Operating System:

Date: .....
Page: .....

An operating system is a set of programs which acts as an interface between computer hardware and user.

Operating system as resource manager

1. Process Management
   - Allocation of resources
   - Deallocation of resources

2. Memory Management
   - Keep record of all memory
   - Allocation of memory
   - Deallocation of memory
   - Manage primary & secondary memory

3. Device Management
   - Interface betn hardware & software
   - Allocation & Deallocation of I/O devices

4. File management
   Naming, Opening, Closing, create, Deleting, Access

5. security Management
   - Keeps all information secure
   - Allow access to authorized user

Date :
Page :

Operating system as Extended Machine

- Hides all unnecessary concerns, create a clean space to work for user.
- In this view, the function of OS is to present the user with the equivalent of an extended machine or virtual machine.

Evolution of OS

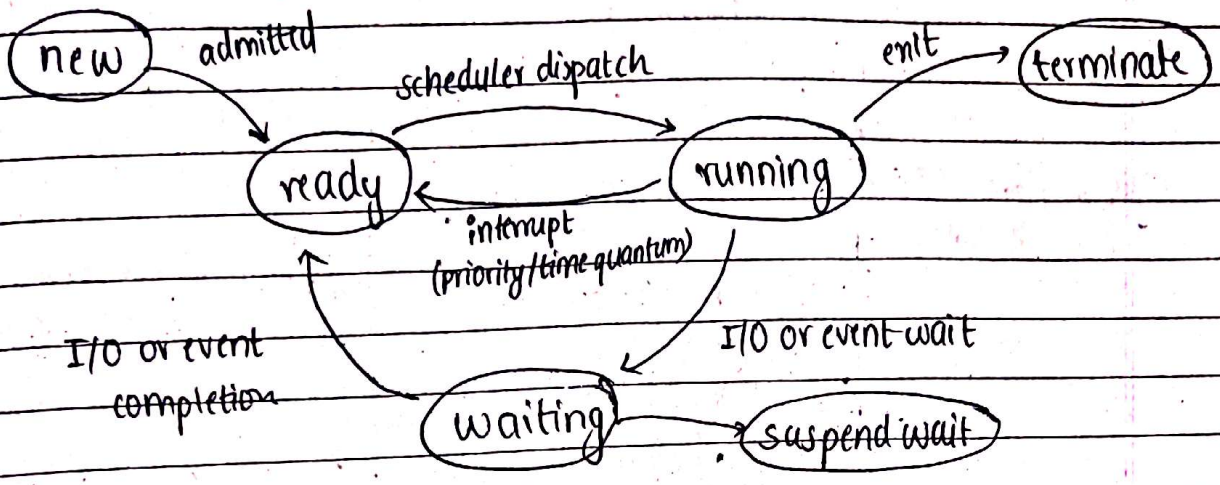
1. Serial Processing system (Before 1940s)
   - Executing one job at a time, sequentially without overlapping or multitasking
2. Batch Processing system (1940 - 1950)
3. MultiProgramming system (1950 - 1960)
4. Time sharing system (1960 - 1970)
5. Real time Systems (1970s - 1980) : Immediate response : Hard real time, soft real time
6. Personal computer OS (1980 - 1990) : GUI,
7. Distributed System (1990 - 2000)
8. Cloud Based and AI integrated systems (2000 - Present)

Date:
Page:

Process

The current program in execution

Process state:



1. New: Process ~~have been~~ created in ~~main~~ secondary memory. waiting for allocation in main memory.

2. Ready: Process loaded to main memory. waiting ~~to~~ for processor allocation. Ready state process are dispatched to running by scheduler.

3. Running: currently being executed. Possess all resources, including processor

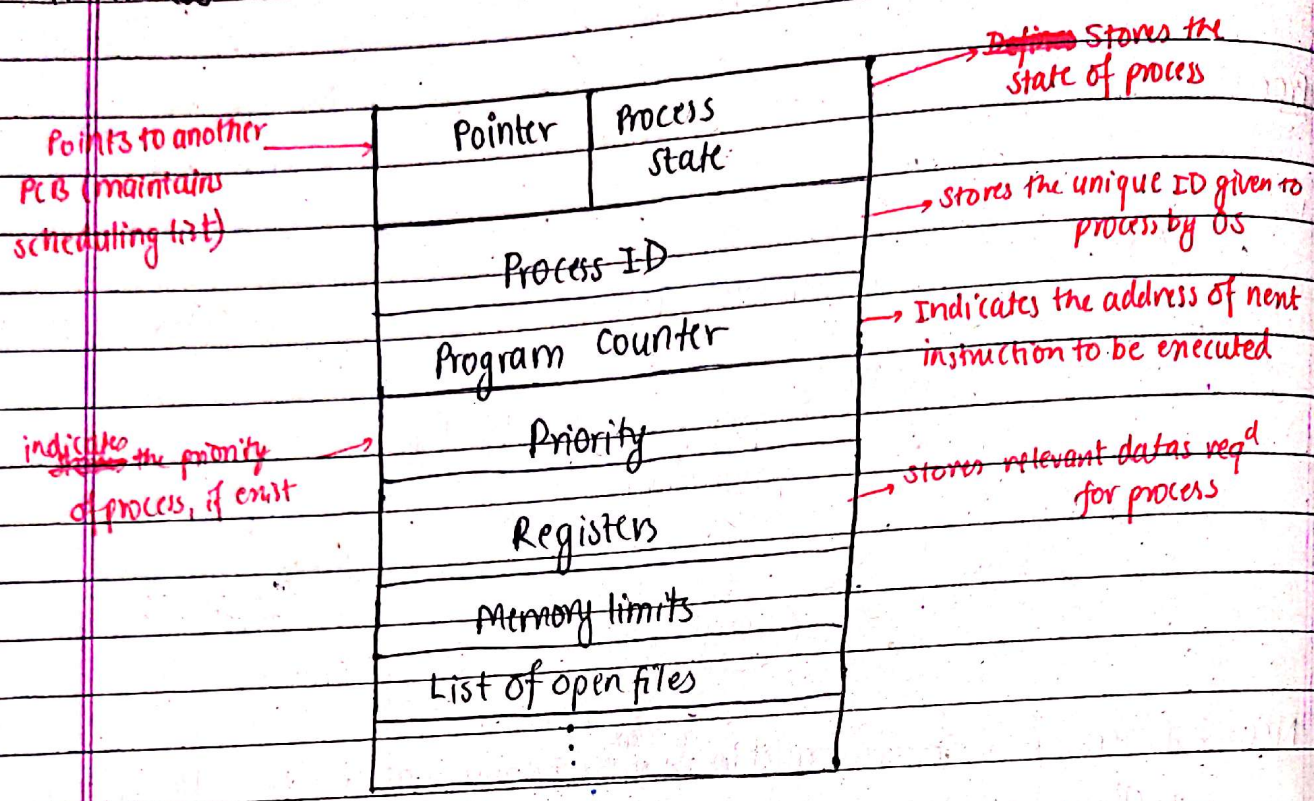
4. Terminate: Process ended. Deallocation of memory.

5. Waiting: Goes to this state if process cannot run until occurence of some event like I/O operations.

6. Suspend wait: If too many process in waiting, the process moves to suspend wait (wait in secondary memory)

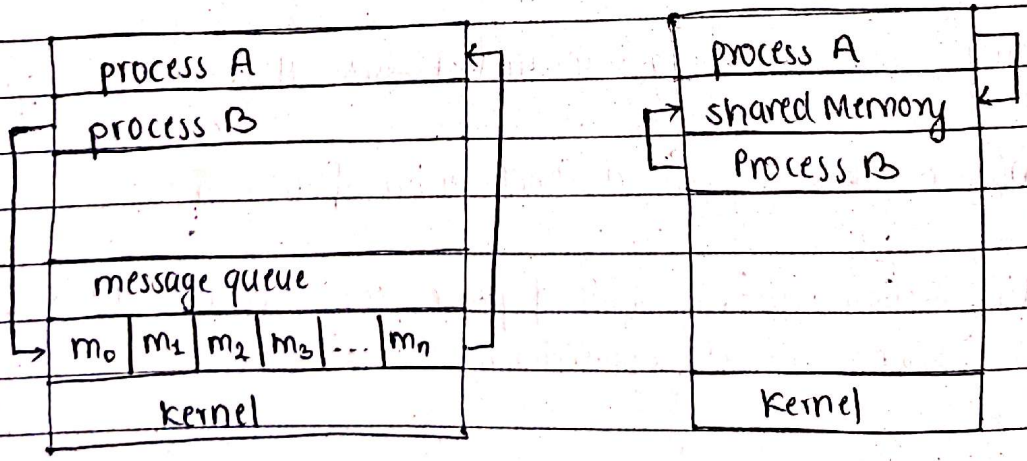
Date :
Page :

Process control Block (PCB)



Interprocess communication

occurs by two ways: (i) Message Passing
(ii) shared Memory



Date: .....
Page: .....

Race Condition (cooperative processes)

A race condition occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in multiple processes.

eg. e.g. - - - - (L3.1.G5).

Critical Section → It is a part of program where shared resources are accessed by the processes.

Four conditions should be met to have a good solution:

1. No two processes may be simultaneously inside critical region (mutual exclusion).
2. No process running outside critical region may block other processes. (Progress)
3. No assumptions should be made about hardware components (speeds/number of CPUs)
4. No process should have to wait forever to enter critical region. (Bounded waiting)

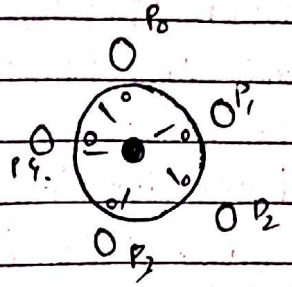
Producer and consumer Problem :

```
#define N 10        /* Initialize buffer size to 10 */
int count = 0;

void producer(void)
{
    int itemp;
    while(true)
    {
        if(count == N) sleep();
        produce_item(itemp);
        itemp = produce_item();
        if(count == N) sleep();
        insert_item(itemp);
        count = count + 1;
        if(count == 1) wakeup(consumer);
    }
}

void consumer(void)
{
    int itemc;
    while(true)
    {
        if(count == 0) sleep();
        itemc = remove_item();
        count = count - 1;
        if(count == N-1) wakeup(producer);
```

Date:
Page:

consume_item(item);
}
}

Dining Philosopher Problem:



Solution:
Represent chopstick with semaphore. Philosopher grabs chopstick by wait() & releases by signal() operation.

do{
wait (chopstick[i]);
wait (chopstick[(i+1)%5]);

//eat
---
signal (chopstick[i]);
signal (chopstick[(i+1)%5]);

//think

}while(true);

P_0	S_0	S_1
P_1	S_1	S_2
P_2	S_2	S_3
P_3	S_3	S_4
P_4	S_4	S_5

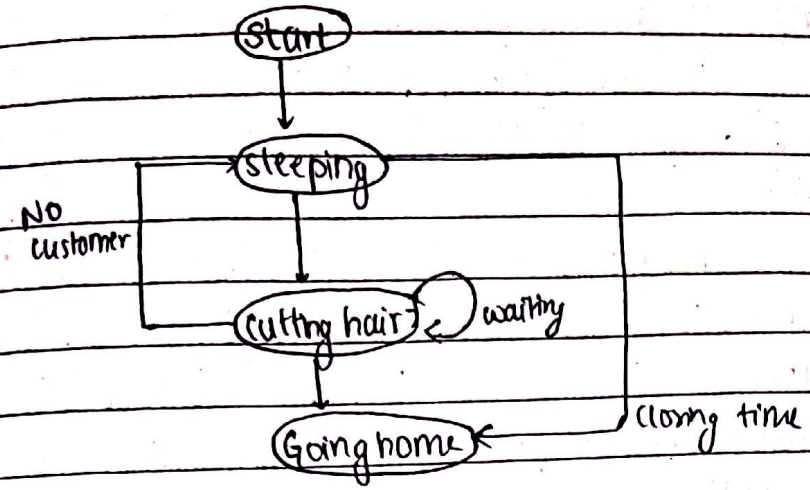
For last, do{
wait (chopstick[(i+1)%5]);
wait (chopstick[i]);

Reader writer Problem

```
int rc = 0;
Semaphore muten = 1;
semaphore db = 1;

void Reader (void)
{
  while (true)
  {
     down(&muten);
     rc = rc+1;
     if (rc == 1)  down(&db);
     up(&muten);
     read_data_base();
     down(&muten);
      rc = rc - 1;
      if (rc == 0)  up(&db);
      up(&muten);
      use_data_read();
  }
}

void writer (void)
{
   while (TRUE) {
     down(&db);
     write_data_base();
     up(&db);
} }
```

Date : .....
Page : .....

Sleeping Barber Problem




```
#define CHAIRS 5
typedef int semaphore;
semaphore customers = 0;
semaphore barbers = 0;
semaphore mutex = 1;
int waiting = 0;

void barber (void)
{
  while (TRUE){
    down (&customers);
    down (&mutex);
    waiting = waiting - 1;
    up(&barbers);
    up(&mutex);
    cut_hair();
  }
}
```

```
void customer (void)
{
    down(&mutex);
    if (waiting < CHAIRS) {
        waiting = waiting + 1;
        up(&customers);
        up(&mutex);
        down(&barber);
        get_haircut();
    }
    else {
        up(&mutex);
    }
}
```