Department of Computer Science and Engineering

FACULTY OF ENGINEERING AND TECHNOLOGY
UNIVERSITY OF LUCKNOW
LUCKNOW

Dr. Zeeshan Ali Siddiqui
Assistant Professor
Deptt. of C.S.E.

PROCESS SYNCHRONIZATION

(PRODUCER–CONSUMER PROBLEM)

Concept of Concurrency

- *A system is parallel if it can perform more than one task simultaneously.* In contrast, *a concurrent system supports more than one task by allowing all the tasks to make progress*. Thus, it is possible to have concurrency without parallelism.

- In single processor. CPU schedulers were designed to provide the *illusion* of *parallelism* by rapidly switching between processes in the system, thereby allowing each process to make *progress*. Such processes were running concurrently, but not in parallel.

- Concurrent access to shared data may result in data *inconsistency*

- Maintaining data consistency requires *mechanisms* to ensure the orderly execution of cooperating processes.

Producer–Consumer Problem

- *A producer process produces information that is consumed by a consumer process.*

- For example:
  - A compiler may produce assembly code that is consumed by an assembler.
  - The assembler, in turn, may produce object modules that are consumed by the loader.

Producer–Consumer Problem: Solution 1/3

- One *solution* to the producer–consumer problem uses *shared memory*.
- To allow producer and consumer processes to run concurrently, we must have available *a buffer of items that can be filled by the producer and emptied by the consumer*.
- This buffer will reside in a region of memory that is *shared* by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be *synchronized*, so that the consumer does not try to consume an item that has not yet been produced.

Producer–Consumer Problem: Solution 2/3

- producer process:

```
while (true) {
     /* produce an item in next_produced */

     while (counter == BUFFER_SIZE)
        ; /* do nothing */

     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
     counter++;
}
```

Producer–Consumer Problem: Solution 3/3

- consumer process:

```
while (true) {
        while (counter == 0)
            ; /* do nothing */

        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        counter--;

        /* consume the item in next_consumed */
}
```

Analysis 1/3

- When executing concurrently the value of counter may be *incorrect* as follows:

- The statement "counter++" may be implemented in machine language (on a typical machine) as follows:

$$register_1 = \texttt{counter}$$
$$register_1 = register_1 + 1$$
$$\texttt{counter} = register_1$$

- Similarly, the statement "counter--" is implemented as follows:

$$register_2 = \texttt{counter}$$
$$register_2 = register_2 - 1$$
$$\texttt{counter} = register_2$$

Analysis 2/3

- Consider this execution interleaving with "count = 5" initially:

T_0:	*producer*	execute	$register_1 = \texttt{counter}$	$\{register_1 = 5\}$
T_1:	*producer*	execute	$register_1 = register_1 + 1$	$\{register_1 = 6\}$
T_2:	*consumer*	execute	$register_2 = \texttt{counter}$	$\{register_2 = 5\}$
T_3:	*consumer*	execute	$register_2 = register_2 - 1$	$\{register_2 = 4\}$
T_4:	*producer*	execute	$\texttt{counter} = register_1$	$\{counter = 6\}$
T_5:	*consumer*	execute	$\texttt{counter} = register_2$	$\{counter = 4\}$

- Notice that we have arrived at the incorrect state "counter == 4", indicating that four buffers are full, when, in fact, five buffers are full.

- Suppose, If we reversed the order of the statements at T4 and T5, we would arrive at the incorrect state "counter == 6".

Analysis 3/3

- 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 in some way.

References

1. Silberschatz, Galvin and Gagne, “Operating Systems Concepts”, Wiley.

2. William Stallings, “Operating Systems: Internals and Design Principles”, 6th Edition, Pearson Education.

3. D M Dhamdhere, “Operating Systems: A Concept based Approach”, 2nd Edition, TMH.

Thank You.