow down system performance.

TLB hit ratio (HH) = 68% = 0.68
TLB miss ratio (MM) = 32% = 0.32
TLB hit time = 20 seconds
TLB miss time = 60 seconds
EAT

$$EAT=(H\times \text{Hit time})+(M\times \text{Miss time})$$
$$= (0.68\times 20\ s)+(0.32\times 60\ s)$$
$$= 13.6 + 19.2$$
$$= 32.8s$$

b. Explain the concept of Compaction, also write its benefits. **[2 Marks]**

[1 Mark for definition + 1 mark for benefits]

Compaction is a technique to collect all the free memory present in the form of fragments into one large chunk of free memory, which can be used to run other processes.

- Reduces external fragmentation.
- Make memory usage efficient.
- Memory become contigous

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

Q 4: **[5 + 5 = 10 Marks]**

a. **In a counting semaphore there are several processes namely P0, P1, P2, P3, P4, and P5 want to enter the critical section. If the semaphore value is set to 4, then suggest what could occur in this scenario. [Note: must mention the block list and awake list]**

[3 Marks for CS and block list + 2 Marks for awake list condition]

- Processes P0, P1, P2, and P3 enter the critical section first, as the semaphore value allows up to 4 concurrent processes.
- Semaphore value decreases by 1 for each process entering:
  - After P0 enters: Semaphore value = 3
  - After P1 enters: Semaphore value = 2
  - After P2 enters: Semaphore value = 1
  - After P3 enters: Semaphore value = 0
- Processes P4 and P5 have to wait since the semaphore value is now 0. It will go to block list.
- **Processes Exiting the Critical Section:**
  - As processes P0, P1, P2, and P3 finish their tasks and exit the critical section, they signal the semaphore, incrementing its value by 1 each time.
  - When P0 exits: Semaphore value = 1 (now P4 can enter)
  - When P1 exits: Semaphore value = 2 (now P5 can enter)
  - When P2 and P3 exit: Semaphore value increases further, allowing any waiting processes to enter based on availability.
- **Remaining Processes:**
  - P4 enters the critical section after P0 or P1 exits.
  - P5 enters the critical section after P1 or P2 exits

a. **Consider a multi-threaded program with two threads T1 and T2. The threads share two semaphores: s1 (initialized to 1) and s2 (initialized to 0). The threads also share a global variable x (initialized to 0). Write the possible outcomes when threads T1 and T2 execute concurrently? The threads execute the code shown below.**

Code for T1	Code for T2

wait(s1);
x = x+1;
print(x);
wait(s2);
signal(s1);
	wait(s1);
	x = x+1;
	print(x);
	signal(s2);
	signal(s1);

[2.5 mark for each condition]

initially S1=1 & both the threads are performing **wait** operation on S1, So, only one thread can succeed & run.If thread T1 runs first, then T1 gets blocked when it executes **wait(S2)**.
Also, T2 cannot run as it would get blocked when it executes wait(S1). So, deadlock occurs.

A. runs first and prints 1, T1 runs next and prints 2

B. T1 runs first and prints 1,T 2 does not print anything (deadlock)

A. T2 runs first and prints 1, T1 runs next and prints 2

Outcome 1 with code explanation

Code for T1	Code for T2
wait(s1); s1 = 0 x = x+1; x = 1+1 = 2 print(x); 2 wait(s2); s2 =0 signal(s1); s1= 1	wait(s1); s1 = 0 x = x+1; x = 0 + 1 print(x); 1 signal(s2); s2 = 1 signal(s1); s1 = 1

B. T1 runs first and prints 1,T 2 does not print anything (deadlock)

Outcome 2 with code explanation

Code for T1	Code for T2

wait(s1); s1 = 0
x = x+1; x = 0+1 = 1
print(x); 1
wait(s2); s2 =-1
signal(s1);
	wait(s1);
	x = x+1;
	print(x);
	signal(s2);
	signal(s1);

CLO 3: **Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc.** **[5+4+6=15Marks]**

Q 5:

a. **What is semaphore? How semaphore works for synchronizations of process?**

Semaphores are integer variables that are used to **solve critical section problems** by using **two atomic operations, wait and signal** that are used for process synchronization. **[mark 1]**

Wait Operation: [Marks 1.5]

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
Wait(S)
{
While(S<=0);
S--;
}
```

Signal Operation: [Marks 1.5]

The signal operation increments the value of its argument S.

```
Signal(S)
{
S++;
}
```

There are two types of Semaphores i.e., **binary and counting semaphores. Their explanation…….. [mark 1]**

b. Explain when a deadlock can and when it cannot occur in the below scenario when two processes (P0, P1) are competing for Semaphores S=1 and Q=1,

P(0)	**P(1)**
Wait(S)	Wait(Q)
Wait(Q)	Wait(S)
.	.
.	.
Signal(S)	Signal(Q)
Signal(Q)	Signal(S)

Deadlock will occur if both processes run concurrently. If **P(0)** executes the wait operation on semaphore **S**, the value of **S** will become 0. At the same time, if **P(1)** executes the wait operation on semaphore **Q**, the value of **Q** will also become 0. After this, if **P(0)** calls the wait operation on semaphore **Q**, it will get blocked since **Q** is held by **P(1)**. Similarly, if **P(1)** calls the wait operation on semaphore **S**, it will get blocked because **S** is held by **P(0**. This circular wait condition results in a deadlock**. [Marks 2]**

However, deadlock will not occur if the processes execute in the following order:

- If **P(0)** executes first, it will complete its operations and release the resources (semaphores **S** and **Q**) by performing the signal operation on **S** and **Q**. These resources will then be available for **P(1)** to consume.
- Similarly, if **P(1)** executes first, it will release the resources (semaphores **S** and **Q**) by performing the signal operation on **S** and **Q**, which will then be consumed by **P(0)**. **[Marks 2]**

Thus, proper sequencing of process execution ensures that deadlock is avoided.

c. How Peterson's solution preserves three properties for critical section problems i.e., Mutual exclusion, bounded-waiting, and progress.

To prove property 1, we note that each Pi enters its critical section only if either flag[j] == false or turn == i. Also note that, if both processes can be executing in their critical sections at the same time, then flag[0] == flag[1] == true. These two observations imply that P0 and P1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1 but cannot be both. Hence, one of the processes—say, Pj—must have successfully executed the while statement, whereas Pi had to execute at least one additional statement ("turn == j"). However, at that time, flag[j] == true and turn == j, and this condition will persist as long as Pj is in its critical section; as a result, mutual exclusion is preserved. **[Marks 2]**

To prove properties 2 and 3, we note that a process Pi can be prevented from entering the critical section only if it is stuck in the while loop with the condition flag[j] == true and turn == j; this loop is the only one possible. If Pj is not ready to enter the critical section, then flag[j] == false, and Pi can enter its critical section. If Pj has set flag[j] to true and is also executing in its while statement, then either turn == i or turn == j. If turn == i, then Pi will enter the critical section. If turn == j, then Pj will enter the critical section. However, once Pj exits its critical section, it will reset flag[j] to false, allowing Pi to enter its critical section. If Pj resets flag[j] to true, it must also set turn to i. Thus, since Pi does not change the value of the variable turn while executing the while statement, Pi will enter the critical section **(progress)** after at most one entry by Pj **(bounded waiting). [Marks 4]**

CLO 4: Understand the dead locks and memory management **[2+6+2=10 Marks]**

Q 6:

A system contains 4 processes (P1, P2, P3, P4) and 3 resource types (A, B, C).
The available instances of each resource type are **A = 3, B = 2, C = 2**. The Maximum resource demand and the Allocated resources for each process are:

Process	Maximum (A, B, C)	Allocated (A, B, C)	Need
P1	(7, 5, 3)	(0, 1, 0)	(7-0, 5-1, 3-0) = (7, 4, 3)
P2	(3, 2, 2)	(2, 0, 0)	(3-2, 2-0, 2-0) = (1, 2, 2)
P3	(9, 0, 2)	(3, 0, 2)	(9-3, 0-0, 2-2) = (6, 0, 0)
P4	(4, 3, 3)	(2, 1, 1)	(4-2, 3-1, 3-1) = (2, 2, 2)

a. Calculate the Need Matrix for all processes using the formula (fill 4th column in the table)

b. Using the Banker's Algorithm, determine whether the system is in a safe state. Provide the step-by-step sequence of processes, if a safe sequence exists.

Initial Available Resources: 2+ 0.5*4= 6

Available = Total Instances - Allocated Total

$$\text{Available}=(3,2,2)-(0+2+3+2,1+0+0+1,0+0+2+1)=(3-7,2-2,2-3)=(0,0,0)$$

Step-by-Step Execution:

1. **Check P1**:
   Need(P1) = (7, 4, 3).
   Available = (0, 0, 0).
   Since Available < Need(P1), P1 **cannot execute**.
2. **Check P2**:
   Need(P2) = (1, 2, 2).
   Available = (0, 0, 0).
   Since Available < Need(P2), P2 **cannot execute**.
3. **Check P3**:
   Need(P3) = (6, 0, 0).
   Available = (0, 0, 0).
   Since Available < Need(P3), P3 **cannot execute**.
4. **Check P4**:
   Need(P4) = (2, 2, 2).
   Available = (0, 0, 0).
   Since Available < Need(P4), P4 **cannot execute**.

c. If the system is not in a safe state, explain why, including any limitations in resource availability.

The system is **not in a safe state**, as no process can execute due to insufficient resources.
Reason for Unsafety: The available resources are (0, 0, 0), which are insufficient to fulfill the "Need" of any process.

CLO 4: Understand the dead locks and memory management **[1+3+1=5 Marks]**

Q 7:

Consider a system with 4 processes (P1, P2, P3, P4) and 3 resource types (R1, R2, R3).
a . Provide a Resource allocation graph
b. Determine whether a deadlock exists in the system by analyzing the RAG.
c. Suggest one approach to break the deadlock, if it exists.

Part a: RAG shows a cycle between

Part b: Analysis: 6*0.5=3

1. Available resources(0,2,1)
2. P2: Holds R2, Requests R3 > it will be executed as 1 resource of R3 is free > after execution it will release R2 then Available resources are (0,3,1)
3. P3: Holds R1, Requests R2 > 1 instance of R2 will be give to and it will complete its execution > available(1,3,1)
4. P4: Holds R3, Requests R2 > instance of R2 will be give to and it will complete its execution > available(1,3,2)
5. Now we have ample resources to execute the P1
6. Thus no deadlock exits

Part c: Suggested approaches to resolve deadlock: (any 1)

- Preemption
- Terminate
- Add more resources.

CLO 2: Implement solutions Employing concepts of Processes and Threads **[5+5+5=15 Marks]**

Q 8:

a. **When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process? Just write name and a reason of one to two lines.**

a. Stack
b. Heap
c. Shared memory segments
Answer:

Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

b. Using Amdahl's Law, calculate the speedup gain of an application that has a 60 percent parallel component for two and four processing cores.

Two processing cores	**Four Processing cores**
Parallel = 60% = 0.6 Serial = 1 – 0.6 = 0.4 Formula $Speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$ $Speedup \leq \frac{1}{0.4 + \frac{(1-0.4)}{2}}$ $Speedup \leq 1.42\ times$	Parallel = 60% = 0.6 Serial = 1 – 0.6 = 0.4 Formula $Speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$ $Speedup \leq \frac{1}{0.4 + \frac{(1-0.4)}{4}}$ $Speedup \leq 1.82\ times$

c. Consider the following code segment:

```
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
   fork();
   thread_create(...);
}
fork();
```

a. How many unique processes are created? 6 Processes [2.5 Mark]
b. How many unique threads are created? 1 thread [2.5 Mark]

CLO 3: **Evaluate the commonly used mechanisms for scheduling of tasks and implement synchronization mechanisms like Semaphores, TSL, etc.** **[15 Marks]**

Q 9:

Consider the following scenario and apply preemptive version of Shortest Job First scheduling and complete the given table.

Process	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Response Time
P1	0	7			
P2	2	4			
P3	4	1			
P4	5	4			
P5	7	3			
P6	8	1			

Draw the Gantt Chart. [4 marks]

Compute Average Turnaround time, Average waiting time and average Response time.

mechanisms like Semaphores, TSL, etc. **[15 Marks]**

Q 9:

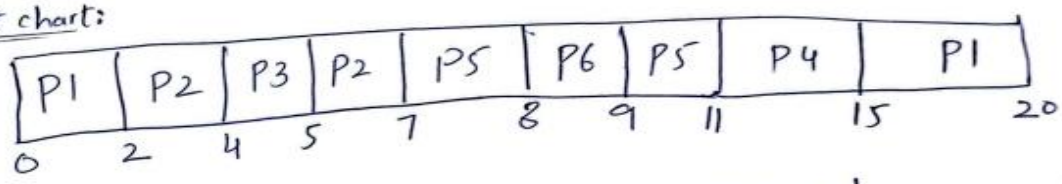
Consider the following scenario and apply preemptive version of Shortest Job First scheduling and complete the given table.

Process	Arrival Time	Burst Time	Turnaround Time (CT-AT)	Waiting Time (TAT-BT)	Response Time (ST-AT)	CT
P1	0	7	20-0=20	20-7=13	0-0=0	20
P2	2	4	7-2=5	5-4=1	2-2=0	7
P3	4	1	5-4=1	1-1=0	4-4=0	5
P4	5	4	15-5=10	10-4=6	11-5=6	15
P5	7	3	11-7=4	4-3=1	7-7=0	11
P6	8	1	9-8=1	1-1=0	8-8=0	9

Draw the Gantt Chart. **[4 marks]**

Compute Average Turnaround time, Average waiting time and average Response time.

Gantt chart:



Average Turnaround Time = $\frac{20+5+1+10+4+1}{6}$

$= \boxed{6.83}$

Average Waiting time = $\frac{13+1+0+6+1+0}{6}$

$= \boxed{3.5}$

Average Response time = $\frac{0+0+0+6+0+0}{6}$

$= \boxed{1}$