Race Condition

A race condition is an undesirable situation that occurs when a system attempts to perform two or more operations at the same time, but because of the nature of the device or s/m, the operations must be done in the proper sequence to be done correctly.

↳ To sum up:

- multiple processes are operating on a shared data

- Final outcome depends on the order in which the processes run.

Consider the Program below:

```c
#include <stdio.h>
#include <unistd.h>
static void charatatime(char *str) {
        char *ptr;   int c;
        setbuf(stdout, NULL);
        for (ptr = str; (c = *ptr++) != 0; )
                putc(c, stdout);
}
int main() {
        pid_t pid;
        if((pid = fork()) < 0)
                printf("Fork Error\n");
        else if (pid == 0)
                charatatime("this is child\n");
        else
                charatatime("this is parent\n");
        return 0;
}
```

The output here can be: (one possible).

this is this is child
parent

We can use wait() to overcome the race condition situation here.

Interprocess Communication

A process can be

Independent: It cannot affect or be affected by the other processes executing in the system.

Cooperating: It can affect or be affected by the other processes executing in the system.
Any process that shares data with other process is a cooperating process.

Why allow process cooperation:

Information sharing. Several users may be interested in the same piece of information.

Computation speedup. Split a process into subtasks, to run faster, & each executes in parallel.

Modularity. We may want to build the system in a modular fashion, dividing the system functions into separate processes or threads.

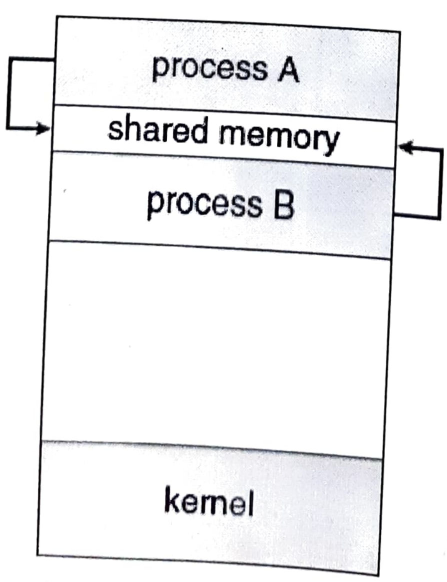
Convenience. Even an individual user may work on many tasks at the same time.

Co-operating processes require an interprocess communication. (IPC) To achieve, we have two fundamental models.

- Shared Memory.

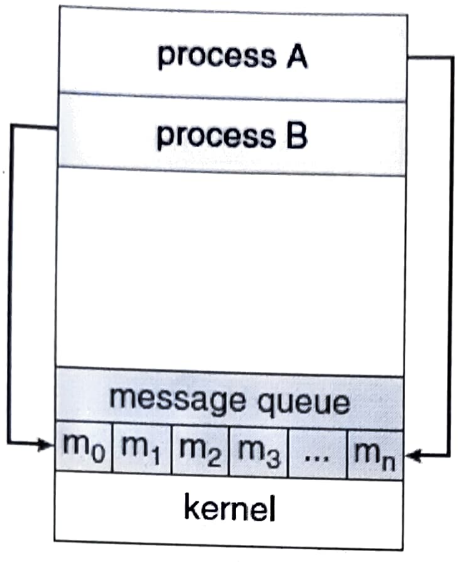
- Message Passing.

(shared memory can be faster as message passing is achieved typically using system calls)



Shared Memory Model

prakash hegade



Messaging Passing Model

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.

- Processes must agree to process management protocols.

-Example: Producer - Consumer Problem

a producer process produces information that is consumed by a consumer process.

↳ A compiler may produce assembly code consumed by assembler. Assembler produces object modules consumed by a loader.

Producer & Consumer interact through shared memory, or as we call it Buffer.

The Synchronization of producer & Consumer happens usually via a implementation of a Circular queue.

Unbounded Buffer: No practical limit on the size of the buffer

Bounded Buffer: Fixed buffer size

Message Passing

Provides atleast two operations:

- Send (message)
- receive (message)

Messages sent can be fixed (easy to implement) or variable in size.

In order to communicate, a communication link must exist between processes. This can be achieved by,

- Direct or indirect communication
- Synchronous or asynchronous Communication
- Automatic or explicit buffering

prakash begade

Naming

In direct communication:

send(P, message)

receive(Q, message)

we explicitly name whom to send or from whom to receive a message.

A communication link, exactly one, with exactly two processes, is established on the need for communication.

In Indirect Communication:

send (A, message)

receive (A, message)

A is a mailbox or port

Communication link may be associated with more than two processes, links will be in correspondence to mailbox associated & links are established only if they have a shared mailbox.

Synchronization

message passing can be
- blocking
- non-blocking

messages

- Blocking (sync)
  - Send
  - Receive
- Non-Blocking (async)
  - Send
  - Receive

Eg:- Blocking Send cannot Send next message until message is received.

A system can have various combinations of above.

Buffering:

Zero Capacity Queue length is 0. Must have blocking Send.

Bounded Capacity Queue has finite length n. Sender blocks when queue full.

Unbounded Capacity Queue length is potentially infinite. Sender never blocks.

prakash hegade

In Client-Server Systems IPC is usually implemen-ted using

- Sockets
- Remote Procedure Calls
- Pipes
- Remote method Invocation.

GATE Question:

The following two functions P1 and P2 that share a variable B with an initial value of 2 execute concurrently.

```
P1 () {                P2 () {
    C=B-1;                 D=2*B;
    B=2*C;                 B=D-1;
}                      }
```

The number of distinct values that B can possibly take after the execution is________.

Answer is 3. B can hold 3 different values

Can you justify?