Lecture#10 Critical Section Problem

Introduction to Critical Section

- Critical Section is a segment of code where shared resources are accessed
- Only one process can execute in critical section at a time
- Ensures data consistency and prevents race conditions
- Example: Multiple ATM transactions accessing same bank account simultaneously

Three Essential Conditions

- Mutual Exclusion:
  - Only one process can execute in critical section at a time
  - Other processes must wait until critical section is free
- Progress:
  - Selection of process cannot be postponed indefinitely
  - If no process is in critical section, a requesting process must be allowed to enter
- Bounded Waiting:
  - Limit exists on number of times other processes enter critical section
  - Prevents starvation of waiting processes

Algorithm 1: Two Process Solution (Shared Variable)

- Uses shared variable 'turn'

Implementation: P1

```
while (1) {
while (turn != 1); // wait
critical_section();
turn = 1;
remainder_section();
}
```

- Satisfies mutual exclusion
- Does not satisfy progress requirement
- Processes must strictly alternate

- Uses shared variable 'turn'

Implementation: P2

```
while (1) {
while (turn != 0); // wait
critical_section();
turn = 0;
remainder_section();
}
```

- Satisfies mutual exclusion
- Does not satisfy progress requirement
- Processes must strictly alternate

Limitations of Algorithm 1

- Strict alternation requirement
- Process must enter critical section even if it doesn't need to
- If one process is slower, both processes slow down
- Progress condition violation
- Not suitable for real-world applications

Algorithm 2: Flag-Based Solution

Implementation: P1

```
While(1){
flag[i] = true;
while (flag[j]); // Value of Flag[j] will not be changed
critical_section();
flag[i] = false;
Reminder section;
}
```

- Satisfies mutual exclusion
- May lead to deadlock if both processes set flags simultaneously

Implementation: P2

```
While(1){
flag[i] = true;
while (flag[j]); // wait
critical_section();
flag[i] = false;
Reminder section;
}
```

Peterson's Solution: Overview

- Combines advantages of previous algorithms
- Uses both flag array and turn variable
- Elegant solution that satisfies all three conditions
- Most significant improvement in critical section problem

Peterson's Solution: Implementation

Implementation: P1 (Flag[i]=False)

```
While(1) {
flag[i] = true;
turn = j;
while (flag[j] && turn == j);
        critical_section();
flag[i] = false;
Reminder section;
}
```

Implementation: P2 (Flag[j]=False)

```
While(1) {
flag[j] = true;
turn = i;
while (flag[i] && turn == i);
        critical_section();
flag[j] = false;
Reminder section;
}
```

Analysis of Peterson's Solution

- Mutual Exclusion: Guaranteed by flag and turn combination
- Progress: No indefinite postponement
- Bounded Waiting: Process can wait at most once before entry
- Software solution requiring no special hardware support
- Suitable for modern systems with proper memory barriers

Summary and Applications

- Critical Section problem fundamental to concurrent programming
- Three conditions must be satisfied for valid solution
- Evolution of solutions:
  - Shared variable approach
  - Flag-based method
  - Peterson's complete solution
- Forms basis for more complex synchronization mechanisms
- Essential for understanding modern operating systems