Important Topics for End-term Examinations

1) Memory management, Memory management techniques.

2) Deadlock and its handling methods

3) Process, Process state life cycle, Multiprocessing

4) Process Synchronization (PS), Solutions for PS, Classical synchronization problems

5) Process scheduling, Disc scheduling, and Page replacement algorithms.

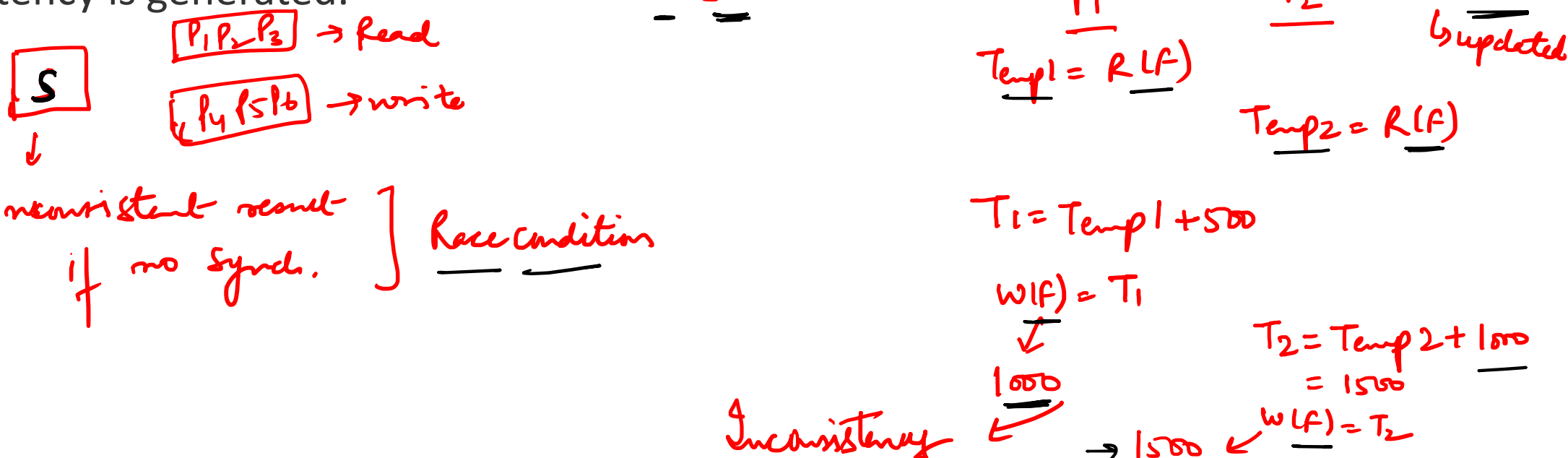
OPERATING SYSTEM: CSET209

OUTLINE

- Reader Writer Problem
- Monitor

1. READER WRITER PROBLEM

- The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets.
- The readers-writers problem is used for managing synchronization among various reader and writer processes so that there are no problems with the data sets, i.e. no inconsistency is generated.



WHY READER WRITER PROBLEM?

- Let's understand with an example - If two or more than two readers want to access the file at the same point in time there will be no problem. However, in other situations like when two writers or one reader and one writer wants to access the file at the same point of time, there may occur some problems, hence the task is to design the code in such a manner that if one reader is reading then no writer is allowed to update at the same point of time, similarly, if one writer is writing no reader is allowed to read the file at that point of time and if one writer is updating a file other writers should not be allowed to update the file at the same point of time. However, multiple readers can access the object at the same time.

TABLE 1

Case	Process 1	Process 2	Allowed / Not Allowed
Case 1	Writing	Writing	Not Allowed
Case 2	Reading	Writing	Not Allowed
Case 3	Writing	Reading	Not Allowed
Case 4	Reading	Reading	Allowed

SOLUTION FOR READER WRITER PROBLEM?

B. Semaphore = 0 or 1

- The solution of readers and writers can be implemented using binary semaphores.
- **Code of reader**
- **Mutex** and **write** are **semaphores** that have an initial value of 1, whereas **the readcount** variable has an initial value as 0.
- Both **mutex** and **write** are common in reader and writer process code
- Semaphore **mutex** ensures mutual exclusion and semaphore **write** handles the writing mechanism.

mutex = 1, write = 1;

```
static int readcount = 0;
wait (mutex);                                   → mutex = 0
readcount ++; // on each entry of reader increment readcount
if (readcount == 1)
{
  wait (write);                                 → write = 0
}
signal(mutex);                                  → mutex = 1

--READ THE FILE?                                R1 ✓

wait(mutex);                                    → mutex = 0
readcount --; // on every exit of reader decrement readcount
if (readcount == 0)
{
  signal (write);                               → write = 1
}
signal(mutex);                                  → mutex = 1
```

R2

SOLUTION FOR READER WRITER PROBLEM?

Code for Reader Process

The **readcount** variable denotes the number of readers accessing the file concurrently. The moment variable **readcount** becomes 1, **wait** operation is used to write semaphore which decreases the value by one. This means that a writer is not allowed how to access the file anymore. On completion of the read operation, **readcount** is decremented by one. When **readcount** becomes 0, the signal operation which is used to **write** permits a writer to access the file.

```
static int readcount = 0;
wait (mutex);
readcount ++; // on each entry of reader increment readcount
if (readcount == 1)
{
  wait (write);
}
signal(mutex);


--READ THE FILE?


wait(mutex);
readcount --; // on every exit of reader decrement readcount
if (readcount == 0)
{
  signal (write);
}
signal(mutex);
```

SOLUTION FOR READER WRITER PROBLEM?

Code for Writer Process

If a writer wishes to access the file, **wait** operation is performed on **write** semaphore, which decrements **write** to 0 and no other writer can access the file. On completion of the writing job by the writer who was accessing the file, the signal operation is performed on **write**.

write = 1

```
wait(write);
WRITE INTO THE FILE
signal(write);
```

write = 0

write

write = 1

MONITOR

A monitor is a synchronization mechanism that allows threads to have:

- *mutual exclusion* – only one thread can execute the method at a certain point in time, using *locks*
- *cooperation* – the ability to make threads wait for certain conditions to be met, using *wait-set*

```
monitor {

   //shared variable declarations
   data variables;
   Procedure P1() { ... }
   Procedure P2() { ... }
   .
   .
   .
   Procedure Pn() { ... }
   Initialization Code() { ... }
}
```

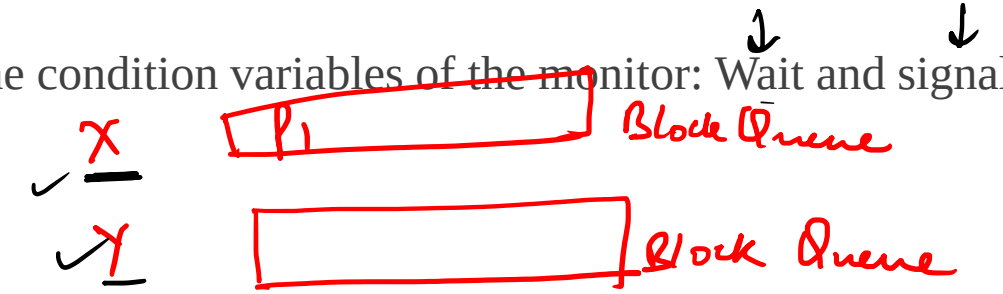
MONITOR

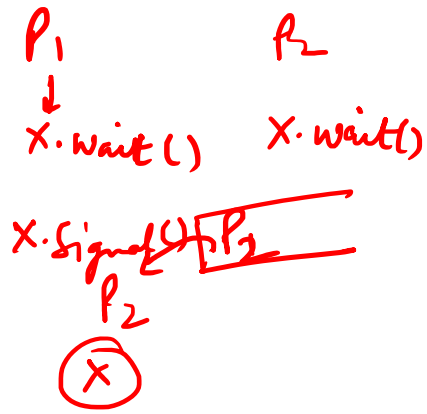
Condition Variables:

Two different operations are performed on the condition variables of the monitor: Wait and signal.

let say we have 2 condition variables

condition x, y; // Declaring variable





Wait operation: x.wait() : Process performing wait operation on any condition variable are suspended. The suspended processes are placed in block queue of that condition variable.

Note: Each condition variable has its unique block queue.

Signal operation: x.signal(): When a process performs signal operation on condition variable, one of the blocked processes is given chance.

```
If (x block queue empty)
  // Ignore signal
else
  // Resume a process from block queue.
```

ADVANTAGE AND LIMITATIONS OF MONITOR

Advantages

1. Mutual exclusion is automatic in monitors.
2. Monitors are less difficult to implement than semaphores.
3. Monitors may overcome the timing errors that occur when semaphores are used.
4. Monitors are a collection of procedures and condition variables that are combined in a special type of module.

Limitations:

1. Monitors must be implemented into the programming language.
2. The compiler should generate code for them.
3. It gives the compiler the additional burden of knowing what operating system features is available for controlling access to crucial sections in concurrent processes.

SEMAPHORE VS MONITOR

Features	Semaphore	Monitor
Definition	A semaphore is an integer variable that allows many processes in a parallel system to manage access to a common resource like a multitasking OS.	It is a synchronization process that enables threads to have mutual exclusion and the wait() for a given condition to become true.

Syntax	// Wait Operation
	wait(Semaphore S) {
	while (S<=0);
	S--;
	}
	// Signal Operation
	signal(Semaphore S) {
	S++;
	}
		monitor {
		//shared variable declarations
		data variables;
		Procedure P1() { ... }
		Procedure P2() { ... }
		.
		.
		.
		Procedure Pn() { ... }
		}

Access	When a process uses shared resources, it calls the wait() method on S, and when it releases them, it uses the signal() method on S.	When a process uses shared resources in the monitor, it has to access them via procedures.
Action	The semaphore's value shows the number of shared resources available in the system.	The Monitor type includes shared variables as well as a set of procedures that operate on them.
Condition Variable	No condition variables.	It has condition variables.

Thank You

Semaphore problems

Problem 1

Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d, and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the V operation (i.e., signal) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes?

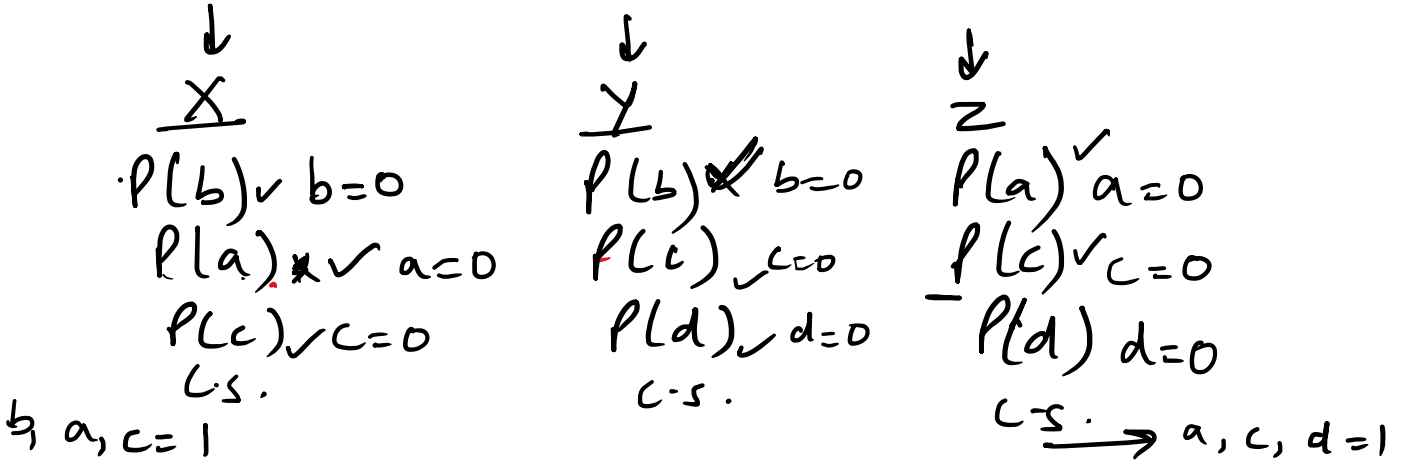
a, b, c, d = 1

(A) X: P(a)P(b)P(c) Y: P(b)P(c)P(d) Z: P(c)P(d)P(a)

(B) X: P(b)P(a)P(c) Y: P(b)P(c)P(d) Z: P(a)P(c)P(d)

(C) X: P(b)P(a)P(c) Y: P(c)P(b)P(d) Z: P(a)P(c)P(d)

(D) X: P(a)P(b)P(c) Y: P(c)P(b)P(d) Z: P(c)P(d)P(a)



Problem 2

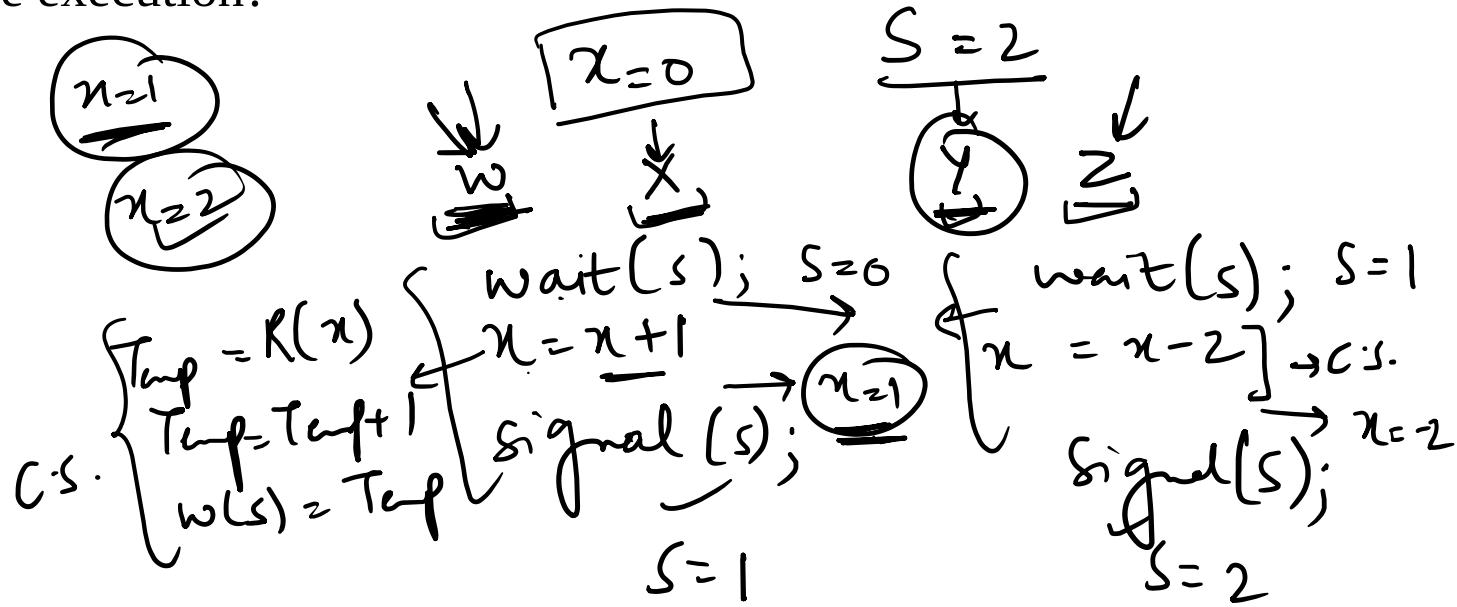
A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all process's complete execution?

(A) -2

(B) -1

(C) 1

(D) 2



Problem 3

The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0 and S2 = 0.

Process P0	Process P1	Process P2

while (true)
{
wait (S0);
print '0'
release (S1);
release (S2);
}
	wait (S1);
	release (S0);
		wait (S2);
		release (S0);

How many times will process P0 print '0'?

1. At least twice
2. Exactly twice
3. Exactly thrice
4. Exactly once

Set-B

Name: **Enrollment Num:**

Quiz-1

Q1-

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)
002451	0	12
002452	2	14
002453	4	18
002456	6	16

For the above-given arrangement of processes and their arrival and burst time, find out the Average Turnaround time and Average waiting time for the Round-robin scheduling algorithm with time quantum T=3 units. Also, draw the Gantt chart.

Solve here:

Set-B

Q2-

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)	Priority
001	0	12	2
010	1	14	1
011	3	18	3
100	4	16	4
101	2	8	5
110	5	10	6

For the above-given arrangement of processes and their arrival and burst time, find out the Average Turnaround time and Average waiting time for the FCFS and preemptive SJF/SRTF scheduling algorithm. Also, draw the Gantt chart for both the scheduling algorithms.

Solve here:

Set-A

Name: **Enrollment Num:**

Quiz-1

Q1-

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)
002451	0	12
002452	2	14
002453	4	18
002456	6	16

For the above-given arrangement of processes and their arrival and burst time, find out the Average Turnaround time and Average waiting time for SJF/SJN and Preemptive SJF scheduling algorithms. Also, draw the Gantt chart for both the scheduling algorithms.

Solve here:

Set-A

Q2-

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)	Priority
001	0	12	2
010	1	14	1
011	3	18	3
100	4	16	4
101	2	8	5
110	5	10	6

For the above-given arrangement of processes and their arrival and burst time, find out the Average Turnaround time and Average waiting time for the Round-robin scheduling algorithm having time quantum T=4 units. Also, draw the Gantt chart.

Solve here:

Scheduling problems

Q1.

Follow FCFS scheduling.
Find avg waiting time and avg turnaround time

Process ID	Arrival Time	Burst Time	Completion Time
P0	0	6	6
P1	1	8	14
P2	2	10	24
P3	3	12	36

Sol.

Avg waiting time = 9.5
Avg turnaround time = 18.5

Q1. Non preemptive SJF scheduling.
Find avg waiting time and avg turnaround time

Process ID	Burst Time	Arrival Time
P0	8	5
P1	5	0
P2	9	4
P3	2	1

Sol.
Avg waiting time = 3.75
Avg turnaround time = 9.75

Q1.
Preemptive SJF (SRTF) scheduling.
Find avg waiting time and avg turnaround time

Process	Burst time	Arrival time
P1	18	0
P2	4	1
P3	7	2
P4	2	3

Sol.
Avg waiting time = 20
Avg turnaround time = 12.75

Q1.
Round Robin scheduling (TQ=8).
Find num of context switches, avg waiting time and avg turnaround time

Process No.	Arrival Time (AT)	Burst Time (BT)
P_1	0	10
P_2	1	9
P_3	2	12
P_4	3	6

Sol.
Num of context switches = 6
Avg waiting time = 22.25
Avg turnaround time = 31.5

Q1.
Priority scheduling.
Find avg waiting time and avg turnaround time

Process	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

Sol.
Avg waiting time = 5.6
Avg turnaround time = 8.8

Practice Questions-2

Q1.

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)
AA	0	4
AB	1	6
BA	3	8
BB	4	4

Q2.

Process Id	Arrival Time (A.T.)	Burst Time (B.T.)
P0	3	6
P1	5	2
P3	1	8
P5	0	3
P6	2	4

Q3.

Process Id	A.T.	B.T.
00	0	9
01	2	6
10	4	4
11	5	6

➔ Find out the Completion/Finish Time, Turn Around Time, and Waiting Time for each of the processes in the above-given Tables using Round Robin (Tq=2 and 4), and Highest Response Ratio Next (HRRN) CPU scheduling algorithms.

Practice Questions

Q1.

Process Id	A.T.	B.T.
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Q2.

Process Id	A.T.	B.T.
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

Q3.

Process Id	A.T.	B.T.
00	0	7
01	2	4
10	4	1
11	5	4

➔ Find out the Completion/Finish Time, Turn Around Time, and Waiting Time for each of the processes in the above-given Tables using FCFS, SJF, and SRTF CPU scheduling algorithms.
Also, arrange the Average waiting time in ascending orders for the above three arrangements.

Note: A.T. and B.T. stand for Arrival and Burst Time respectively.