Name : Shubham
Roll no : 2301730100
Course : B Tech CSE AI/ML
Sec - B

DELTA Page No. Date / /

Assignment - 5

1Q → Modern computers still need operating system because the OS manages Hardware and provide an easy interface for users and program.

→ It creates abstraction like processes for Running program, virtual memory for efficient storage Use, and files for I/O operations.

→ The OS handle process Management CPU scheduling, Multitasking, memory management (allocation protection) and I/O management (Device communication).

→ it hides hardware complexity, ensures Security, and enables Resource sharing.

19/11/25

2Q → Monolithic os divides has all system services in one large Kernal - fast but Hard to Maintain.

→ layered OS divides the system into layers, each built on the one below - easiest to debug.

→ MicroKernal OS Keeps only core functions (like communication) scheduling in the Kernal, moving other to user space. It is slower But more Reliable and Secure.

→ for a Distributed Web application, a MicroKernal OS is best because failures in one Service Don't crash the whole System, and update are easier.

3Q → Threads are lighter than processes Because they share the Same memory, so the OS doesn't need to create a new address space like it does in a process.

→ A process control block (PCB) stores a lot of

2301730100

DELTA Page No. Date / /

Heavy Information, while a Thread control Block (TCB) stores much less.

→ Because of this, thread content switching is faster than switching Blw processes.

→ processes need more Resources and protection, making them slower to manage.

→ so threads are usually more efficient, But processes give Better isolation and Safety.

4Q case study: LINUX OS - demonstrating a Basic system call (file creation & Read).

✓ python program (uses os system call through open(), write(), Read().

```
# file creation and Reading using system calls.
# creating file (writes system call)
with open ("sample.txt", "w") as f:
    f.write ("Hello OS! This is a linux system call demo.").
# Reading the file (Read system call).
with open ("sample.txt", "r") as f:
    Data = f.Read()
print ("file content:", data)
```

Output:

file content: Hello OS! This is a linux system call Demo.

✓ Relevance to OS Architecture.

→ when python uses open(), Read(), or write(), it asks the Linux Kernel to perform a task.

→ These actions are called System calls, which act like a Bridge blw the program and the OS.

→ The OS ensures safe access to the file system, Manages permissions and stores data correctly.
→ This Demonstrates how user-level programs depends on Kernel services, which is a core concept in OS Architecture.

SQ Synchronous checkpointing:

→ All nodes stopes together, save their state at the same Time then continue.
like Taking a group photo everyone freezes for a moment

• Diagram:

Node A: ... work ... [checkpoint] ... work ...
Node B: ... work ... [checkpoint] ... work ...
Node C: ... work ... [checkpoint] ... work ...
↑ All checkpoints Together.

• pseudocode:
→ coordinator → send checkpoints Request to all nodes
All nodes pause updates.
each node save state to stable storage.
All nodes confirm DONE.
System Resume.

• Recovery:
→ load last global checkpoint.
Restart all nodes from the same point in time.

✓ Asynchronous checkpointing:
Nodes take checkpoints whenever they want, without

2301730100

DELTA Page No.
Date / /

stoping other.
like taking individual Selfie instead of Group photo.

a) Diagram

Time →

Node A : ..... work ... [CP] .... work ... [CP] ...
Node B : ..... work ... [CP] ...
Node C : ..... work ... [CP] ..... work .....

(checkpoints occur at different Times).

- Pseudocode : each node independtly
  if Times expires
  save checkpoints.
- Recovery :
  May need Message logs to re Build a consistent state.

60 a)

1) Transparency (Access + location Transparency)
   User should feel like all files are on one computer even though they are spread accross many Machines.
   - The system must hide where the file is Stored
   - Ensures easy file sharing and uniform Access.

2) fault Tolerance and Reliability.
   Distributed system must Keep working even if one machine crashes.
   - file must be Replicated or logged.
   - Recovery must ensures no data loss.

b) 1) client - server Architecture (NFS style)
   Client Requests file → server stores files.
   - cache on the client to Reduce load.
   - stateless servers improve crash Recovery.

   Benefits : simple Scalable, good performance.

2301730100
DELTA Page No.
Date / /

2) Replication - Based Architecture : (Make copies of files across Multiple Sites -
- ensures fault Tolerance
- Read operation Becomes faster.

3) Distributed Naming and Metadata servers :
stores file names separatly from file data
- uses lookup Tables (like HDFS Name Node + Data Node Model).

Benefits : faster search, load balancing.
easier Management.

7Q a) Banker's algorithm checks Before granting a lock whether the system will still be in safe state. like a Banker giving loans only when he Knows every customer can finish and Return Money. So the OS only locks accounts if all Transcations can still complete, preventing deadlocks.

b) Detection Approach :
→ Build a wait-for-graph (who is waiting for whose account lock).
→ if the Graph forms a cycle → Deadlocks detected
Recovery Approach -
→ About or Rollback one of the Transactions holding the locks.
→ Release its lock, letting others continues
→ Restart the aborted Transaction later.

8Q producer - consumer using Semaphores
we use :
- Mutex (Binary Semaphore) - allow only one thread

to access the buffer at a Time.
→ Empty Semaphore → counts empty slots (producers wait if Buffer fuels)
→ full Semaphores → counts filled slots (consumers wait if Buffer empty).

Simple Diagram :

producer → [empty---] → [MUTEX] → BUFFER → [full] → Consumer.

Python Code : (Semaphore Solution)

```
import threading, time, random
from threading import Semaphore

buffer = [ ]
Buffer_size = 5.
mutex = Semaphore (1)
empty = Semaphore (Buffer_size)
full = Semaphore (0).

def producer ();
    while True :
        item = random . ~~randint~~ randint (1, 100)
        empty . acquire ()
        mutex . acquire ()
        buffer . append (item)
        print ("produced :", item)
        mutex . Release ().
        full . Release ().

def Consumer ();
    while True :
```

full.acquire()
muten.acquire()
item = Buffer.pop(0).
print("consumed:", item)
muten.Release().
empty.Release().

Threading.thread(target = producer).start()
Threading.thread(target = consumer).start()

q11 a) Scheduling strategy :
→ Use priority scheduling with preemption :
- Security Devices (cameras, alarms) = highest priority
→ their interupts immediatly preempt other tasks.
- Normal tasks (lights, AC) = lower priority → Runs only when no critical Tasks

Justification : ensures instant Response to security events while Delaying non-critical work.

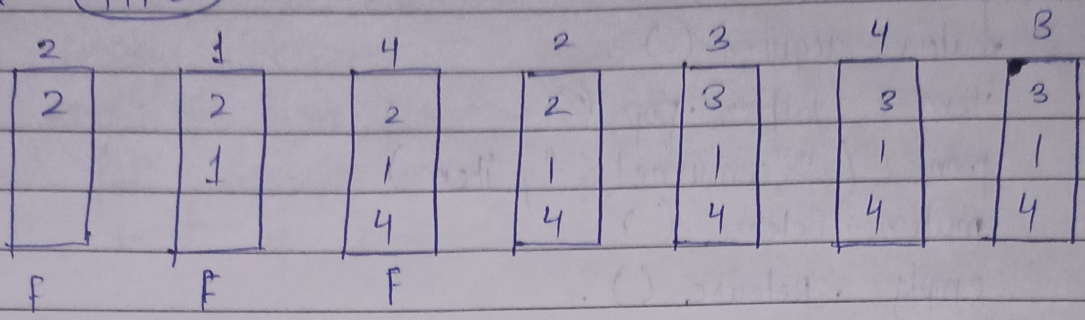
b) IPC Methods :
use Message Queues + shared Memory + signals :
- Message Queues : send alerts from Devices → OS Reliably (ordered + asynchronous)
- shared Memory : fast Data sharing blw processes (eg : sensor Reading).
- Signals / Interrupts : notify OS immediatly when Security trigger events

Justification : provides fast, Reliable, event-Driven communication in a Smart home-IOT Environment.

2301730100

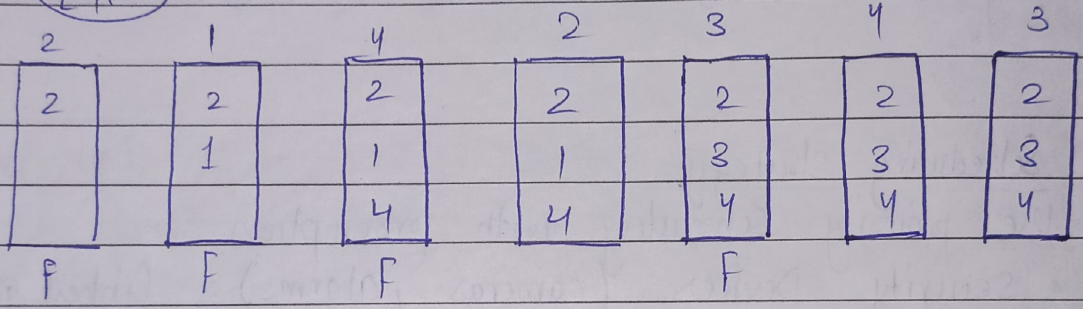
DELTA Page No. Date / /

Ques (FIFO)



FIFO page fault = 4

LRU

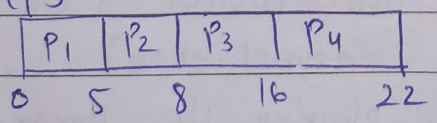


LRU page fault = 4

Ques

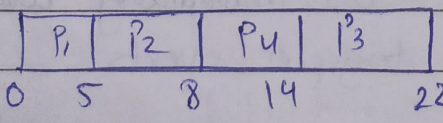
process	Burst Time (ms)	Arrival Time (ms)
P_1	5	0
P_2	3	1
P_3	8	2
P_4	6	3

a) FCFS



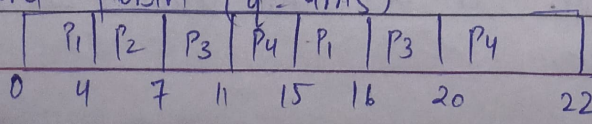
order : $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$

b) SJF (non-preemptive)



- t = 0 → P_1
- t = 5 → P_2, P_3, P_4 (choose P_2) shortest
- After P_2 → choose P_4.
- finally P_3

c) Round Robin (q = 4ms)



2301730100

DELTA Page No
Date / /

P_1 : Runs 4 → Remaining 1
P_2 : Runs 3 → Done
P_3 : " 4 → Remaining 4
P_4 : " 4 → " 2
P_1 : " 1 → Done
P_3 : " 4 → Done
P_4 : " 2 → Done

(5) i) FCFS

process	WT	TAT
P_1	0	5
P_2	4	7
P_3	6	14
P_4	13	19

Avg WT = (0+4+6+13) /4 = 5.75.
Avg TAT = 11.25

2) SJF (Non- preemptive)

process	WT	TAT
P_1	0	5
P_2	4	7
P_3	5	11
P_4	12	20

Avg WT = 5.25
Avg TAT = 10.75

3) Round Robin (q=4)

process	WT	TAT
P_1	10-0-5 = 11	16
P_2	7-1-3 = 3	6
P_3	10	18
P_4	13	19

Avg WT = 9.25
Avg TAT = 14.75

DELTA Page No.
Date / /

c) FCFS –

- simple But suffer from "convoy effect"
- long Jobs Block ones
- worst Responsiveness –

→ SJF –

- ~~non~~ Minimize average waiting Time.

→ Round Robin –

- Best fairness and Responsiveness –
- Higher content – switch overhead.