name - yadav amit
roll no - 2301010218
course- B. Tech CSE

Assignment- 1

1. Despite the evolution of hardware, why do modern systems still rely heavily on operating Systems?
Answer:
- Hardware provides raw capabilities; the OS provides abstraction (processes, files, devices)
making hardware usable.
- Resource management: CPU scheduling, memory management, I/O coordination and device drivers.
- Security and access control (user accounts, permissions, isolation).
- Multitasking and concurrency support plus APIs for applications -> portability and developer
productivity.
- Performance optimizations (caching,

buffering) and fault tolerance features (process
isolation, recovery).
2. OS for a wearable heart-rate monitor - which type and why?
Answer:
- A real-time embedded OS (RTOS) - e.g., FreeRTOS, Zephyr.
- Reason: deterministic, low-latency scheduling (hard/soft real-time constraints), small footprint, energy efficiency, reliable interrupt handling and simple task model (tasks + ISRs).
- Provides predictable deadlines for sampling/alerts and safe, low-power operation.
3. For a performance-critical environment which structure would you avoid (Monolithic, Layered, Microkernel), and why?
Answer:
- Avoid Microkernel for strict performance-critical cases because it incurs extra IPC and context-switching overhead between user-space services and the kernel.
- Monolithic kernels generally deliver better raw performance (fewer context switches), while

layered designs trade some performance for modularity and maintainability.

4. A developer claims OS structure doesn't matter as long as processes run. Support or refute.

Answer:

- Refute: Structure affects performance, security, robustness, maintainability, and extensibility.
- Examples: microkernels improve fault isolation but may reduce throughput; monolithic kernels are faster but risk system-wide crashes from buggy drivers. So structure matters for real-world
requirements.

5. while debugging, you find a process switching error.

i) How analysing the PCB points to misinitialized registers or incorrect states?

Answer:

- The PCB saves CPU registers, program counter, stack pointer and process state. Mismatch
between saved/restored register values (e.g.,

wrong PC or SP) in PCB indicates misinitialization or corrupted context.
- Inconsistent state fields (e.g., wrong process state: RUNNING vs READY) in PCB reveals scheduler/driver bugs.
ii) If a task is moved unexpectedly from running to waiting, what precisely does context switching involve?
Answer:
- Save current CPU state into the outgoing process's PCB (registers, PC, SP, flags).
- Update process state to WAITING and place it in appropriate wait queue.
- Scheduler picks a new process: update its PCB to RUNNING and restore its CPU state (load
registers, PC, SP).
- Update MMU/TLB as needed and perform any accounting (timestamping).
iii) OS needs to allocate I/O resources mid-execution. which type of system call to use and
why?
Answer:

- use asynchronous, non-blocking I/O system calls when possible: the process requests the I/O
and continues executing while the OS completes the operation (e.g., submit + callback/notification).
- Rationale: prevents stalling CPU-bound work, improves concurrency. Blocking synchronous calls
are simpler but may hurt responsiveness.

Part B - Application / Numerical

6. Context Switching Time Calculation:

Given: Save state = 2 ms, Load state = 3 ms, Scheduler overhead = 1 ms

a) Total context switching time = 2 + 3 + 1 = 6 ms.

b) Impact on multitasking:
- Each context switch costs CPU time (overhead) reducing usable CPU for tasks.
- High context-switch frequency (small time-slices) increases overhead and lowers throughput;
choose quantum to balance responsiveness vs overhead.

7. Thread Efficiency Check:

Given: Total time (single-threaded) = 40 s, threads per process (ideal conditions) = 2

Estimate execution time:

- In ideal perfect parallelism across 2 threads: 40 / 2 = 20 s.

Explanation:

- Multithreading allows overlapping I/O and CPU work, better CPU utilization, reduced idle time due to blocking calls, and potential parallel execution on multi-core CPUs. overheads (synchronization, contention) reduce the ideal gain.

8. Given processes P1..P4 with bursts:

P1=5 ms, P2=3 ms, P3=8 ms, P4=6 ms

a) Gantt charts

FCFS (P1, P2, P3, P4):

|0-5 |5-8 |8-16 |16-22|

P1 P2 P3 P4

Non-preemptive SJF (order by burst: P2, P1, P4, P3):

|0-3 |3-8 |8-14 |14-22|

P2 P1 P4 P3

Round Robin (Quantum = 4 ms) - execution

order (time slices):

|0-4|4-7|7-11|11-15|15-16|16-20|20-22|

P1 P2 P3 P4 P1 P3 P4

(See computed completion times below)

b) Average waiting Time (AWT) and Average Turnaround Time (ATT)

FCFS:

- waiting times: P1=0, P2=5, P3=8, P4=16
- Turnaround times: P1=5, P2=8, P3=16, P4=22
- AWT = (0+5+8+16)/4 = 7.25 ms
- ATT = (5+8+16+22)/4 = 12.75 ms

Non-preemptive SJF:

- order: P2, P1, P4, P3
- waiting: P2=0, P1=3, P4=8, P3=14
- Turnaround: P2=3, P1=8, P4=14, P3=22
- AWT = (0+3+8+14)/4 = 6.25 ms
- ATT = (3+8+14+22)/4 = 11.75 ms

Round Robin (Quantum=4 ms):

- Completion times: P1=16, P2=7, P3=20, P4=22
- waiting = Turnaround - Burst:

P1 waiting = 16 - 5 = 11

P2 waiting = 7 - 3 = 4

P3 waiting = 20 - 8 = 12

P4 waiting = 22 - 6 = 16

- AWT = (11+4+12+16)/4 = 10.75 ms
- ATT = (16+7+20+22)/4 = 16.25 ms

c) which algorithm best balances throughput and turnaround?

- Non-preemptive SJF gives the lowest average waiting time (6.25 ms) and lowest average turnaround (11.75 ms) for this job mix, so it best balances throughput and turnaround here.
- Note: SJF requires knowledge of burst lengths and can starve long jobs.

9. i) Company migrating services to virtualized cloud:

a) which OS architecture to select for scalability and security - and why?

Answer:

- Microkernel or minimal-kernel approach is preferable for strong security/isolation (small trusted kernel, services in user space). Combine with a hypervisor (Type-1) for VM-level isolation.
- Microkernel reduces attack surface and isolates drivers/services; hypervisors provide

tenant
separation and resource control.
b) How VMs aid isolation, management, resource optimization:
- Isolation: VMs keep faults within the guest boundary and separate tenants.
- Management: snapshots, live migration, cloning, easy upgrades and rollback.
- Resource optimization: overcommit, dynamic allocation, autoscaling, and monitoring.
ii) Smart home system - scheduling and IPC:
a) How OS uses scheduling and IPC to prioritize intrusion detection vs lighting:
- Use priority-based preemptive scheduling and interrupts to ensure intrusion detection gets
CPU immediately.
- Use IPC (message queues, event notifications) to pass sensor events to handlers; avoid busy-
waiting.
- Use priority inheritance to prevent priority inversion on shared resources.
b) Candidate scheduling algorithms:

- Rate Monotonic Scheduling (RMS) for periodic tasks with static priorities.
- Earliest Deadline First (EDF) for dynamic deadlines and higher utilization.

Priority-based preemptive scheduling with real-time extensions and priority inheritance is practical for mixed-criticality IoT controllers.

End of solutions.