**CSCI 5103**  
**Operating Systems**  

Instructor: Abhishek Chandra

---

**Producer-Consumer Problem**

- Two kinds of entities:
  - Producers: Generate new objects
  - Consumers: Consume these objects
- Producers and consumers running concurrently
  - May be running at different speeds
- Intermediate buffer
  - A data structure containing objects

---

**Synchronization**

- Producer-Consumer Problem
- Synchronization Problem
- Critical Section

---

**Producer:**

```c
Buffer_type buffer;
while(1)
{
    produce(item);
    insert(item, buffer);
}
```

**Consumer:**

```c
while(1)
{
    remove(item, buffer);
    consume(item);
}
```

- Buffer can be bounded or unbounded
  - Bounded => buffer size is fixed
Bounded Buffer Problem

```c
item buffer[BUF_SIZE];
int in = out = 0;
```

#### Producer:
```c
while(1){
    produce(item);
    while((in+1)%BUFSIZE==out)
        /* do nothing */;
    buffer[in] = item;
    in = (in+1)%BUF_SIZE;
}
```

#### Consumer:
```c
while(1){
    while (in==out)
        /* do nothing */;
    item = buffer[out];
    out = (out+1)%BUF_SIZE;
    consume(item);
    counter++;
}
```

- Problem: Buffer holds max BUF_SIZE-1 items

----

#### What is the value of counter?

```c
what is the value of counter?
```

#### Producer:
```c
while(1){
    produce(item);
    while (counter==BUFSIZE)
        /* do nothing */;
    buffer[in] = item;
    in = (in+1)%BUF_SIZE;
    consume(item);
    counter++;
}
```

#### Consumer:
```c
while(1){
    while (counter==0)
        /* do nothing */;
    item = buffer[out];
    out = (out+1)%BUF_SIZE;
    counter--;
    consume(item);
}
```

---

What is the value of counter?

```c
what is the value of counter
```

#### Producer:
```c
counter++:
1a reg1=counter
2a reg1=reg1+1
3a counter=reg1
```

#### Consumer:
```c
counter--:
1b reg2=counter
2b reg2=reg2-1
3b counter=reg2
```
Race Condition

- Situation in which outcome is dependent on the order of execution
  - Concurrent threads/processes
  - Accessing and modifying a shared resource
  - Some of the outcomes may be undesirable
    - Wrong data value
    - Corrupt data structures

Race Condition: Example

Thread A:
```c
add(list,node) {
  1   p = list->next;
  2   list->next = node;
  3   node->next = p;
}
```

Thread B:
```c
delete(list) {
  4   q = list->next;
  5   list->next = q->next;
  6   q->next = NULL;
}
```

- What does the linked list look like after executing threads A and B?

Synchronization Problem

- Concurrency
  - Multiple processes or threads running in parallel
  - Execution can be interleaved
  - No guarantees on order or speed of execution
- Data Sharing
  - Common data could be accessed/modified by multiple activities
  - E.g.: Global variables, memory buffers, files
- Synchronization problem:
  - When we have both concurrency and data sharing
  - Ensure consistent state of shared data

Critical Section

- Code segment that modifies shared data
- Requirement: Only one process should execute in the critical section at a time
- All executions of critical section are sequentialized
  - No concurrent execution
- Examples:
  - Code modifying shared variable
  - Code updating a shared data structure
  - Code writing to a shared file
General Structure of a Process

- **Entry section**: Request exclusive access
- **Critical section**: Only one process can execute
- **Exit section**: Give up exclusive access
- **Remainder section**: Concurrent execution: Any number of processes allowed

Critical Section Problem

- Design a protocol that allows processes to synchronize access to the critical section
- Three Requirements:
  - Mutual exclusion
  - Progress
  - Bounded waiting

Mutual Exclusion

- Only one process should execute in critical section
  - One-at-a-time access to the shared data
  - All other processes must wait their turn
- Examples:
  - Only one process should modify shared variable (e.g., counter, list)
  - Only one process should write to a shared file

Progress

- If some processes request entry to critical section
  - Only processes not in their remainder sections should participate in the selection
  - This selection cannot be postponed indefinitely
  - Some process should be allowed to enter the critical section if nobody is executing there
**Bounded Waiting**

- Once a process requests to enter the critical section
- There is a bound on the number of times other processes can enter the critical section
- No starvation

**Critical Section Problem: Software Solution**

- Software-based solution to critical section problem
  - Is it possible?
  - Scenario: Two processes P0 and P1
  - Alternate between critical and remainder section

**Critical Section Solution: Try 1**

```c
int turn;

Process Pi:
while(1)
{
    turn=j;
    while (turn == j) /* loop */;
    Critical Section
    Remainder Section
}
```

**Critical Section Solution: Try 2**

```c
boolean flag[2];

Process Pi:
while(1)
{
    flag[i] = TRUE;
    while (flag[j]) /* loop */;
    Critical Section
    flag[i] = FALSE;
    Remainder Section
}
```
Peterson’s Solution

```c
int turn;
boolean flag[2];
```

Process Pi:
while(1)
{
    flag[i] = TRUE;
    turn=j;
    while (flag[j] && turn == j):
        Critical Section
    flag[i] = FALSE;
        Remainder Section
}

Does it satisfy the three requirements of critical section problem?

- Problems:
  - Extension to n processes is not efficient
  - Relies on busy waiting