Concurrent Processes Notes for AKTU Semester Exam

samir suman

Prepared for AKTU Semester Exam

Unit-2

Contents

1 Process Concept

- **Definition**: A process is a program in execution, encompassing its code, data, and system resources.
- **Components**:
  - **Code Segment**: The executable instructions.
  - **Data Segment**: Global and static variables.
  - **Stack**: Temporary storage for function calls.
  - **Heap**: Dynamically allocated memory.
  - **Process Control Block (PCB)**: Stores process state, program counter, registers, and scheduling information.
- **States**: New, Ready, Running, Waiting, Terminated.
- **Significance**: Processes are units of execution managed by the OS for resource allocation and scheduling.

2 Principle of Concurrency

- **Definition**: Concurrency is the ability of multiple processes or threads to execute simultaneously, improving system efficiency.
- **Key Principles**:
  - **Parallel Execution**: Processes run on multiple CPUs or cores.
  - **Interleaved Execution**: Processes share a single CPU via time-slicing.
  - **Shared Resources**: Processes access common resources (e.g., memory, files), requiring synchronization.
- **Advantages**:
  - Improved CPU utilization and throughput.
  - Faster response times for multiple tasks.
- **Challenges**:
  - Race conditions: Uncontrolled access to shared resources.
  - Deadlocks: Processes waiting indefinitely for resources.
  - Synchronization overhead.

3 Producer/Consumer Problem

- **Definition**: A classic concurrency problem where producers generate data and place it in a shared buffer, and consumers retrieve data from it.
- **Components**:
  - **Producer**: Generates data and adds it to the buffer.
  - **Consumer**: Removes and processes data from the buffer.
  - **Buffer**: Fixed-size storage for data, requiring synchronization.
- **Challenges**:
  - Buffer overflow: Producer adds to a full buffer.
  - Buffer underflow: Consumer removes from an empty buffer.
  - Race conditions: Concurrent access to the buffer.
- **Solution**: Use synchronization mechanisms like semaphores or monitors to ensure mutual exclusion and proper buffer management.
- **Diagram**:

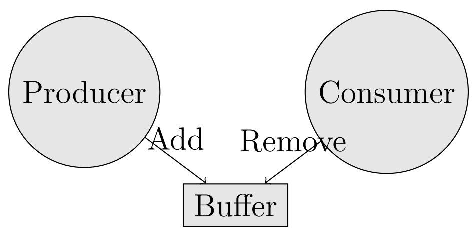

Figure 1: Producer/Consumer Problem

4 Mutual Exclusion

- **Definition**: Mutual exclusion ensures that only one process or thread accesses a shared resource at a time to prevent race conditions.
- **Requirements**:
  - Only one process can enter its critical section at a time.
  - No process should be forced to wait unnecessarily.
  - No process outside the critical section should block others.
- **Mechanisms**: Locks, semaphores, monitors, and atomic operations like Test and Set.
- **Significance**: Prevents data corruption in shared resources like memory or files.

5 Critical Section Problem

- **Definition**: The critical section is a part of a program accessing shared resources, requiring mutual exclusion to avoid conflicts.
- **Structure**:
  - **Entry Section**: Requests access to the critical section.
  - **Critical Section**: Executes operations on shared resources.
  - **Exit Section**: Releases the critical section.
  - **Remainder Section**: Non-critical code.
- **Solution Requirements**:
  - **Mutual Exclusion**: Only one process in the critical section.
  - **Progress**: No unnecessary waiting for entry.
  - **Bounded Waiting**: Finite waiting time for processes.

6 Dekkers Solution

- **Definition**: Dekkers algorithm is a software solution for mutual exclusion between two processes, ensuring no race conditions.
- **Working**:
  - Uses shared variables: `turn` (indicating which process can enter) and `want`$_t o_e nter[2](indicatin$
  - If conflict occurs, a process waits until the other releases the critical section.
- **Algorithm (for Process 0)**:

```
want_to_enter[0] = true;
while (want_to_enter[1]) {
    if (turn != 0) {
        want_to_enter[0] = false;
        while (turn != 0);
        want_to_enter[0] = true;
    }
}
// Critical Section
turn = 1;
want_to_enter[0] = false;
```

- **Advantages**: Satisfies mutual exclusion, progress, and bounded waiting.
- **Disadvantages**: Complex for more than two processes, busy waiting.

7 Petersons Solution

- **Definition**: Petersons algorithm is a simpler software solution for mutual exclusion between two processes.
- **Working**:
  - Uses shared variables: `turn` (whose turn to enter) and `interested[2]` (intent to enter).
  - A process sets its interest and yields turn to the other, waiting if necessary.
- **Algorithm (for Process i)**:

```
interested[i] = true;
turn = j; // j is the other process
while (interested[j] && turn == j);
// Critical Section
interested[i] = false;
```

- **Advantages**: Simple, ensures mutual exclusion, progress, and bounded waiting.
- **Disadvantages**: Limited to two processes, involves busy waiting.

8 Semaphores

- **Definition**: A semaphore is a synchronization tool used to control access to shared resources or signal events.
- **Types**:
  - **Binary Semaphore**: Values 0 or 1, used for mutual exclusion.
  - **Counting Semaphore**: Integer value, used for resource counting or signaling.
- **Operations**:
  - **Wait (P)**: Decrements semaphore; blocks if value is 0.
  - **Signal (V)**: Increments semaphore; wakes a waiting process.
- **Example (Producer/Consumer with Semaphores)**:

```
semaphore mutex = 1, full = 0, empty = N;
Producer() {
    while (true) {
        wait(empty); wait(mutex);
        // Add item to buffer
        signal(mutex); signal(full);
    }
}
Consumer() {
    while (true) {
```

```
        wait(full); wait(mutex);
        // Remove item from buffer
        signal(mutex); signal(empty);
    }
}
```

- **Advantages**: Flexible, supports multiple processes, avoids busy waiting (if implemented with blocking).
- **Disadvantages**: Incorrect usage can lead to deadlocks or starvation.

9 Test and Set Operation

- **Definition**: Test and Set (TS) is a hardware-supported atomic operation to achieve mutual exclusion.
- **Operation**:
  - Atomically tests a boolean variable and sets it to true.
  - Returns the original value of the variable.
- **Algorithm**:

```
boolean lock = false;
TestAndSet(boolean &target) {
    boolean old = target;
    target = true;
    return old;
}
while (TestAndSet(lock)); // Wait until lock is false
// Critical Section
lock = false;
```

- **Advantages**: Simple, hardware-supported, ensures atomicity.
- **Disadvantages**: Busy waiting, limited to mutual exclusion.

10 Dining Philosopher Problem

- **Definition**: A classic concurrency problem where five philosophers sit at a round table, each needing two forks (shared resources) to eat.
- **Problem**:
  - Each philosopher alternates between thinking and eating.
  - Forks are placed between philosophers; each needs the left and right fork to eat.
  - Challenges: Deadlock (all grab one fork), starvation, and mutual exclusion.