1

Chapter-3: Real Time Operating System:-

All operating systems must provide three specific functions:-
- task scheduling
- task dispatching
- Intertask communication.

- A scheduler determines which task will run next in multitasking system
- A dispatcher determines the necessary bookkeeping to start that task.
- Intertask communication assures that the task cooperate.

Kernel (Executive / nucleus)

→ A kernel is the smallest portion of the operating system that provide the functions:- -task scheduling
-task dispatching
-intertask communication.

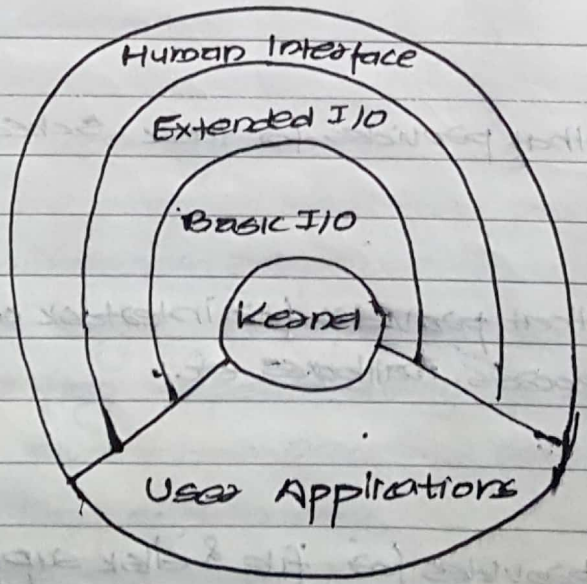


fig: The role of a kernel in operating system.

Users

User Interface Shell, Security	Operating system
File & disk support / I/O services	Executive
Interask communication & synchronization	Kernel
Task scheduling	Micro-kernel
Thread/Task Control Block Management	Nano-kernel

Hardware

fig: Kernel Hierarchy.

- Nano-kernel:-
  - A nano-kernel provides for simple thread-of-execution
  - It provides for task-dispatching

1. Nano-kernel
   - A simple thread-of-execution that provides for task dispatching.

Inter dependence

2. Micro-kernel:
   - A nano kernel that provides for task scheduling

3. Kernel:-
   - A micro-kernel that provides for interask communication via semaphores, mailboxes etc.

4. Executive:-
   - A kernel that provides for file & disk support, I/O services and other features

Operating System:- An executive that provides for general user, interface, security, file management etc.

Imp # Different types of Real Time Kernels:-
- Pseudo-kernels
- Interrupt-Driven System
- Preemptive Priority System
- Hybrid Systems.

Imp # Pseudo-kernels:-
1 Polled loop system
2 Polled loop with delay / Synchronized Pool Loop.
~~Interrupt Driven System / Interrupt-Only System~~
3 Cyclic Executives
4 Phase/State driven code
5 Coroutines.

1. Polled Loop System:-
- It is the simplest real time pseudo kernal.
- It provides fast response to single device
- Here, a single test instructions is used to test a flag repeatedly. The flag indicates whether the same event has occured or not. If the event has not occured, the polling continuous
- Only one task exists, so no intertask communication and scheduling is required.

```
while (true){
    if (event_occured)
    {
        process_event()
        event_occured = false;
    }
}.
```

- It wastes CPU time.

2. Synchronized Polled Loop or Polled Loop with Delay:-

- It is the extension of polled loop where a fixed clock interrupt is used to pause between when the signalling event is triggered and then reset.
- It is used for overcoming the false alarms generated due to switch bounce phenomenon.

```
while (true){
    if (flag)
    {
        pause (20); // wait 20 ms.
        process_event();
        flag = false;
    }
}.
```

3) Cyclic Executives:-

- It is the noninterrupt driven systems that can provide the illusion of simultaneity by taking advantage of sequentially short processes on a fast processor in a continuous loop.
- A de-facto cycle time is established which is same for each task as they execute on "round-robin fashion"
- Interlock communication could be achieved through global variables.
- ~~Different data structure~~

```
While (true) {
        Process_1();
        Process_2();
        ....
        task_n();
}
```

- Different data structure can be achieved by repeating a task in list.

4) Phase / State Driven Code:-

- It uses nested if-then statement, case-statement or a finite state automation to break up the processing of a function into discrete code segments.
- This allows temporary suspension of code segment without loss of critical data.

```
procedure task;
begin
  case flag
1: begin
     perform_part_1;
     flag := 2;
   end
2: begin
     perform_part_2;
     flag := 3
   end
end
end
```

At the end of each state, a flag is set and the task is terminated. Upon restarting, the process resumes from where it left off.

(5) Coroutines

→ Here, two or more tasks are coded in state driven fashion and after each phase is completed, a call is made to a central dispatcher.

→ The dispatcher holds the program counter for a list of tasks that are executed in round-robin fashion. i.e Dispatcher selects next task to execute. This task then executes until its next phase is completed and the central dispatcher is called again.

→ If there is only one task, then it becomes cyclic.

→ Communication between the tasks is achieved via global variable.

```
void task_a(){
  switch(state_a)
  {
    case1: phase_a1(); //to dispatcher
    break;
    case2: phase_a2();
    break;
    case3: phase_a3();
    break;
  }
}
```

```
void task_b(){
switch(state_b)
{
case 1: phase_b1();
break;
case2: phase_b2();
break;
case3: phase_b3();
break;
}
}
```

- Here, variables state_a & state_b are state counters that are global variables managed by dispatcher.
- The intertask communication & ~~dist~~ synchronization are maintained by global variables and coordinated by dispatcher.

ii) Interrupt-Driven System / Interrupt-Only System:

- In interrupt driven systems, the main program is a single "jump-to-self" instruction.
- Here, the various tasks in the system are scheduled via either hardware or software interrupts and dispatching is performed by the interrupt handling routines
- The interrupts in an interrupt driven system may occur at fixed rates (periodically), aperiodically or both
- There are two kinds of interrupts
  - Hardware interrupt
  - Software interrupt.
- The trigger of the hardware interrupt is an electrical signal from some external device, the trigger of the software interrupt is the execution of specific machine language instruction.
- When interrupted, the program is ~~stop~~ suspended & the CPU invokes the interrupt handler (IH)
- When pure hardware interrupt scheduling is used, a real-time clock or other external devices issues interrupt signals and direct it to an interrupt controller,
- The interrupt controller issues interrupt signals for the CPU, depending on the order of arrival and priorities of the interrupts
- If the computer architecture supports multiple ~~hardware~~ interrupts, then the hardware handles dispatching as well.
- But, if the computer architecture support only one

interrupt, then the interrupt-handling routine read the interrupt vector on the interrupt controller, determine which interrupt occured and dispatch the appropriate tasks

- The most important part of the interrupt handling system is the context switching
- Context switching is the process of saving and restoring sufficient information for a real-time task so that it can be resumed after being interrupted.
- The context is saved to a stack data structure.
- While context switching, the information typically includes
  - contents of work registers
  - content of program counted registers.
  - Content of memory page registers.
  - Special variable's values

- Interrupts are disabled during the critical context-switching period.

Disadvantages of Interrupt Driven system:-

- Time wasted in jump to self loop
- Difficulty in providing advanced services.
- Vulnerable to several types of malfunctions due to timing variations, race conditions etc.

```
void main()
{
    init();          //system initialization
    while(TRUE);     // jump to self.
}

void interrupt1()
{
    save(context);
    ~~task();~~
    execute_interrupt1();  //execute task1
    restore(context);
}

void interrupt2()
{
    save(context);
    execute_interrupt2();
    restore(context);
}
```

Preemptive Priority Systems:-

- Systems that use preemption schemes instead of round-robin or first-come-first-served scheduling are called preemptive priority systems.
- Here, A higher priority task is said to preempt a lower-priority one if it interrupts the executing lower-priority task.
- The priorities assigned to each interrupt are based on the importance and urgency of task associated with the interrupt.
- Prioritized interrupts can be either fixed priority or dynamic priority.
- In fixed priority systems, the task priorities can't be changed after initialization. Most embedded systems are implemented with fixed priorities, because there are limited situations in which the system need to adjust the priorities during runtime.
- In dynamic priority systems, the priorities of the tasks are to be adjusted at runtime to meet the changing real-time demands (priorities).
- Preemptive priority schemes can suffer from resource hogging (taking too much of sth.) by higher priority tasks. This can lead to a lack of available resources for lower priority tasks. In such case, lower priority tasks are suffering from resource starvation
- A rate monotonic system is the fixed priority interrupt driven system in which the priorities are assigned in such a way that, higher the execution frequency, higher the priority. (lower the period)

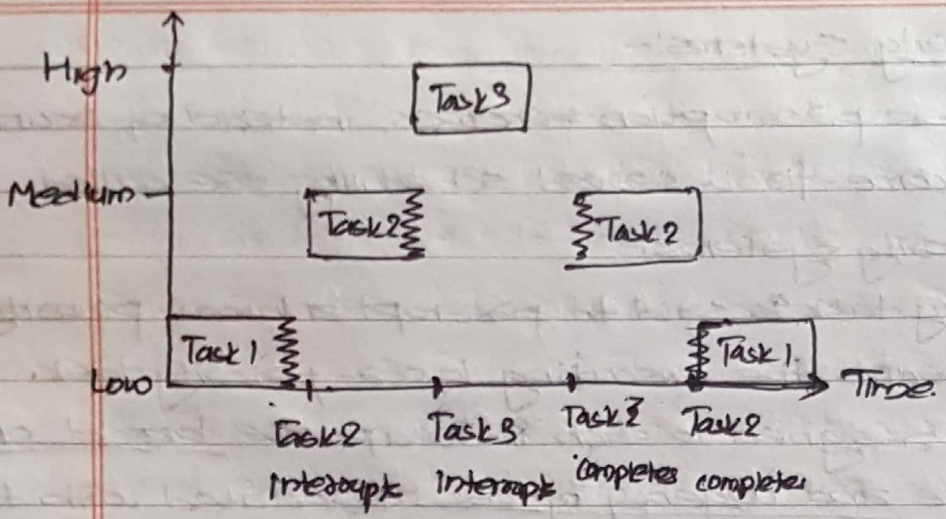


fig: Pacemptive scheduling of 3 tasks

5. Hybrid Systems:-

- Hybrid systems include interrupts that occur at both fixed rates and sporadically.
- The sporadic interrupts can be used to handle a critical event that requires immediate attention, and thus have the highest priority. This type of hybrid-interrupt system is common in embedded system.
- The another type of hybrid scheduling is the combination of round-robin and pacemptive priority systems. In these systems, tasks of higher priority can always preempt those of lower priority. But, if two or more tasks have same priority than, they run in round-robin fashion. This type of hybrid system is common in commercial operating systems.
- Another type of hybrid scheduling is foreground/background systems.

Foreground/background systems

- Foreground/background systems are the most common solution for embedded applications.
- They involve a set of interrupt driven processes called foreground and a set of non interrupt driven processes called background.
- The foreground tasks run in round-robin, preemptive, priority or combination fashion.
- The background task is fully preemptable by any foreground task and, has a lowest priority task in the system.

for eg; polled loop is a simply a foreground/background system with no foreground and polled loop as a background.

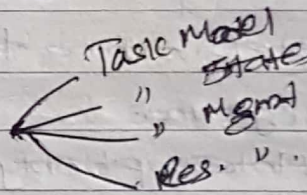
- and interrupt only systems are foreground/background systems without a background.

Imp

Full-featured Real-Time Operating system.

We can extend the foreground/background solution into an operating system by adding additional functions such as network interfaces, complicated device drivers, and complex debugging tools.

- Such systems rely on a complex operating system using round-robin, preemptive priority or combination of both schemes to provide scheduling
  - Commercial real-time operating systems are most often of this type.
- The TCB (Task Control Block) model is most often used in these types of system.



Task-Control Block Model

- The TCB model is the most popular method for implementing commercial, full-featured real time operating systems because number of real time tasks can be variable
- This can use round-robin, preemptive or combination of both to schedule the tasks.
- It facilitates dynamic task prioritization.

The Model:

(register contents, program counter)

- In model, we associate each task: a context, an identification string or number, status and a priority if applicable.
- These items are stored in a structure called a task control block (or TCB).
- The TCB are collected and stored in one or more data structures, such as a linked list.

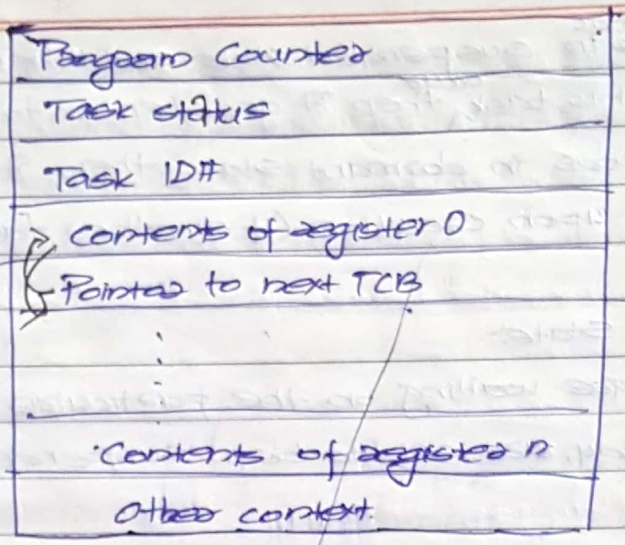


fig: A typical TCB.

Task States:-

- Each task can typically be in any one of the following states:-
  1. Executing
  2. Ready
  3. Suspended
  4. Dormant/Sleeping.
  5. Terminated

1. Executing:

- The executing task is one that is actually running
- A task can enter the executing state when it is created (if no longer tasks are ready) or from the ready state (if it is eligible to run based on its priority or its position in the round robin ready list).

2. Ready

- Ready state are those that are ready to run but are not running.
- The task enters the ready state if it was executing and its time slice runs out, or if it was preempted.

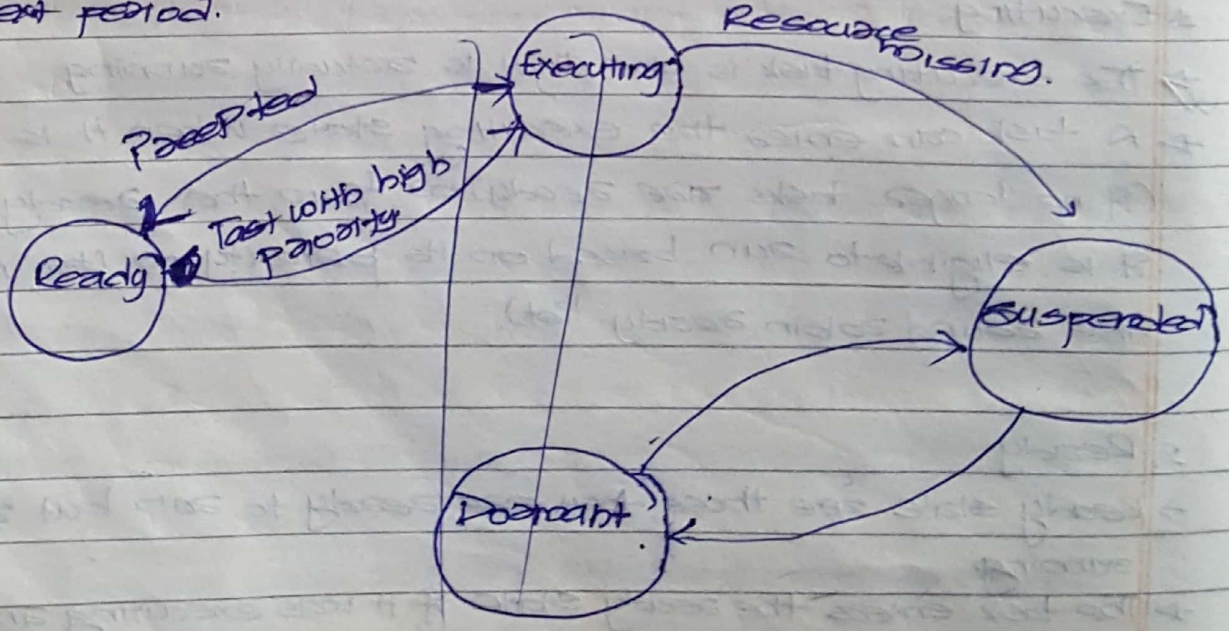
- If the task was in suspended state and when an event that tagged this task occurs then it goes/enters to ready state.
- If the task was in dormant state, then it enters to ready state upon creation (if another task is executing).

3) Suspended State:-

- Tasks that are waiting on the particular resource and are not ready, are said to be in suspended state.

4) Dormant State:-

- The dormant state is used only in the systems where the number of task-control blocks is fixed.
- Tasks that exists but not available (for scheduling) to operating systems are said to be dormant state. (task crea
- Once the task has been created, it can become dormant by deleting it.
- A periodic task enters this state when it completes its execution and has to wait for the beginning of the next period.



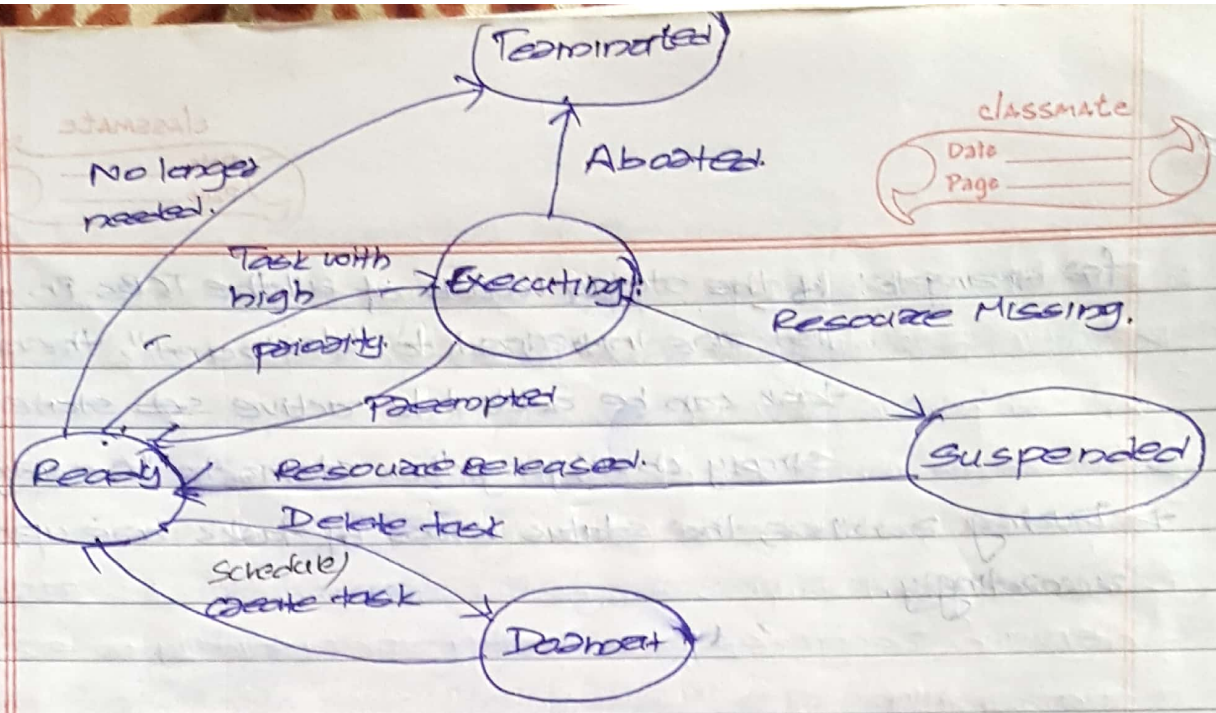


fig: state transitions in TCB model system

Task Management: (status word ⊂ TCB)

- Every hardware interrupt and every system call (such as request on a resource) invokes the real time operating system
- The operating systems maintains 2 linked lists
  - One linked list contain TCBs of all the ready tasks
  - Another linked list contain TCBs of all the suspended tasks.
- The operating system maintains 2 tables.
  - One table for resources
  - Another table for resource requests.

- When operating system is invoked, the operating system checks the ready list to see if any task is eligible for execution or not.
- If any task in ready list is eligible, then the TCB of the currently executing task is moved to the end of the ready list (round-robin approach) and the the eligible task is removed from the beginning of the ready list and its execution begins.
- Task state management can be achieved by manipulating the status word

Linked list: - Suspended list
- Ready list.

table - resource
- resource req.

for example:- If the status words of all the TCBs in the list are initialized to "dormant", then each task can be added to active sets. state by simply changing the status to "ready".

- During runtime, the status words of tasks are updated accordingly.

Resource Management:-

- The operating system checks the status of all resources in the suspended list.
- If the task is suspended due to a wait for a resource, then that task can enter the ready state only upon availability of the resource.
- When two tasks are suspended on the same resource, ~~table of resource~~ message request list structure is used to decide between them.
- If a resource becomes available to a suspended task, then the resource table is updated and the ~~super~~ eligible task is moved from the suspended list to the ready list

Theoritical Foundation of Scheduling.

- Many real time systems are inherently concurrent
- The interaction of the real time system with the external events requires multiple simultaneous tasks.
- A task is the active object of real time system and is the basic unit of processing managed by scheduler.
- As a task is running, it dynamically changes its state, and at any time, it may be in one, ~~but only one~~, out of 4 states.
- But sometimes, a fifth state "terminated" is included, if the task has finished its running, has aborted or self-terminated or no longer needed.

8.2.1. Process Scheduling:-

- Scheduling is the primary function of operating system.
- In real-time system, in order to meet the requirements in some real-time, a solid strategy is required for ~~the~~ scheduling the use of system resources and for a particular scheduling policy, its response time should be determined.
- The goal of scheduling is to strictly satisfy response time specifications.
- There are two types of scheduling.

1. Pre-run-time scheduling:- (Static)
- Here scheduling is done offline. so, the execution order of processes ~~and~~ prevents simultaneous access to shared resources
- It reduces the cost of context switching overhead.

preemption - possibly (frequent)
context switching overhead

2. Run-time scheduling:-
- Static priorities are assigned & resources are allocated on a priority basis. (Scheduling)
- It relies on a complex run-time mechanism for process synchronization & communication.
- This approach allows events to interrupt tasks and demand resources periodically, aperiodically or even sporadically.

Task Characteristics of a Real Workload:

- The workload on processors consists of individual tasks.
- Each individual task is a unit of processing to be allocated CPU time and other resources when needed.
- Every single CPU is assigned to at most one task at any time.
- No task is scheduled before its release time.
- Each task (T_i) is characterised with following parameters

1. Precedence Constraints:-
   - Specify if any task needs to precede other tasks.
2. Release time ~~(rel)~~ $r_{i,j}$:
   - The release time of the j^{th} instance of task T_i
3. Phase Φ_i
   - The release time of the first instance of T_i.

2)

4. Response time:-
- The time span between task activation and its completion.

5. Absolute deadline d_i:
- The instant by which the task τ_i must complete.

6. Relative deadline D_i:
- The maximum allowable response time of task τ_i.

7. Period P_i:
- The minimum length of interval between two consequtive release times of task τ_i.

8. Execution time e_i:
- The maximum amount of time required to complete the execution of task τ_i.

Typical Task Model:-
- A task model is presented in order to describe some standard scheduling policies used in real time systems.
- A task model has following assumptions:-

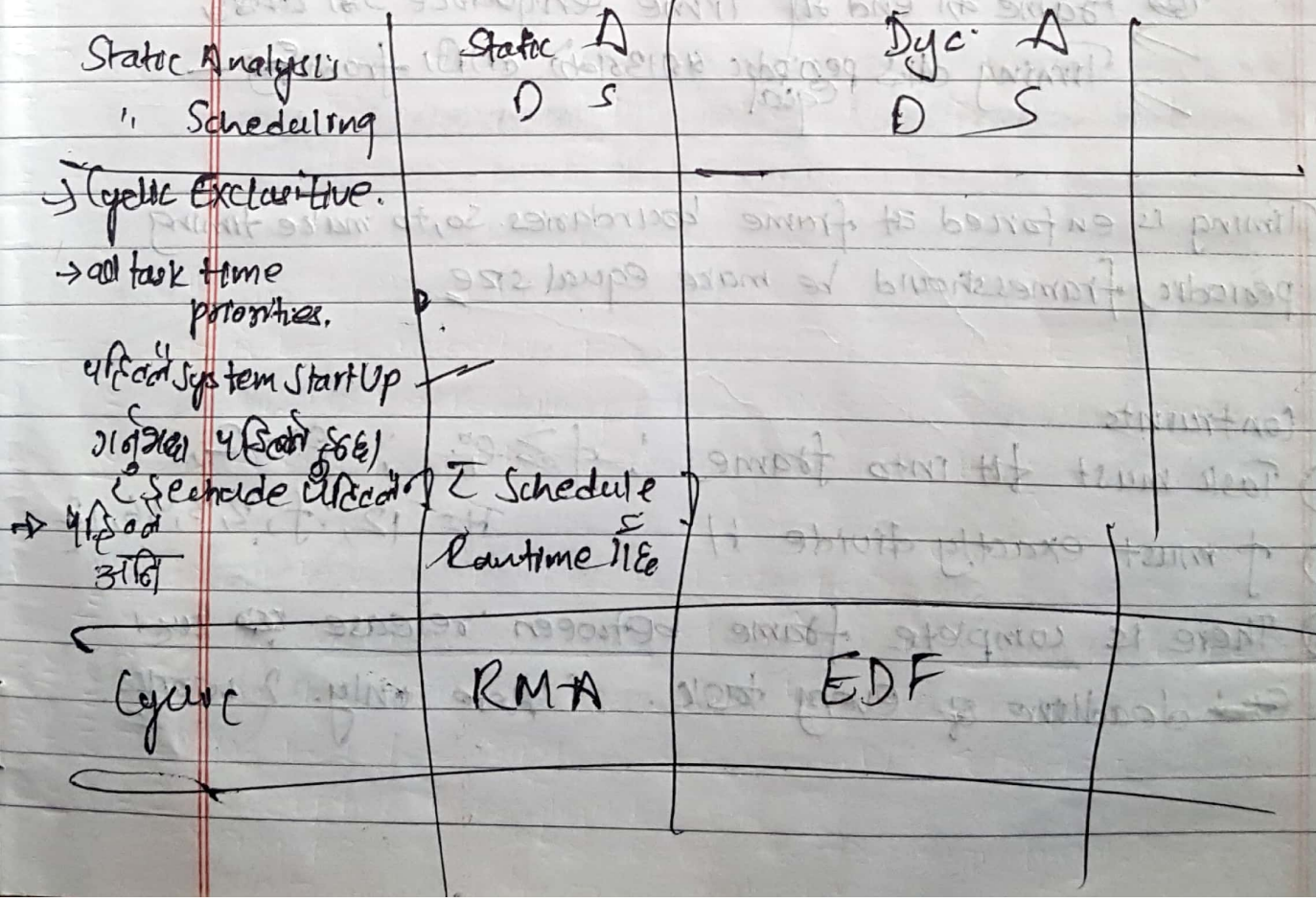
1) All tasks in the task set are periodic.
2) The relative deadline of a task is equal to its period.
3) All tasks are independent, there are no precedence constraints.
4) No task has any nonpreemptible section, and the cost of preemption is negligible.
5) Only processing requirements are significant; memory & I/O requirements are negligible.

- predictable schedule, - round robin scheduling use गर्द
periodic
Relative time = Period.
independent
preemptible
- Processing deep मात्र significant हुन्छ, (context switching आदि negligible हुन्छ)

- For real time system, it is important to use scheduling algorithm to produce a predictable schedule. i.e to know which task is going to execute next
- In many real time operating system, round-robin scheduling policy is used because it is simple and predictable.

3.2.2. Round Robin Scheduling:

- One of the oldest & mostly used algorithm for process scheduling
- Each process is assigned a time interval-quantum
- Here several tasks are executed sequentially to completion, often in conjunction with a cyclic code structure.
- If the process is still running at the end of the quantum, the CPU is preempted & given to another process.



Cyclic Executive:-

* आगाडि को टी,
Schedule गरि Table मा राखिन्छ

* non-preemptive Scheduling

* non periodic task आउँदा CPU Idel

Hyperperiod:- सबै Task schedule गरि Table मा राख्ने र
त्यो table execute गर्ने Time लाई भनिन्छ
- LCM of all ~~task~~ periods

किन्छ eqn size को frame size चाहिन्छ ?
~~का~~ frame को end मा Time enforce गर्ने भएकाले
→ Timing लाई periodic बनाउनको लागि framing equal

Timing is enforced at frame boundaries. So, to make timing periodic, frames should be make equal size.

Contraints
(1) Task must fit into frame; $f \geq e_i$; $f \geq 2$,
(2) f must exactly divide H $H = 12$, f; 2, 3, 4, 6, 12
(3) There is complete frame between release ~~and~~ and ~~end~~ deadline of every task. Here only 2 works

	e_i	P_i
1	1	3
2	2	4
3	1	6

(2) R.M.A

Utilization factor $n(2^{1/n}-1)$

Total Utilization $\sum\left(\frac{e_i}{P_i}\right)$ < Utilization factor

L.C.M

Book

→ EDA Utilization

$\sum\left(\frac{e_i}{P_i}\right) \leq 1$