L10: Inter process Communication, Shared memory solution to bounded buffer problem

Dr. Rajashree Dash

Associate Professor,
Department of CSE,
ITER, Siksha O Anusandhan Deemed to be University

Outline

1. Independent vs Cooperating Process
2. Interprocess communication
3. Shared memory Model
   - Producer-Consumer Problem
   - Shared-Memory Solution of Bounded-Buffer Producer Consumer Problem

Independent vs Cooperating Process

Processes executing concurrently in the operating system may be either independent processes or cooperating processes.

- A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.
- A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

Reason for process cooperation

There are several reasons for providing an environment that allows process cooperation:

- Information sharing: Since several users may be interested in the same piece of information , it is required to provide an environment to allow concurrent access to such information.
- Computation speed-up: To run a particular task faster,it can be broken into subtasks, each of which will be executing in parallel with the others. Such a speedup can be achieved only if the computer has multiple processing cores.
- Modularity: To construct the system in a modular fashion,the system functions can be divided into separate processes or threads.
- Convenience: Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music, and compiling in parallel.

Interprocess communication (IPC)

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: shared memory and message passing.

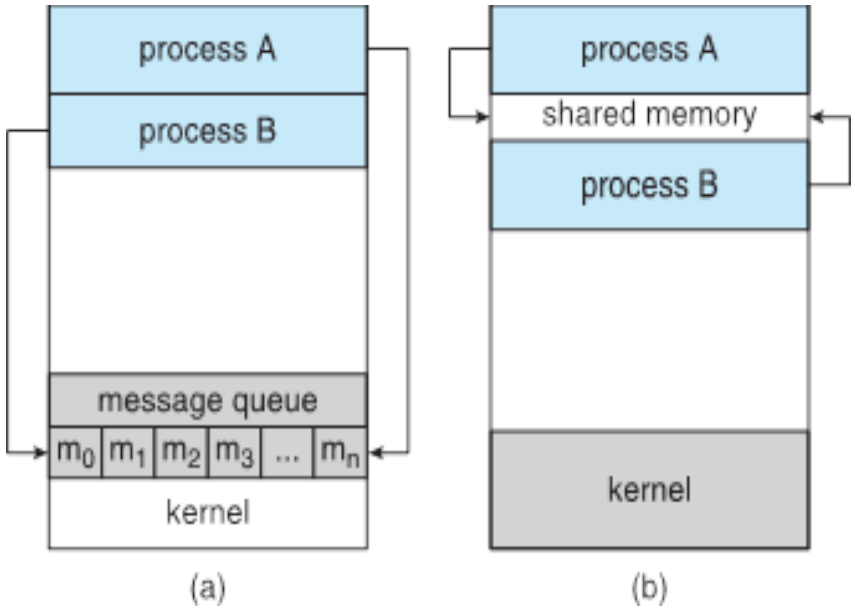

Figure: a)Message passing model b) Shared memory model

Interprocess communication (IPC)

Shared memory model	Message passing model
A region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.	Communication takes place by means of messages exchanged between the cooperating processes.
It is useful for exchanging large amounts of data.	It is useful for exchanging smaller amounts of data.
It allows max speed and convenience of communication, as it can be done at memory speed when within computer.	It is easier to implement in a distributed system than shared memory.
It is faster.	It is slower.

Shared memory model

- IPC using shared memory requires communicating processes to establish a region of shared memory.
- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space.
- Normally, the operating system tries 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 then exchange information by reading and writing data in the shared areas.

Shared memory model

- The form of the data and the location are determined by these processes and are not under the operating system's control.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.
- In shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required. So it is faster.

Producer-Consumer Problem

- Producer–consumer problem is a common paradigm for cooperating processes, in which producer process produces information that is consumed by a consumer process.
- To allow producer and consumer processes to run concurrently, there should be 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.

Types of buffer used in Producer-Consumer Problem

- Unbounded-buffer:

  It places no practical limit on the size of the buffer.

  The consumer may have to wait for new items, but the producer can always produce new items.

- Bounded-buffer:

  It assumes that there is a fixed buffer size.

  The consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Shared-Memory Solution of Bounded-Buffer Producer Consumer Problem

- The producer and consumer process use a shared buffer and two logical pointers in and out those are stored in the shared region.
- The shared buffer is implemented as a circular array.
- The variable in points to the next free position in the buffer; out points to the first full position in the buffer.
- Shared data used by producer and consumer process:

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Shared-Memory Solution of Bounded-Buffer Producer Consumer Problem

- The buffer is empty when in == out
- The buffer is full when

$$((in + 1)\%BUFFERSIZE) == out$$

Code of producer and consumer process

It allows at most BUFFER SIZE-1 items in the buffer.

Producer

```
item nextp;
while (true) {
    /* produce an item in nextp  */
  while (((in + 1) % BUFFER_SIZE) == out)
              ; /* do nothing */
      buffer[in] = nextp;
       in = (in + 1) % BUFFER_SIZE;
}
```

Consumer

```
item nextc;
while (true) {
      while (in == out)
              ; /* do nothing */
              nextc = buffer[out];
              out = (out + 1) % BUFFER_SIZE;
     /* consume the item in nextc */
}
```