OPERATING SYSTEMS

Module2_Part3

Textbook : Operating Systems Concepts by Silberschatz

Interprocess communication

Processes executing concurrently in the operating system may be

independent processes

cooperating processes.

A process is ***independent*** if

- it cannot affect or be affected by the other processes executing in the system.

-it does not share data with any other process .

A process is ***cooperating*** if

-it can affect or be affected by the other processes executing in the system.

-it shares data with other processes

Interprocess communication

- Reasons for providing an environment that allows process cooperation:

  Information sharing. Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.

  Computation speedup. If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.

  Modularity. We may want to construct the system in a modular fashion, dividing the system functions 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

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information.

Two fundamental models of interprocess communication:

- **shared memory**
- **message passing**.

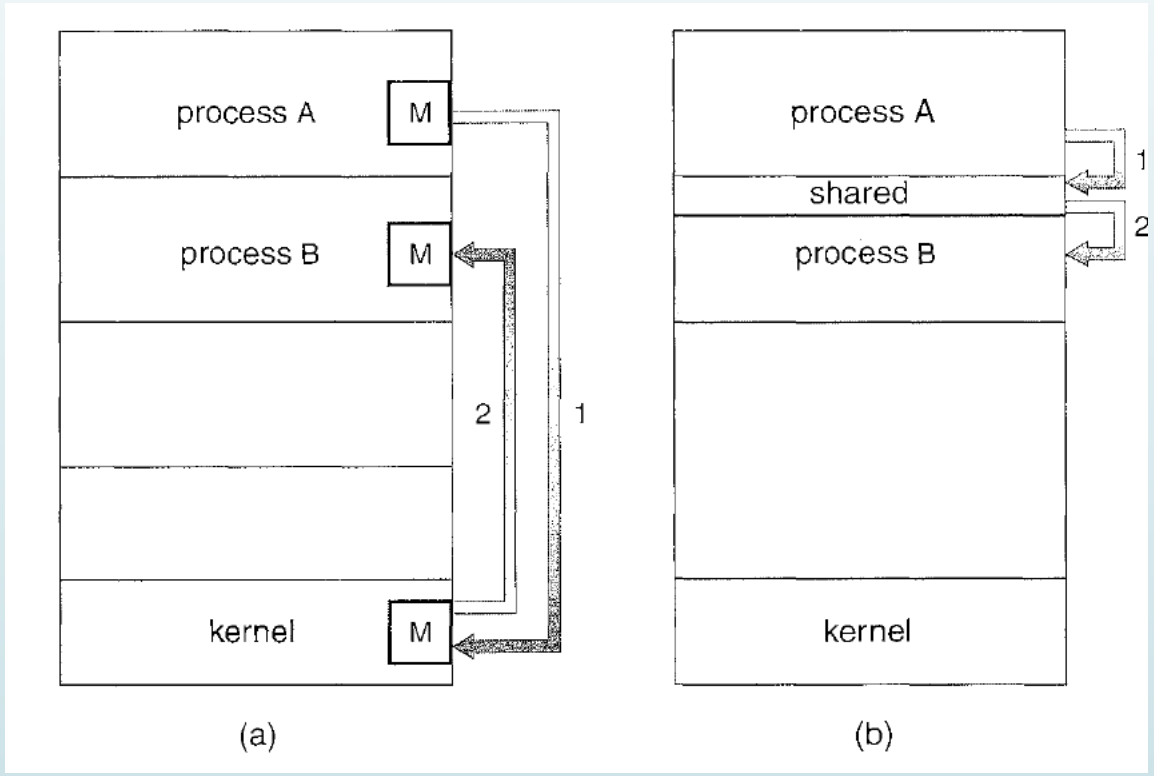
In the shared memory model

- 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.

In the message-passing model,

- communication takes place by means of messages exchanged between the cooperating processes.

Interprocess communication



a)Message passing b)shared memory

Abraham Silberschatzs

Shared memory and Message passing

- Message passing

  -is useful for exchanging smaller amounts of data.

  -is easier to implement than is shared memory for intercomputer communication.

  -are typically implemented using system calls and thus require the more time-consuming task of kernel intervention.

Shared memory allows

-maximum speed and convenience of communication.

-is faster than message passing, In contrast,

- system calls , 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.

Shared memory systems

- Interprocess communication 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.
- In shared memory system , cooperating processes can exchange information by reading and writing data in the shared areas.
- 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.

Shared memory systems

let's consider the producer-consumer problem, which is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process.

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.

The producer should produce data only when the buffer is not full.

Accessing memory buffer should not be allowed to producer and consumer at the same time.

Shared memory systems

Two types of buffers can be used.

unbounded buffer 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 assumes a 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.

Shared memory systems

Bounded buffer , inter process communication using shared memory.

The following variables reside in a region of memory shared by the producer and consumer processes:

Shared data

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

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

Bounded buffer , inter process communication using shared memory.

The shared buffer is implemented as a circular array with two logical pointers: **in** and **out**.

in points to the next free position in the buffer;

out points to the first full position in the buffer.

The buffer is empty when **in == out**;

the buffer is full when **((in + 1)% BUFFER SIZE) == out**.

Producer consumer problem using shared memory

The producer process has a local variable **next produced** in which the new item to be

produced is stored.

```
item next_produced;

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

The producer process using shared memory.

Producer consumer problem using shared memory

The consumer process has a local variable **next consumed** in which the item to be consumed is stored.

```
item next_consumed;

while (true) {
       while (in == out)
       ; /* do nothing */
       next_consumed = buffer[out];
       out = (out + 1) % BUFFER_SIZE;

       /* consume the item in next consumed */
}
```

The consumer process using shared memory.