1/10/24

Traditional UNIX Scheduling (SVR 3, BSD UNIX 4.3)

→ The scheduling algorithm is designed to provide good response time for interactive users while ensuring that low-priority background jobs do not starve.

→ SVR in chapter 10. Not in syllabus.

→ The traditional UNIX scheduler employs multilevel feedback using round robin within each of the priority queues.

→ The system makes use of one-second preemption: if a running process does not block or exit within one second, it is preempted.

→ Priority is based on process type and execution history. It is recomputed once per second, at which time a new scheduling decision is made.

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

where $CPU_j(i)$ = measure of processor utilization by process j through interval i;

$P_j(i)$ = priority of ~~proc~~ process j at the beginning of interval i; lower values → higher priorities,

$Base_j$ = base priority of process j,

$nice_j$ = adjustment factor.

→ The purpose of base priority is to divide all processes into fixed bands of priority levels.

→ The CPU and nice components are restricted to prevent a process from migrating out of its assigned band.

→ The bands are used to optimize access to block devices (disk) and allow OS to respond quickly to system calls.

→ In decreasing order of priority:

a) Swapper
b) Block I/O device control
c) File Manipulation
d) Character I/O device control
e) User processes.

→ Prefers CPU-bound over I/O-bound processes.

8/11/24

→ On a timesharing system, the kernel allocates the CPU to a process for a period of time called a time slice / time quantum.

→ Scheduler in Unix uses relative time of execution.

→ Each active process has a scheduling priority. ~~Every~~ The kernel switches context to that of the process with the highest priority.

→ The kernel recalculates the priority of the running process when it returns from kernel to user mode.

→ The system keeps time with a hardware clock that interrupts the CPU at a fixed rate – each occurrence of a clock interrupt is called a clock tick.

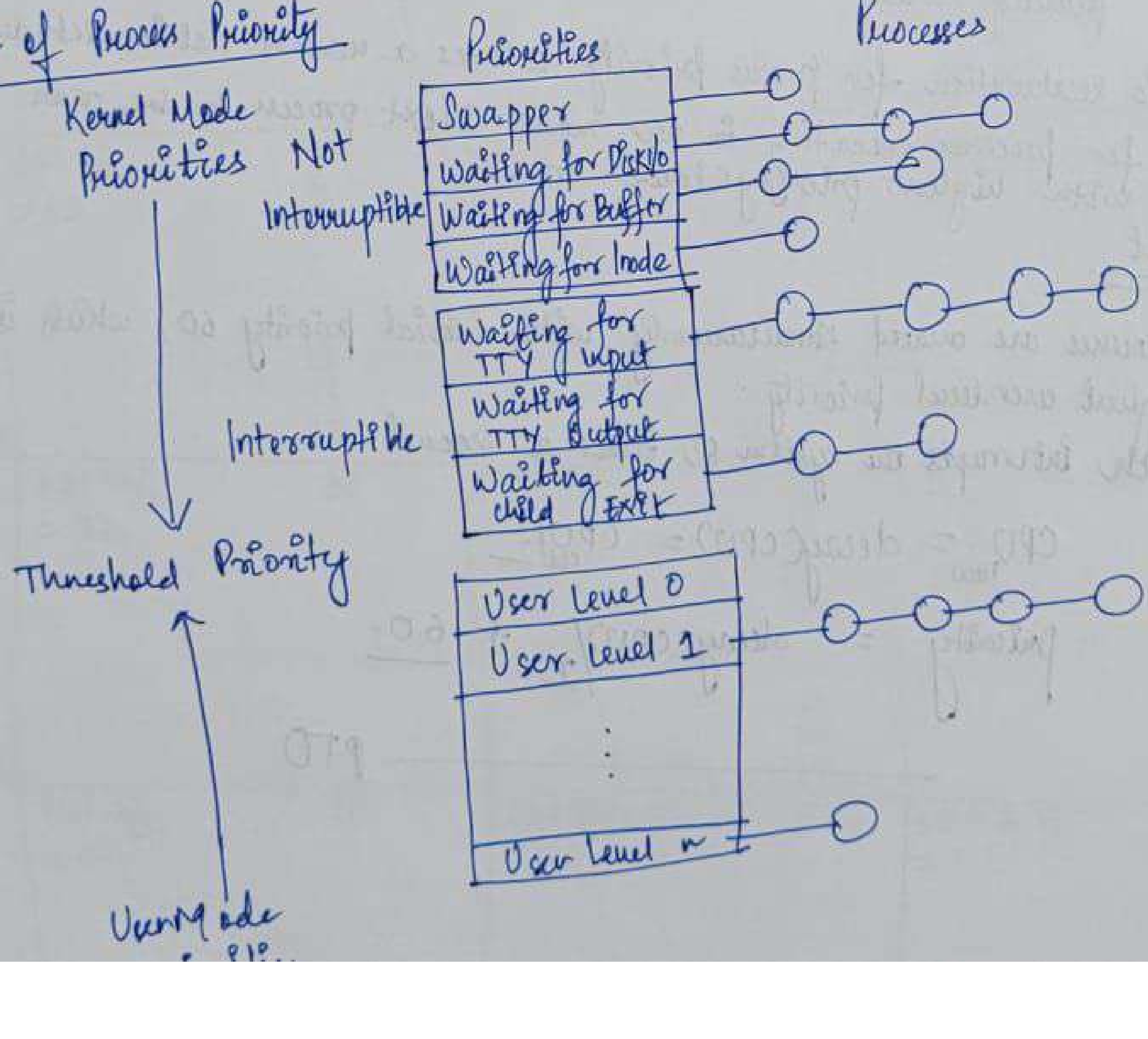
Scheduling Parameters

→ Each process table entry contains a priority field for process scheduling.

→ The range of process priority

- User priority: Is below a threshold value, Is preempted on their return from the kernel to user mode
- Kernel priority: Is above a threshold value, Achieves them in the sleep algorithm.

Range of Process Priority



Calculation of the priority of a process

- Scheduler assigns priority to a process about to go to sleep, correlating a fixed priority value with the reason for sleeping.
- The Kernel adjusts the priority of a process that returns from Kernel mode to user mode.
- Clock handler runs scheduling algo at 1 second intervals.
  - Increments a field in the process table that records the recent CPU usage of the process.
  - Adjusts the recent CPU usage of each process acc. to a decay func^n

$$\boxed{\text{decay(CPU)} = \text{CPU}/2}$$

  - Recalculates the priority of every process

$$\text{Priority} = \left(\cancel{\text{recent}}\ \text{CPU}\ \cancel{\text{usage}}\,\text{decay}/2 \right) + \text{base level user priority}$$

  - Processes in the ready-to-run state will tend to occupy more priority level.
- Periodic recalculation for process priority assures a round robin scheduling policy for processes executing in user mode. Next process to be run is the one with highest priority (lower no.s).

EXAMPLE

- The processes are created simultaneously with initial priority 60, which is also the highest user level priority.
- The clock interrupts the system 60 times a second.

$$\text{CPU}_{new} = \text{decay(CPU)} = \text{CPU}_{old}/2$$

$$\text{priority} = \text{decay(CPU)}/2 + \underline{60}$$

PTO

	Process A		Process B		Process C	
	Priority	CPU count	Priority	CPU count	Priority	CPU count
0	60	0 1 2 3 . . . 59 60 (to ticks)	60	0 . . . 0	60	0 . . . 0
1	$60+\frac{30}{2} = 75$	30	60	0 1 2 3 . . . 59 60	60	
2	$60+\frac{15}{2} = 67$ (only Entegers)	15	75	30	60	0 1 2 . . . 58 59 60
3	$60+7/2 = 63$	7 8 9 . . . 65 66 67	$60+15/2 = 67$	15	$60+\frac{30}{2} = 75$	30
4	$60+33/2 = 76$	33	$60+7/2 = 63$	7 8 9 . . . 65 66 67	$60+15/2 = 67$	15
5	$60+16/2 = 68$	16	$60+33/2 = 76$	33	$60+7/2 = 63$	7 8 9 . . .

9/11/24

FAIR SHARE SCHEDULING

→ The principle:- The user community is divided into a set of groups. The system allocates its CPU time proportionally to each group.

→ In a multiuser system, if individual user applications or jobs may be organized as multiple processes (threads), then there is a structure to the collection of processes that is not recognized by a traditional scheduler.

↳ From the user's POV, the concern is not how a particular process performs but rather how his/her set of processes performs.

↳ It then becomes attractive to make scheduling decisions on the basis of these process sets.

↳ This concept of process sets can be extended to groups of users, even if each user is represented by a single process.

→ Scheduling decisions are made with an aim to give each group similar service - if a large # of people from one ~~department~~ group log on to the system, response time degradation should affect only members of that group, rather than users from other groups.

→ The term fairshare indicates that each user is assigned a weighing of some sort that defines that users' share of system resources as a fraction of the total usage of those resources.

→ Each user is assigned a share of the processor. If user A has twice the weighting of user B, then in the long run, user A should be able to do twice as much work as user B. $\boxed{\text{FAIR} \neq \text{EQUAL}}$

↳ Objective of FSS is to monitor usage to give fewer resources to users who have had more than their fair share, and more to those who have had less than their fair share.

→ FSS considers the execution history of a related group of processes, along with the individual execution history of each process in making scheduling decisions.

→ Scheduling is done on the basis of priority, which is calculated as follows:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_K(i) = \frac{GCPU_K(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_K(i)}{4 \times W_K}$$

where $CPU_j(i)$ = measure of processor utilization by ~~prog~~ process j through interval i,

$GCPU_K(i)$ = measure of processor utilization of group K through interval

$P_j(i)$ = priority of process j at beginning of interval i; lower values equal higher priorities,

$Base_j$ = base priority of process j, and

W_K = weighting assigned to group k, $0 \leq W_K \leq 1$ and $\sum_K W_K = 1$

→ Each process is assigned a base priority. The priority of a process drops as the process uses the processor and as the group to which the process belongs uses the processor.

→ In the case of group utilization, the avg. is normalized by dividing the weight of that group.

↳ the greater the weight assigned to the group, the less its utilization will affect its priority.

→ Example:- Four groups, each group has 1, 2, 3, and 4 processes respectively

a) Regular scheduling algorithm: 10% of CPU time for each process.

b) Fair share scheduler: 25% of CPU time for each group.

Fair share group priority = $CPU/_2$ + Group $CPU/_2$ + Basekuel priority

→ Let's say process A is in one group, processes B and C are in a second group with each group having a weighting of 0.5

→ All processes are CPU bound, ready to run. All have a base priority = 60.

→ The processor is ~~incremented~~ interrupted 60 times per second; during each interrupt, the processor usage field of the currently running process is incremented, as is the corresponding group processor field. → Processor Utilization Measurement.

→ Once per second, priorities are recalculated. Next process ~~to~~ chosen is the one with highest priority.

→ Bigger numbers for priority → lower priority.

→ The Kernel schedules A, B, A, C, A, B, ... — 50% to group 1 (A), 50% to group 2 (B & C).

PTO

	Group 1			Group 2					
	Process A			Process B			Process C		
	Priority	CPU count	Group count	Priority	CPU count	Group count	Priority	CPU count	Group count
0	60	0 1 2 ⋮ 60	0 1 2 ⋮ 60	60	0	0	60	0	0
1	$60+\frac{30}{2}+\frac{30}{2}$ $=90$	30	30	60	0 1 2 ⋮ 59 60	0 1 2 ⋮ 59 60	60	0	0 1 2 ⋮ 59 60
2	$60+\frac{15}{2}+\frac{15}{2}$ $=74$	15 16 17 ⋮ 74 75	15 16 17 ⋮ 74 75	$60+30/2+30/2$ $=90$	30	30	$60+0/2+30/2$ $=75$	0	30
3	$60+\frac{37}{2}+\frac{37}{2}$ $=96$	37	37	$60+\frac{15}{2}+\frac{15}{2}$ $=74$	15	15 16 17 ⋮ 74 75	$60+0/2+15/2$ $=67$	0 1 2 ⋮ 58 59 60	15 16 17 ⋮ 74 75
4	$60+\frac{18}{2}+\frac{18}{2}$ $=78$	18 19 20 ⋮ 76 77 78	18 19 20 ⋮ 77 78	$60+7/2+37/2$ $=81$	7	37	$60+\frac{30}{2}+\frac{37}{2}$ $=93$	30	37
5	$60+\frac{39}{2}+\frac{39}{2}$ $=98$	39	39	$60+\frac{3}{2}+\frac{18}{2}$ $=70$	3	18	$60+\frac{15}{2}+\frac{18}{2}$ $=76$	15	18
6									

next