**CO3**

**PROCESS SYNCHORNIZATION**

A cooperating process is one that can affect or be affected by other processes executing in the system. Cooperating processes can either directly share a logical address space (that is, both code and data) or be allowed to share data only through files or messages. Concurrent access to shared data may result in data inconsistency. Various mechanisms to ensure the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

**Example:** Producer consumer problem

To add an integer variable counter, initialized to 0. counter is incremented every time we add a new item to the buffer and is decremented every time, we remove one item from the buffer. The code for the producer process can be modified as follows:

while (true) { /\* produce an item in next produced \*/

while (counter == BUFFER SIZE)

; /\* do nothing \*/

buffer[in] = next\_produced;

in = (in + 1) % BUFFER SIZE;

counter++;

}

The code for the consumer process can be modified as follows:

while (true) { while (counter == 0)

; /\* do nothing \*/

Next\_consumed = buffer[out];

out = (out + 1) % BUFFER SIZE;

counter--;

/\* consume the item in next consumed \*/

}

The producer and consumer routines shown above are correct separately, they may not function correctly when executed concurrently. Suppose that the value of the variable counter is currently 5 and that the producer and consumer processes concurrently execute the statements “counter++” and “counter--”. Following the execution of these two statements, the value of the variable counter may be 4, 5, or 6! The only correct result, though, is counter == 5, which is generated correctly if the producer and consumer execute separately.

The statement “counter++” may be implemented in machine language as follows:

register1 = counter

register1 = register1 + 1

counter = register1

where register1 is one of the local CPU registers.

Similarly, the statement “counter--” is implemented as follows:

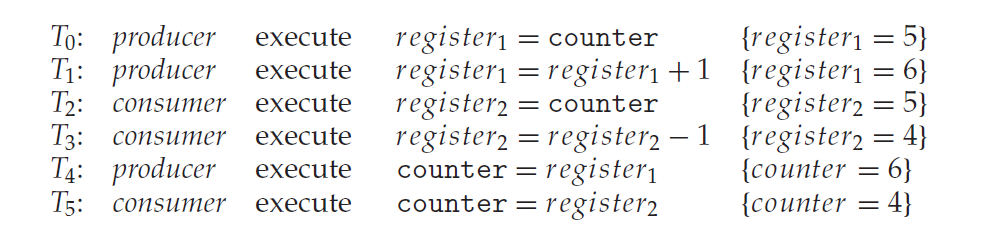
register2 = counter

register2 = register2 − 1

counter = register2

where register2 is one of the local CPU registers.

The concurrent execution of “counter++” and “counter--” is



we have arrived at the incorrect state “counter == 4”, indicating that four buffers are full, but it is five buffers are full. If we reversed the order of the statements at T4 and T5, we would arrive at the incorrect state “counter == 6”. We would arrive at this incorrect state because we allowed both processes to manipulate the variable counter concurrently.

We would arrive at this incorrect state because we allowed both processes to manipulate the variable counter concurrently. A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a race condition. To guard against the race condition above, we need to ensure that only one process at a time can be manipulating the variable counter. To make such a guarantee, we require that the **processes be synchronized**.