Concurrency and Resource Sharing

- Concurrency
  - Multiple activities happening in parallel
  - E.g.: Multiple threads or processes

- Resource Sharing
  - Common resources could be accessed/modified by multiple activities
  - E.g.: Global variables, memory buffers, files, pipes

- What happens when we have both concurrency and resource sharing?

Example

```c
counter=1;
Thread A:
  counter++;
Thread B:
  counter--;
```

- What is the value of counter after executing both threads A and B?
**Example**

```
Thread A: counter++:
1a reg1=counter
2a reg1=reg1+1
3a counter=reg1

Thread B: counter--:
1b reg2=counter
2b reg2=reg2-1
3b counter=reg2
```

- What is the value of `counter` after executing both threads A and B?
  - If we execute: 1a -> 1b-3b -> 2a-3a
  - If we execute: 1b -> 1a-3a -> 2b-3b
  - If we execute: 1a-3a -> 1b-3b

**Example 2**

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

Thread B: 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?
  - If we execute: 1 -> 4-6 -> 2-3
  - If we execute: 4 -> 1-3 -> 5-6
  - If we execute: 1-3 -> 4-6

**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 values
    - Corrupt data structures

**Synchronization**

- Controlling the order between multiple threads/processes
  - For accessing a shared resource
  - For executing a code segment
- Requires
  - Some way of protecting shared resources
  - Some signaling mechanism between threads
Synchronization: Concepts

- Atomic operations
  - Non-divisible operations
- Mutual exclusion
  - One-at-a-time access to a shared resource or code segment
- Critical sections
  - Code segment that should not be executed concurrently

Atomic Operations

- An operation that is indivisible
  - Would not be “interrupted”
  - Would either be executed completely or not at all
- Examples:
  - Hardware instructions
  - Software constructs
    - Atomic from program perspective
    - Could consist of multiple instructions
    - Require OS, language or library support

Mutual Exclusion

- One-at-a-time access to a resource or a code segment
- Only one thread given access
- All other threads must wait their turn
- Examples:
  - Only one thread should modify shared variable (e.g., counter, list)
  - Only one thread should print a file at a time

Critical Section

- Code segment that requires mutual exclusion
- Only one thread can execute critical section at a time
- All executions of critical section are sequentialized
  - No concurrent execution
- Examples:
  - Code modifying shared variable (e.g., add, delete)
  - Code for printing file
Controlling access to a Critical Section

- Entry section: Allow only one thread to proceed
- Critical section: Mutual exclusion; only one thread can execute
- Exit section: Give up exclusive access
- Remainder section: Concurrent execution; any number of threads allowed

Producer-Consumer Problem

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

Examples?

Summary

- Conflict and Cooperation
  - Race conditions
- Synchronization Concepts
  - Atomic operations
  - Mutual exclusion
  - Critical sections

Producer-Consumer Problem

```c
Buffer_type buffer;

Producer:
while(1)
{
    produce(item);
    put(item, buffer);
}

Consumer:
while(1)
{
    item = get(buffer);
    consume(item);
}
```

- What's the critical section?
- Question: How do we protect the critical section?