Computer System Software Course

CE 403

Dr. Mohamed Abd El-Latif Mowad
Lecture- 4

Producer- Consumer Problem- and Buffering

Dr. Mohamed Abd El-Latif Mowad
Email: Turbocode2000@yahoo.com
Cell Phone: 01007532006
• Cooperative processes require an inter-process communication (IPC) mechanism that will allow them to exchange data and information.

• There are two fundamental modes of (IPC). Shared Memory and Message Passing can be used for this purpose.

• We know that: one of the O.S functions is to prevent one process from accessing another process’s memory.

• Shared memory requires that two or more processes agree to remove this restriction, they can exchange information by reading and writing data in the shared areas.
• Data form and location are determined by processes not under control of O.S.
• Processes are also responsible for ensuring that they are not writing to the same location simultaneously. To illustrate the concept of cooperative processes let us consider the consumer–producer problem.
• A producer produces an information that is consumed by the consumer. The producer–consuming problem can be described using Client-Server example.
• We always think of a server as a producer and a client as a consumer for example: a web server produces (provides) HTML files and images which are consumed (read) by the client web browser.
One solution for the producer – consumer problem which uses a shared memory is to allow both producer and consumer processes to run concurrently, we must have available 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.

The producer and consumer must be synchronized, such that consumer does not try to consume an item that has not yet been produced.
We Have Two Buffer Types:

a- Unbounded Buffer

No practical limits on the size, the consumer may have to wait for new items but the producer can always produce new items.

b- Bounded Buffer (fixed buffer size)

In this case the consumer must wait if the buffer is empty and the producer must wait if the buffer is full.
The O-S provides the means of cooperative processes to communicate with each other via message-passing facility. Message passing is useful in distributed environment where the communicating processes may reside on different computers connected by a network such as Chat programs used in World Wide Web (WWW) such as yahoo messenger which can be designed so that the participants communicate with one another by exchanging messages.
A **Message passing** facility provides at least two operations (**Send** message and (**Receive**) message.

If processes (P)&(Q) want to communicate they must send and receive messages from each other. A **communication link** must be exist.

The communication link can be implemented in a variety of ways. **We did not concerned her** with **Physical Implementation** but in communication link **Logical Implementation**.

**There are several methods** for **logically implemented a link** and the **Send** (/) / **Receive** (/) operations:
Direct or Indirect Communication

Under **direct communication** each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send ( ) and receive ( ) primitives are defined as:

1. **Send** ( p, message) means **Send** a message to process (P).
   
2. **Receive** ( Q, message) means **Receive** a message from process (Q).
This scheme exhibits **Symmetry in addressing**, that is both the sender and receive processes must name the other to communicate.

A variant of this scheme employs **Asymmetry in addressing**. Here only the sender names the recipient, the recipient is not required to name the sender.

In this scheme the `send( )` and `receive( )` primitives are defined as:

1- `Send ( p, message) ...........................................` means
   Send a message to process (P).

2- `Receive(id,message).................................` means
   Receive a message from any process.
Indirect Communication

1- The messages are sent to and received from Mail-boxes.

2- A mail-box can be considered as an object into which messages can be placed by processes and from which messages can be removed.

3- Each mail-box has a unique identification, some message queues use an integer value to identify a mail-box. (example send E-Mail to many peoples)

4- Using indirect communication, process can communicate with some other processes not with another process via number of different mail-boxes. (E-Mail)
5-Two processes can communicate only if the two processes have **shared mailbox** (**yahoo messenger**). The `send()` and `receive()` primitives are defined as follows:

- **Send** (A, message)…send a message to mailbox A
- **Receive** (A, message) receive a message from mailbox A.

- **Message passing** may be either **Blocking** or **non-Blocking** also known as **Synchronous** and **Asynchronous**.
- **Blocking Send:**
  
  Sending process is blocked until the message is received by the receiving process. *(Acknowledgment)*
- Non-Blocking Send:
  Sending process sends the message and resumes operation.
- Blocking Receive:
  The receiver blocked until a message is available
- Non-Blocking Receive:
  The receiver retrieves either a valid message or a null

**Buffering**

Whether communication is direct or indirect, messages exchanged by communicating processes residue in a temporary queue. Basically such queues can be implemented in three ways:
1- **Zero Capacity:**

The queue has a maximum length of zero, thus the link can not have any messages waiting in it, in this case, the sender must block until the recipient receives the message.

2- **Bounded Capacity:**

The queue has finite length (n), almost (n) messages can residue in it.

3- **Unbounded Capacity:**

The queues length is potentially infinite, thus any number of messages can wait in it, the sender never blocks (waits).

Zero capacity is sometimes referred to as a message with no buffering, the other cases are referred to as systems with automatic buffering.