Operating Systems Revision Test - Week 6
ANSWER KEY

SECTION A: MULTIPLE CHOICE QUESTIONS (15 Questions - 30 Marks)

1. Answer: c) Compiling source code
- *Explanation:* Compiling is done by compilers, not the OS. The OS provides process management, memory management, and file system management as core functions.

2. Answer: c) Kernel layer
- *Explanation:* In layered architecture, the lowest layer (kernel) directly interacts with hardware, while higher layers provide abstractions.

3. Answer: b) Windows NT (original design)
- *Explanation:* Windows NT was originally designed with a microkernel architecture. Minix and QNX are other examples. Linux uses a monolithic kernel.

4. Answer: c) Better performance due to reduced context switching
- *Explanation:* Monolithic kernels have all services in kernel space, reducing context switches between user and kernel mode, resulting in better performance.

5. Answer: b) Bootloader
- *Explanation:* The bootloader (like GRUB, LILO) loads the kernel into memory. BIOS/UEFI initializes hardware and loads the bootloader.

6. Answer: b) GNU's Not Unix
- *Explanation:* GNU is a recursive acronym meaning "GNU's Not Unix." It represents the free software project that provides utilities used with Linux.

7. Answer: c) Shortest Job First (SJF)
- *Explanation:* SJF can cause starvation as longer processes may wait indefinitely if shorter processes keep arriving.

8. Answer: b) The outcome depends on the non-deterministic ordering of execution
- *Explanation:* Race conditions occur when multiple processes/threads access shared data concurrently, and the result depends on timing/ordering.

9. Answer: c) Semaphore
- *Explanation:* A semaphore uses a counter to control access. Mutex is binary (locked/unlocked), while semaphores can count.

10. Answer: b) Best-Fit
- *Explanation:* Best-Fit selects the smallest hole that can accommodate the process, minimizing wasted space.

11. Answer: b) Internal fragmentation only
- *Explanation:* Paging eliminates external fragmentation but can have internal fragmentation (unused space within a page).

12. Answer: b) FIFOs have a name in the filesystem and can be used by unrelated processes
- *Explanation:* Named pipes (FIFOs) exist as special files in the filesystem and can be accessed by any process with appropriate permissions.

13. Answer: c) SIGINT
- *Explanation:* SIGINT (interrupt signal) is sent when Ctrl+C is pressed. SIGKILL forcefully terminates, SIGTERM requests termination.

SECTION B: NUMERICAL QUESTIONS - CPU SCHEDULING (5 Questions - 30 Marks)
Question 14: FCFS Scheduling

Solution:
Gantt Chart:
| P1 (0-8) | P2 (8-12) | P3 (12-21) | P4 (21-26) |
Calculations:

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	8	8	8 - 0 = 8	8 - 8 = 0
P2	1	4	12	12 - 1 = 11	11 - 4 = 7
P3	2	9	21	21 - 2 = 19	19 - 9 = 10
P4	3	5	26	26 - 3 = 23	23 - 5 = 18

Average Waiting Time = (0 + 7 + 10 + 18) / 4 = 35 / 4 = 8.75 units
Average Turnaround Time = (8 + 11 + 19 + 23) / 4 = 61 / 4 = 15.25 units

---

Question 15: SJF Non-Preemptive Scheduling
Solution:
Process execution order (by shortest burst time when CPU is free):

- At t=0: P1 arrives, starts executing
- At t=8: P1 completes. P2 (BT=4), P3 (BT=9), P4 (BT=5) are ready. P2 has shortest burst time.
- At t=12: P2 completes. P3 (BT=9), P4 (BT=5) are ready. P4 has shortest burst time.
- At t=17: P4 completes. P3 executes.
- At t=26: P3 completes.

Gantt Chart:
| P1 (0-8) | P2 (8-12) | P4 (12-17) | P3 (17-26) |

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	8	8	8 - 0 = 8	8 - 8 = 0
P2	1	4	12	12 - 1 = 11	11 - 4 = 7
P4	3	5	17	17 - 3 = 14	14 - 5 = 9
P3	2	9	26	26 - 2 = 24	24 - 9 = 15

Average Waiting Time = (0 + 7 + 15 + 9) / 4 = 31 / 4 = 7.75 units
Average Turnaround Time = (8 + 11 + 24 + 14) / 4 = 57 / 4 = 14.25 units

---

SECTION C: SCENARIO BASED QUESTIONS (10 Questions - 40 Marks)
Question 16 (3 marks)
Answer:
(a) Bottleneck Identification: The bottleneck is **Disk I/O** (85% wait time indicates disk is the limiting factor).
(b) Two Solutions:

1. **Upgrade to SSD/NVMe drives** - Solid-state drives provide much faster I/O operations compared to traditional HDDs, reducing wait times significantly.
2. **Implement caching mechanisms** - Use a reverse proxy with caching (like Varnish or Redis) to cache frequently accessed content in RAM, reducing disk reads.

Alternative solutions: Increase RAM for file system cache, optimize database queries to reduce disk access, implement RAID for better I/O performance.

Question 17 (4 marks)

Answer:
Available holes: 100 KB, 250 KB, 180 KB, 300 KB Request: 150 KB
(a) Best-Fit algorithm: 180 KB hole selected (smallest hole that fits 150 KB)
(b) Worst-Fit algorithm: 300 KB hole selected (largest available hole)
(c) Remaining hole sizes:

- **Best-Fit:** 180 - 150 = **30 KB remaining**
  - Final holes: 100 KB, 250 KB, 30 KB, 300 KB
- **Worst-Fit:** 300 - 150 = **150 KB remaining**
  - Final holes: 100 KB, 250 KB, 180 KB, 150 KB

(d) Less external fragmentation: Worst-Fit results in less external fragmentation in this case because:

- Worst-Fit leaves a 150 KB hole (large enough for future allocations)
- Best-Fit leaves a 30 KB hole (too small for most requests, creating unusable fragmented space)

Worst-Fit results in LESS external fragmentation in this case.
Reasoning:

- Worst-Fit leaves a **150 KB hole**, which is large enough to accommodate future allocation requests
- Best-Fit leaves a **30 KB hole**, which is too small for most allocations, creating **wasted fragmented space**
- Smaller unusable holes = more external fragmentation

General Principle:

- **External fragmentation is worse when you have many small, unusable holes**
- Larger remaining holes (even if fewer) are better because they can satisfy future requests
- In this specific scenario, Worst-Fit performs better by avoiding tiny unusable fragments

---

Question 18 (4 marks)
Answer:
(a) Problem without synchronization (Race Condition):
Without synchronization, the following can occur:

1. Thread 1 reads balance = $600
2. Thread 2 reads balance = $600 (before Thread 1 updates)
3. Thread 1 checks if $500 ≤ $600 (yes), calculates new balance = $100
4. Thread 2 checks if $300 ≤ $600 (yes), calculates new balance = $300
5. Thread 1 writes balance = $100
6. Thread 2 writes balance = $300 (overwrites Thread 1's update)

Result: Final balance is $300, but $800 was withdrawn from a $600 account! The account is now overdrawn and Thread 1's withdrawal is lost.
Alternatively, Thread 2 could overwrite first, leaving balance at $100, still incorrect.
(b) Synchronization mechanism: Mutex (Mutual Exclusion Lock) or **Semaphore (binary semaphore)**

The critical section (read balance → check → calculate → write) must be atomic. A mutex ensures only one thread can execute this section at a time, preventing the race condition.

Question 19 (5 marks)
Answer:
(a) Bottleneck identification: The bottleneck is **CPU scheduling/context switching overhead**. Despite high CPU utilization (95%), throughput is low because:

- CPU-bound processes are monopolizing the CPU
- I/O-bound processes are not getting enough CPU time to initiate I/O operations
- I/O devices are underutilized (30%) due to I/O-bound processes waiting for CPU

(b) Two specific improvements:

1. **Adjust scheduling algorithm or priorities:**
    - Give higher priority to I/O-bound processes so they can quickly execute their CPU bursts and issue I/O requests
    - This allows I/O devices to stay busy while CPU-bound processes use the CPU during I/O wait times
2. **Implement multi-level queue scheduling:**
    - Separate queues for CPU-bound and I/O-bound processes
    - Use shorter time quantum for I/O-bound processes
    - This ensures I/O-bound processes get frequent CPU access to maintain high I/O utilization

(c) Suitable scheduling algorithm:
Multi-Level Feedback Queue (MLFQ) or **Priority Scheduling with different priorities for I/O and CPU-bound processes**
Reason:

- MLFQ automatically identifies I/O-bound processes (those that frequently yield CPU before time quantum expires) and moves them to higher priority queues
- I/O-bound processes get quick CPU access to issue I/O operations
- CPU-bound processes get longer time slices when they run, reducing context switching
- This balances CPU and I/O utilization, improving overall throughput
- Achieves better overlap of CPU and I/O operations

Question 20 (3 marks)
Answer:
(a) IPC mechanism choice: Unnamed Pipe (Anonymous Pipe)
(b) Two justifications:

1. **Parent-child relationship:** Pipes are specifically designed for parent-child process communication. They are automatically created before the fork, and the child inherits the pipe file descriptors, making setup simple and efficient.
2. **One-time, small data transfer:** For a 2 KB one-time transfer, pipes are perfect. They provide a simple, lightweight FIFO buffer without the overhead of:
    - Shared memory (requires synchronization mechanisms)
    - FIFO (requires filesystem name, cleanup, accessible to unrelated processes - unnecessary here)
    - Sockets (overkill for local, one-way, parent-child communication)

3. **Automatic cleanup:** Pipes are automatically cleaned up when both ends are closed, requiring no manual resource management.

Additional note: Shared memory would be overkill for 2 KB, and sockets are unnecessary for local parent-child communication.

Question 21 (5 marks)

Answer:

(a) More efficient mechanism: Shared Memory is more efficient for frequently exchanging large amounts of data (100 MB).

Why:

- Shared memory allows both processes to directly access the same physical memory region
- Data is written once and read once (no copying between kernel and user space)
- No system call overhead for each data transfer after initial setup
- Message queues require copying data: Process A → kernel space → Process B (double copy)

(b) Two advantages of shared memory:

1. **Zero-copy data transfer:** Once the shared memory segment is mapped, processes can read/write directly without data copying, resulting in minimal overhead and maximum throughput for large data transfers.
2. **High performance:** After initial setup, accessing shared memory is as fast as accessing regular memory (simple pointer dereference), with no system call overhead per access.

Additional advantage: Scalable for very large data sets - 100 MB fits easily in shared memory, whereas message queues have size limits.

(c) One disadvantage of shared memory:

Requires explicit synchronization: Processes must use semaphores, mutexes, or other synchronization primitives to coordinate access, preventing race conditions. This adds complexity to the implementation - the programmer must:

- Implement reader-writer locks or similar mechanisms
- Handle synchronization errors
- Ensure proper sequencing of reads/writes

Message queues, in contrast, handle synchronization automatically (FIFO ordering, blocking on empty/full).

Question 22 (4 marks)

Answer:

Segment Table:

Segment	Base	Limit
1	4300	1200

(a) Segmentation fault analysis:

Logical address: (Segment 1, Offset 1500)

Check: Is Offset < Limit?

- Offset = 1500
- Limit = 1200
- 1500 > 1200 → **Offset exceeds segment limit**

Answer: YES, this will generate a segmentation fault.

Reason: The offset (1500) is beyond the segment's limit (1200), meaning the process is trying to access memory outside its allocated segment. This is a protection violation, and the hardware will generate a segmentation fault to prevent the invalid access.

(b) Physical address calculation:

Logical address: (Segment 0, Offset 400)

From segment table:

- Segment 0: Base = 2000, Limit = 800

Check validity: 400 < 800 ✓ (Valid access)

Calculate physical address:

- Physical Address = Base + Offset
- Physical Address = 2000 + 400
- **Physical Address = 2400**

Question 23 (4 marks)

Answer:

(a) IPC mechanism: Sockets (specifically **network sockets**)

Reason: Sockets are the only IPC mechanism that works across different machines over a network. Pipes, FIFOs, and shared memory are limited to processes on the same machine.

(b) Socket type for file transfer: TCP (Transmission Control Protocol) sockets - specifically **SOCK_STREAM**

Justification:

Why TCP:

1. **Reliability:** TCP guarantees delivery of all data in the correct order. For file transfer, data integrity is critical - missing or corrupted bytes would corrupt the file. TCP provides:
    - Acknowledgments for received data
    - Automatic retransmission of lost packets
    - Error checking with checksums
2. **Connection-oriented:** TCP establishes a reliable connection before data transfer, ensuring both endpoints are ready and maintaining the connection throughout the transfer. This is ideal for transferring complete files.

Why not UDP:

- UDP is connectionless and unreliable (no guaranteed delivery)
- Packets can be lost, duplicated, or arrive out of order
- Would require implementing custom reliability mechanisms
- Better for real-time streaming where occasional data loss is acceptable (video/audio streaming)

Summary: For file transfer where data integrity is paramount, TCP is the appropriate choice.

Question 24 (4 marks)

Answer:

(a) Answer: c) SIGTERM

Explanation: SIGTERM can be caught and handled, allowing the server to implement custom shutdown logic (stop accepting connections, wait for requests, cleanup). SIGKILL cannot be caught and would cause abrupt termination without cleanup.

b)(Answer: a) Parent sends SIGTERM to children → Children exit → Kernel sends SIGCHLD to parent
Explanation: The parent forwards SIGTERM to children for graceful shutdown. When children terminate, the kernel automatically sends SIGCHLD back to the parent, which can then call wait() to collect exit status.

(c) Answer: b) Send SIGKILL to force immediate termination
Explanation: SIGKILL is the only signal that cannot be caught or ignored, ensuring the unresponsive process is terminated. After graceful attempts (SIGTERM) fail, SIGKILL is the appropriate last resort.

(d) Answer: b) Parent exits immediately without waiting for children to finish, creating zombie processes
Explanation: The parent must wait for SIGCHLD signals and call wait()/waitpid() to collect child exit status. Exiting immediately leaves children as zombies (terminated but not reaped). The parent should set a flag in the handler and perform wait() operations in the main loop.
