× brk : lock data region to prevent changes. Next if increase operation, if new region is illegal (overlap, unavail) unlock & fail. Else, change size (growreg), zero out data spc addrs. & unlock region.

× Process Scheduling, Time & Clock —:

- × CPU scheduler (part of OS) selects processes from ready queue for execution.
- × Attempts : Maximize : Throughput, CPU utilization
  Minimize : Waiting, Response, Turn-around Time.

Throughput : # procs / unit time (completed)
CPU Util : Time in which CPU is busy.
Wait Time : Time spent in ready queue
Response Time : Time spent in waiting until first schedule. (first CPU allot)
Turn-Around Time : Total difference b/w start & end time stamps.

× Dispatcher : Allots CPU to process in ready state & move to running state after process is scheduled by scheduler. (Performs context switch, switch to usrmode, jump to proper instruction to resume exec)
Dispatch Latency : Time taken for switch.

× Non-preemptive Scheduling : Process always completes execution prior to switch. (No preempt based on priority)

× Preemptive Scheduling : Process may be preempted to allot CPU to another higher priority process.

Multilevel Queue Scheduling —:

- Ready queue partitioned into queues (eg. foreground & background), & each queue is scheduled via a separate scheduling algorithm.
- Process do not change queues.
- Scheduling also done b/w queues. (Using a sched. algo.)

Multilevel Feedback Queue Scheduling —:

- MQS, w/ process allowed to switch queues (based on task size). (Process moved automatically b/w queues (prioritizing))
- Eg : Q_1 : RR (Quantum = 8ms)
  Q_2 : RR (Quantum = 16ms)
  Q_3 : FCFS

  P_1 w/ burst time 16+ ms first enters Q_0, if it does not complete moved to Q_2 (lower priority) & if it does not complete moved to Q_3 (FCFS, lowest priority).

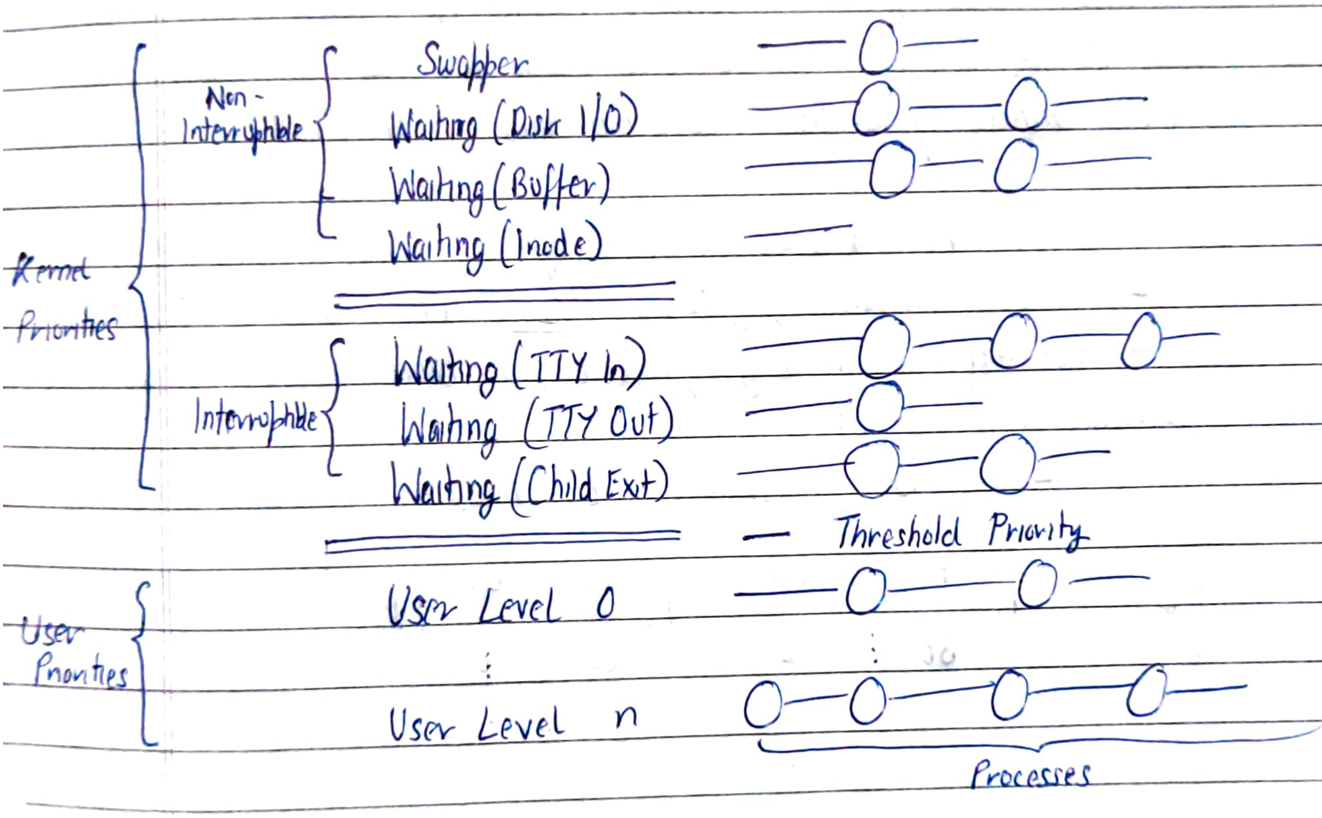
× Unix-Scheduler : Fair-Share Scheduler (Incorporates group info in Round Robin), w/ MFQS.

- Uses a time-slice / time-quantum.
- Process preempted post expiration of time slice.
- Processes also have priority (influences decision of scheduling)

- tie breaking rule : kernel picks process w/ longest ready to run ti
- Process table maintains priority field.
  - Low Priority ⇒ Recently alloted CPU.

schedule_process : While a process has not been picked for execution,

× Priority Levels :
User Priorities (Lower than Kernel)
Kernel Priorities (priority threshold above a threshold).



- Kernel mode priority levels in order of resource utilization levels.
  (Higher resource usage => higher priority level (to free resources faster))
- Priorities recomputed after every time slice (user & kernel levels).

Priority Adjustment — :

× Assigned to process about to sleep
× Reassigned to process moving from kernel to user mode.

- × Clock handler readjusts priorities of all processes in user mode at 1s intervals (causes kernel to go through the scheduler to prevent CPU use monopoly).
- × Clock may interrupt process several times during its time quantum.
- × decay (CPU_Usage_Time) = CPU_usage_time/2.
- × Priority readjustment :
  - Priority = (base level user priority) + $\frac{1}{2}$decay(recent_cpu_usage)

    Lower priority => Higher chance of scheduling.
- × Priorities of processes in kernel mode are not readjusted.
- × User mode priorities restricted with a threshold.

Eg:- P_1, P_2, P_3, created simultaneously w/ initial priority 60.
- Time Quantum : ~~(1/60)s~~ 1s, Clock tick : (1/60)s
- No syscall in any processes, no other ready processes.

Scheduling flow :
- Initial priorities : 60 (Base) + CPU/2 = 60

T↓	P_1	Prio.	CPU	P_2	Prio.	CPU	P_3	Prio.	CPU
0		60	0		60	0		60	0
			⋮			⋮			⋮
			60						
1	P_1	75	30	P_2	60	0		60	0
						⋮			⋮
			30			60			
2	P_2	67	15	P_2	75	30	P_3	60	0
			⋮			⋮			⋮
			15			30			60
3	P_1	63	7	P_2	67	15	P_3	75	30

3	P_1	63	7	P_2	67	15	P_3	75	30
			⋮			⋮			⋮
			67			15			30
4	P_1	76	33	P_2	63	7	P_3	67	15
			⋮			⋮			⋮
			33			67			15
5	P_1	68	16	P_2	76	33	P_3	63	7
			⋮			⋮			⋮
			16			33			67
6	P_1	64	8	P_2	68	16	P_3	76	33

Procedure : Allot CPU to lowest priority value (highest priority) process.
Add (seconds / time quantum) * (ticks / seconds) to CPU usage.
For next step : decay CPU usage.
Calculate new priority via 'decayed' CPU usage.

× Controlling Process Priorities —:

- External modification via nice (syscall).
- Child process inherits nice value of parent.
- nice(int nice_value) : increments / decrements nice field / value of process, where nice value is added to process priority.

$$\text{New Priority} = (\text{base priority}) + \frac{1}{2}\,\text{decay}(\text{recent\_cpu\_usage}) + (\text{nice\_value}).$$

- Only change of self's nice value is allowed.

× Fair - Share Scheduler —:

× Divide processes into set of fair-share groups.
× CPU allocated proportionally to each group, regardless of processes in group.

× Handled by adding fair-share field to priority.
- Priority = Old Priority + (Group CPU Usage)/2.

Eg. P_1 in G_1, P_2 & P_3 in G_2.
Fair-Share of group added to priority value (decayed as well)
60 ticks/s, 1s time quantum.

	T ↓	P_1			P_2			P_3		
	0	Prio.	CPU	Grp.	Prio.	CPU	Grp.	Prio.	CPU	Grp.
Schedule P_1	0	60	0	0	60	0	0	60	0	0
↓			⋮	⋮		⋮	⋮		⋮	⋮
			60	60		0	0		0	0
Schedule P_2	1	90	30	30	60	0	0	60	0	0
↓			⋮	⋮		⋮	⋮		⋮	⋮
			30	30		60	60		0	60
Schedule P_1	2	74	15	15	90	30	30	75	0	30
↓			⋮	⋮		⋮	⋮		⋮	⋮
			75	75		30	30		0	30
Schedule P_3	3	96	37	37	74	15	15	67	0	15
↓			⋮	⋮		⋮	⋮		⋮	⋮
			37	37		15	75		60	75
Schedule P_1	4	78	18	18	80	7	37	93	30	37
↓			⋮	⋮		⋮	⋮		⋮	⋮
			78	78		7	37		30	37
Schedule P_2	5	98	39	39	70	3	18	76	15	18
↓										

× Unix Scheduler not suitable for real-time OS, as the scheduler is time-sharing.

Time Utility System Calls — : <time.h>, <sys/times.h>, <unistd.h>

× stime(int timestamp) : Sets timestamp to given value
× time (time_t* ref) : Gets the timestamp of the system / given to the process. in the variable 'ref'.
× times (tms* tbuffer) : Gets info. on time spent in user & kernel mode for both parent & child in tbuffer.

```
typedef struct_tms {
        time_t  tms_utime ;   // User mode time.( relative to sys boot)
        time_t  tms_stime ;   // System/ Kernel mode time. (relative to sys boot
        time_t  tms_cutime ;  // User mode time (child/all children).
        time_t  tms_cstime ;  // System mode time (all children).
    } tms ;   // Use struct_tms as tms .
```

× alarm (int seconds) : Sends SIGALRM to current process after specified time in seconds.

Clock Interrupt Handler — :

× Functions : × Restart clk.
× Schedule invoking of internal kernel fx. based on timers.
× Provide execution profiling data & capability for kernel & user processes.
× Gather sys & process accounting stats.
× Keep track of time.
× Send alarm signals to processes on request.
× (Some operations are performed every clock intr., others after clock tick)

WRITEWELL
Date
Page

× Internal System Timeouts —:

- Kernel ops. require invocation of kernel fx. of real-time basis.
- Utilizes a callout table —:
  - Stores time related necessary info.
  - Functions to invoke on timeout.
  - Params for fx.
  - Time (clock ticks) until function should be called.
- Callout table entries sorted in ascending order of time to fire.
- Time to fire relative to current time & time of previous entry.
- Eg

Function	Time to Fire
a()	-2
b()	3
c()	10

- Absolute time to fire for a function : Cumulative time to fire of previous entries. (accumulated value) (excl. negatives).

× Insertion to Callout Table :

- Kernel finds correct timed position for new entry.
- Time field of successive entry adjusted to reflect correction.

Eg : add f() [Abs. Time to Fire : 5]

=>

Function	Time to Fire
a()	-2
b()	3
f()	2
c()	8

add d() [Abs. Time to Fire : 12]

=>

a()	-2
b()	3
f()	2
d()	7
c()	1

× During Execution :
- Decrease tick / time to fire value for first entry by 1 (from 1st +ve t.t.f. entry).
- Changes in successive entries are not required.
- On 0, fire the function associated on timeout.

× Negative Time To Fire entries :
- Signify functions NOT triggered upon their timeouts, due to kernel being busy or resource unavailability or interrupt.
- TTF value decremented below 0 on successive ticks.
- On interrupt for timeout, kernel execs. fc. whose TTF are 0 or -ve.
- Timeout signals implemented as software signals & interrupts (∴ Low priority compared to other interrupts).

Clock Algorithm : Maintains program counter info for profiling, if requested & repeatedly cycles to detect timeouts.
: First restart clock, on non empty callout tble.
adjust callout times & schedule timeout fc if timeout.
On kernel profiling, note PC value on interrupt.

× Profiling -:

× Measure of execution of system in system (Kernel) & user mode.
× Measure of time spent executing individual kernel routines.
× Kernel Profile Driver :
- Monitors relative perf. of kernel modules by sampling activity.

× Profile execution of processes at usr-lvl w/ profil
profil (buff, bufsize, offset, scale) ;

buff : Array addr. in usr spc to store info.
bufsize : Size of buffer array, in bytes.
offset : Start addr of process/fx to profile (user virt. addr)
scale : factor to map user virt. addr. into arr.

× Accounting & Statistics —:

- Processes have two fields in u-area : elapsed kernel & usr time.
- Clock interrupts : record usr & kernel time, as well as memory usage (function of priv regions & prop shared region usage) ~~for~~ processes (done by kernel).
- Shared memory usage recorded as proportion of processes sharing the memory.

Eg : Shared Text = 50K (w 4 oth proc.)
Stack = 40K
Data = 25K
Proc. Mem. Usage : 40 + 25 + (50/5) = 75K