* 1. **Introduction about Inter Process Communication**

Processes can coordinate and interact with one another using a method called inter-process communication (IPC) . Through facilitating process collaboration, it significantly contributes to improving the effectiveness, modularity, and ease of software systems.

**1.2 Types of Process, Examples of IPC systems**

**Types of Process**

* Independent process
* Co-operating process

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when cooperative nature can be utilized for increasing computational speed, convenience, and modularity. Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of cooperation between them. Processes can communicate with each other through both:

**Methods of IPC:**

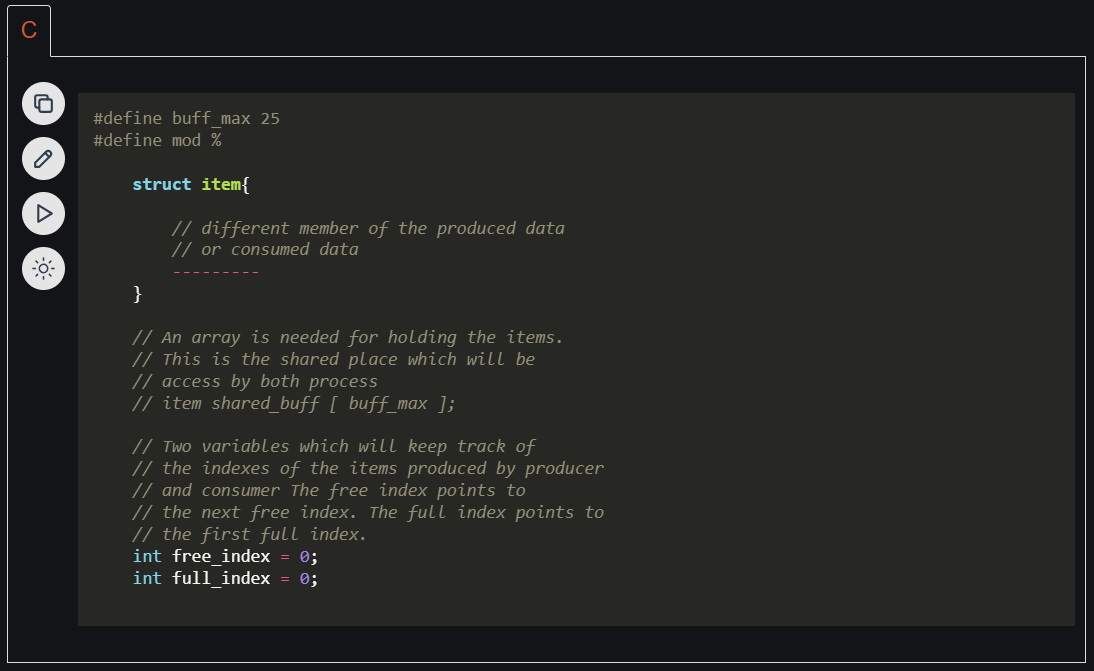
* Shared Memory
* Message Passing

**Shared Memory:**

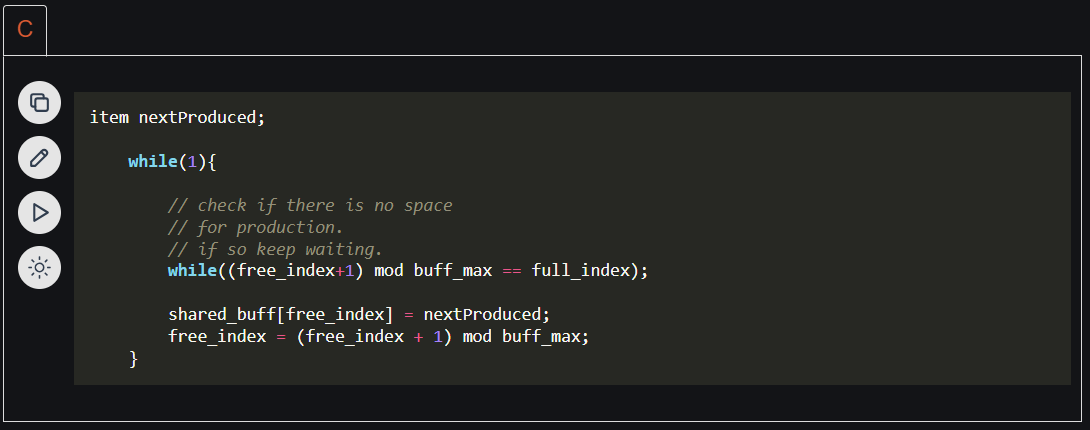
Ex: Producer-Consumer problem

There are two processes: Producer and Consumer . The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed. There are two versions of this problem: the first one is known as the unbounded buffer problem in which the Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which the Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then the producer will start producing items. If the total produced item is equal to the size of the buffer, the producer will wait to get it consumed by the Consumer. Similarly, the consumer will first check for the availability of the item. If no item is available, the Consumer will wait for the Producer to produce it. If there are items available, Consumer will consume them. The pseudo-code to demonstrate is provided below:

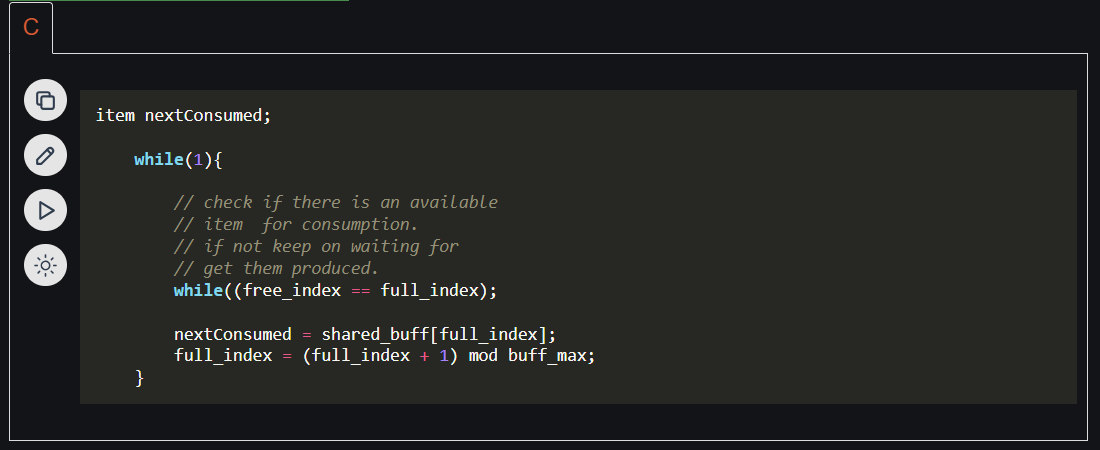
Shared Data Between the two Processes



Producer Process Code



Consumer Process Code



**Messaging Passing:**

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

* Establish a communication link (if a link already exists, no need to establish it again.)
* Start exchanging messages using basic primitives.

We need at least two primitives:

– send (message, destination) or send (message)

– receive (message, host) or receive (message)

The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: header and body. The header part is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

**Direct and Indirect Communication link**

A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity. In zero capacity, the sender waits until the receiver informs the sender that it has received the message. In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation, it can be either a direct communication link or an in-directed communication link.

Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time.

* For example the print server.

In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

**Synchronous and Asynchronous Message Passing**

A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. IPC is possible between the processes on same computer as well as on the processes running on different computer i.e. in networked/distributed system. In both cases, the process may or may not be blocked while sending a message or attempting to receive a message so message passing may be blocking or non-blocking. Blocking is considered synchronous and blocking send means the sender will be blocked until the message is received by receiver. Similarly, blocking receive has the receiver block until a message is available. Non-blocking is considered asynchronous and Non-blocking send has the sender sends the message and continue. Similarly, Non-blocking receive has the receiver receive a valid message or null. After a careful analysis, we can come to a conclusion that for a sender it is more natural to be non-blocking after message passing as there may be a need to send the message to different processes. However, the sender expects acknowledgment from the receiver in case the send fails. Similarly, it is more natural for a receiver to be blocking after issuing the receive as the information from the received message may be used for further execution. At the same time, if the message send keep on failing, the receiver will have to wait indefinitely. That is why we also consider the other possibility of message passing.

There are basically three preferred combinations:

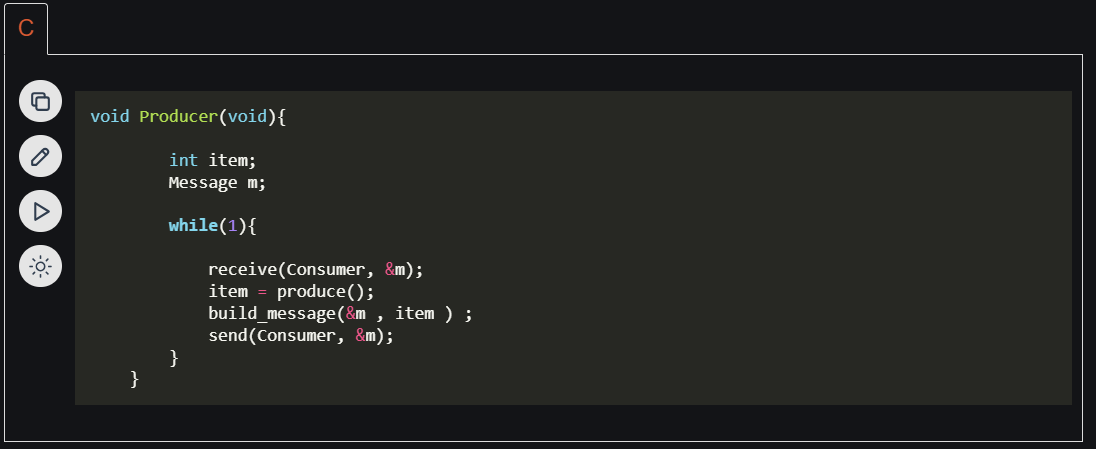
* Blocking send and blocking receive
* Non-blocking send and Non-blocking receive
* Non-blocking send and Blocking receive (Mostly used)

Ex: send(p1, message) means send the message to p1.

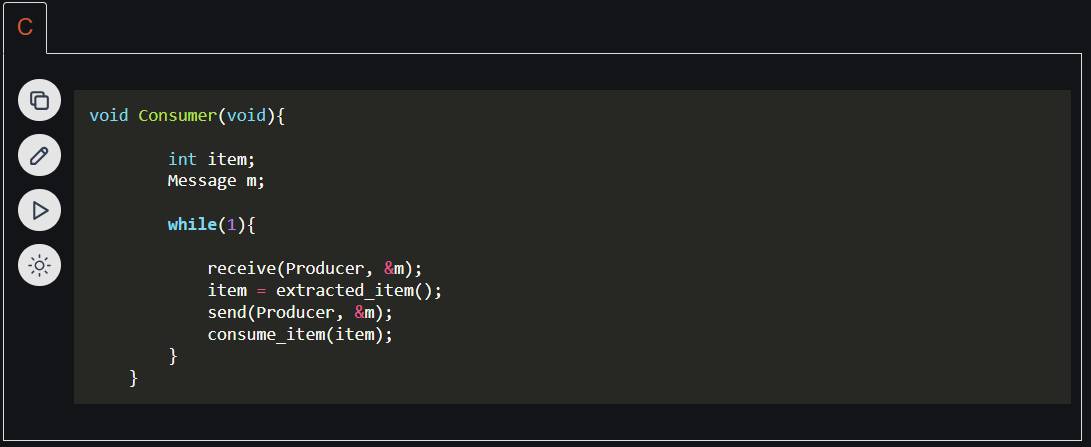
Similarly, receive(p2, message) means to receive the message from p2.

In this method of communication, the communication link gets established automatically, which can be either unidirectional or bidirectional, but one link can be used between one pair of the sender and receiver and one pair of sender and receiver should not possess more than one pair of links. Symmetry and asymmetry between sending and receiving can also be implemented i.e. either both processes will name each other for sending and receiving the messages or only the sender will name the receiver for sending the message and there is no need for the receiver for naming the sender for receiving the message. The problem with this method of communication is that if the name of one process changes, this method will not work.

Producer Code



Consumer Code



Examples of IPC systems

* Posix : uses shared memory method.
* Mach : uses message passing
* Windows XP : uses message passing using local procedural calls

Communication in Client/Server Architecture

There are various mechanism

* Pipe
* Socket
* Remote Procedural calls (RPCs)

The above three methods will be discussed in later articles as all of them are quite conceptual and deserve their own separate articles.

References:

* Operating System Concepts by Galvin et al.
* Lecture notes/ppt of Ariel J. Frank, Bar-Ilan University

Inter-process communication (IPC) is the mechanism through which processes or threads can communicate and exchange data with each other on a computer or across a network. IPC is an important aspect of modern operating systems, as it enables different processes to work together and share resources, leading to increased efficiency and flexibility.