OS Fundamentals

Task
Executed by OS under certain conditions → Routine of code

main
{
Init()
create task()
os_start()
}

Types of Task
- → Cyclic Task.
- → Initialization Task
  Periodicity = 0
- → Event based Task

ISR
Create task
Task
delete task
}

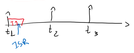
Task Event
Task 1
{
if(~)
(create task)
}

Scheduler

Types
- → Non-Preemptive
- → Preemptive.

Ready Queue
T_1
T_2
T_3
T_4
T_5

- Non-Preemptive
  T_1 will finish execution then T_2 will execute



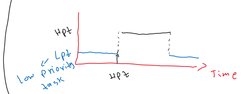
- Preemptive
  T_2 will execute then system resume to complete task 1

ISR
{
create_Task (T_2)
}

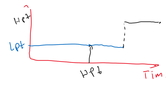
T_2 Priority > T_1 Priority

what does it mean by task fired:-
- option 1
  ISR fired
  ↳ create task
- option 2
  current executing task created another task

Preemptive systym



Non Preemptive System

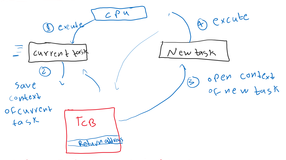


How preemptive system work?

Non preemptive → scheduling Point → every tick

Preemptive system → scheduling Point → every tick / → create task / → Resume task

Dispatcher



TCB → Task control Block.

→ Kernel
Scheduler + Dispatcher + timer

Task Implementation techniques

→ Ending task

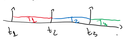
```
task 1
{
}
```

→ End less task 1

```
Task 1
{
while (1)
{
}
}
```

Scheduling Techniques

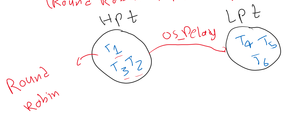
↳ 1- Round Robin
TickTime = 1m



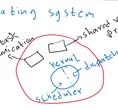
2- priority based
↳ Delay APS

```
HPt
{
while (1)
{
}
Zzzz ← OS_Delay (~~)
}
}
```

3- Hybrid scheduler
(Round Robin + Priority based)



Operating system



Real Time operating systems

↳ Action → correct time

- Hard Real time
  Tough timing constrains
- Soft Real time
  Higher tolerance than hard

Semaphore

Software key used to
→ 1) Shared resource problem
→ 2) Inter task synchronization

usb $S_1 = S_2 = 0 = 1$
Printer → Shared resource
Hello which Hello world

Priority

```
Task 1
{
Take Semaphore(S1);
Release semaphore(S1);
}
```

```
ISR
{
create task(2) → T1
}
```

```
Task - 2
take semaphore(S1)
Release semaphore(S1)
}
```

Example-2

waiting step

```
Task 1
{
}
Take-semaphore(S1)
}
```

semaphore S1=0; = 1;

```
Task - 2
{
}
Release- semaphore(S1);
}
```

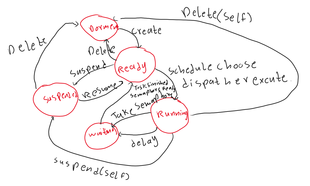
excute before Task2

excute after task2

Types of semaphore

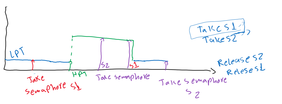
1) Binary semaphore
   Semaphore value = 1.
2) Counting semaphore
   Semaphore value ≥ 1

Task states



Semaphore problems

→ Dead Lock

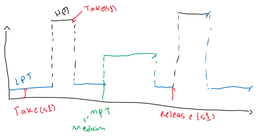


Dead Lock Solutions:
1) No waiting with semaphore Acquisition
2) Semaphore Acquisition timeout
3) Verify resources Availability before excution.

Semaphore problems:-

Finishing order
M → H → L

Priority Inversion

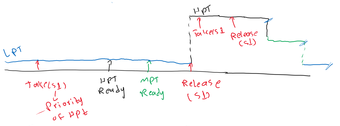


Priority inversion Solution

↳ Priority Inheritance

① every semaphore has priority
= the highest priority of the tasks that acquire this semaphore.

② any task acquire this semaphore. would inherit its priority.



LPT would upgrade its piriority to HPT

Finishing sequence
H → M → L

Mutex

- protection mechanism for shared Resources
- Binary only.

Take_ mutex(my)
Release_ mutex(my)

- OS can't force Release the Mutex from any task.
- If Dead lock happend OS can't Release Mutex from itself.
  (Task must Release mutex for itself)