Question no. 1

1. **How is FCFS scheduling not fair?**
FCFS (First-Come, First-Served) can lead to the “convoy effect,” where short processes get stuck behind long ones. This causes high average waiting time and is unfair to short jobs that arrived later.

2. **How are interrupts used during memory access in a modern OS? Give a specific example.**
When a page fault occurs (e.g., accessing a page not in memory), the OS uses an interrupt to pause the current process, load the required page into memory, and then resume the process. Example: accessing an array element whose page has been swapped out.

3. **Why does frequent context switching occur for long-running processes?**
To ensure responsiveness and fairness, preemptive schedulers periodically interrupt long processes to allow other processes (especially short or I/O-bound) to run. This leads to more frequent context switches.

4. **An application binary compiled for machine A may need to be recompiled to run on machine B.**
Because machine A and B may have different instruction sets, word sizes, or OS system call conventions. Compilation translates code into machine-specific binary; recompiling ensures compatibility.

5. **Do you think the term “voluntary context switch” is wrong? Justify.**
No, it’s appropriate. A voluntary context switch occurs when a process willingly gives up the CPU (e.g., by calling sleep or waiting for I/O), as opposed to being preempted by the scheduler.

6. **Why can a C program not directly invoke system calls and must use the standard C library?**
The standard C library (glibc) provides a portable API. Directly invoking system calls requires knowledge of low-level OS details like syscall numbers and calling conventions, which are abstracted by the library.

7. **Explain starvation in the context of the Readers-Writers problem.**
If readers are continuously allowed access, writers may starve—never getting a chance to write. This happens in a reader-preference solution where writer access is postponed until no readers are present.

---

8. **C statements to ensure moo() (P1) executes only after foo() (P0):**

```
#include <semaphore.h>
sem_t sem;

void P0() {
    foo();
    sem_post(&sem); // signal P1
}

void P1() {
    sem_wait(&sem); // wait until P0 signals
    moo();
}
```

Initialize sem to 0 using: sem_init(&sem, 0, 0);

9. **Difference between mutex and binary semaphore:**

- Mutex is owned by the thread that locks it and must be unlocked by the same thread.
- Binary semaphore does not have ownership; any thread can signal or wait. Mutex is stricter and prevents misuse.

10. **How does a race condition occur in a Producer-Consumer scenario?**
If producer and consumer access the shared buffer without synchronization, both may update buffer indices simultaneously, leading to inconsistent or corrupted data (e.g., overwriting or skipping items).

11. **What is a working set in the context of process memory management?**
The working set is the set of pages a process has accessed recently. It approximates the process's current locality of reference. The OS uses it to determine which pages to keep in memory to reduce page faults.

12. When do detached threads become necessary? Explain.
Detached threads are used when a thread's return value is not needed, and the main thread does not need to wait for it to finish. They are necessary in scenarios where resources should be automatically released when the thread terminates, such as background tasks or event listeners.

13. A multithreaded program shows no speedup on a 16-core CPU? Explain using Amdahl's law equation only.
Amdahl's Law:
Speedup = 1 / ((1 - P) + (P / N))
Where:

- *P* = proportion of program that can be parallelized
- *N* = number of processors (16 in this case)

If there is no speedup, then P is very small or 0. For example, if P = 0 (no parallel part), then:
Speedup = 1 / (1) = 1, i.e., no speedup regardless of the number of cores.

14. A process of 450 KB needs to allocate a non-paged memory block. Given blocks of 500 KB, 300 KB, and 400 KB, which strategy (First Fit / Best Fit / Worst Fit) will successfully allocate memory, and what will be the remaining space?

- **First Fit**: Allocates 450 KB in 500 KB block. Remaining = 50 KB
- **Best Fit**: Chooses 500 KB (smallest sufficient block). Remaining = 50 KB
- **Worst Fit**: Chooses 500 KB (largest block). Remaining = 50 KB

All three will allocate from the 500 KB block and leave 50 KB remaining.

15. Logical address space = 2048 pages, Page size = 4 KB, Physical memory = 512 frames

a. **Logical address bits** = $\log_2$(2048 pages) + $\log_2$(4 KB) = 11 + 12 = **23 bits**
b. **Physical address bits** = $\log_2$(512 frames) + $\log_2$(4 KB) = 9 + 12 = **21 bits**

16. Paging with TLB, 75% hits, memory reference time = 50 ns

Effective Memory Access Time (EMAT) =
(TLB hit rate × (TLB access + Memory access)) + (Miss rate × (TLB access + 2 × Memory access))

Assuming TLB access = negligible (≈0), EMAT =
= 0.75 × 50 + 0.25 × 100 = **37.5 + 25 = 62.5 ns**

17. In a multithreaded program solving a critical-section problem, some threads experience unexplained delay. Diagnose the issue(s).
Possible issues:

- **Deadlock**: Threads are waiting indefinitely for resources.
- **Starvation**: Some threads never get access to critical section due to scheduling.

- **Priority inversion**: Lower-priority thread holding resource needed by a higher-priority one.

18. How is a frame allocation problem different from a page replacement problem?

- **Frame allocation**: Decides how many frames each process gets.
- **Page replacement**: Decides which page to remove when a new page must be loaded and memory is full.

19. How does circular wait happen in the Dining Philosopher problem? Explain using a diagram.
Circular wait occurs when each philosopher holds one fork and waits for the next, forming a cycle.

```
P1 → fork1 → waiting for fork2 → held by P2
P2 → fork2 → waiting for fork3 → held by P3
...
P5 → fork5 → waiting for fork1 → held by P1
```

All philosophers are waiting; no one can proceed → deadlock.

20. How does an inverted page table work?
Unlike traditional page tables that map virtual to physical pages per process, an **inverted page table** has one entry per physical frame and stores:

- Virtual page number
- Process ID
- Control bits

To find a mapping, a search is done over the table to match the virtual page and PID to find the frame.