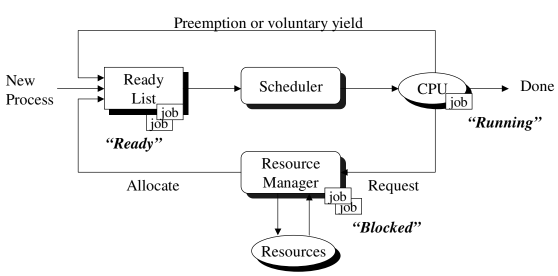
Chapter 7: Scheduling

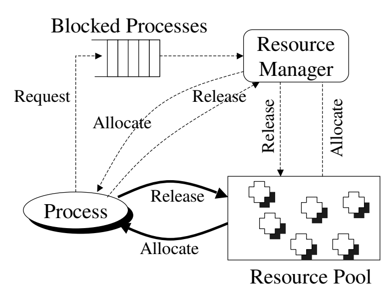
Process Scheduler

- Why do we even need to a process scheduler ?
  - In simplest form, CPU must be *shared* by
    - OS
    - Application
  - In reality, [multiprogramming]
    - OS : many separate pieces (processes)
    - Many Applications
- Scheduling [Policy] addresses…
  - When to remove a process from CPU ?
  - Which ready process to allocate the CPU to ?

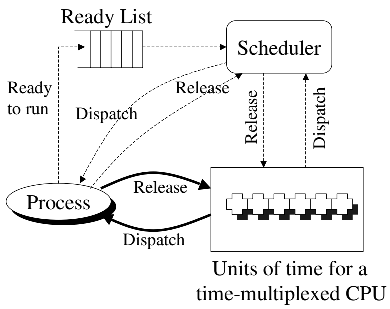
Model of Process Execution



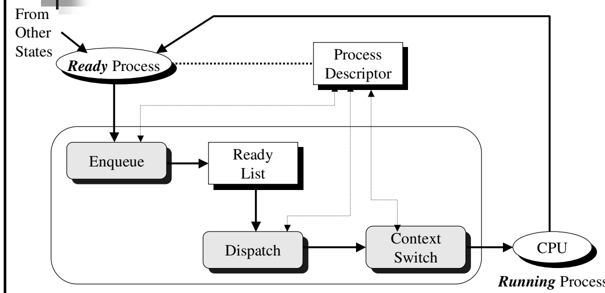
Recall Resource Manager



Scheduler as CPU Res Mgr



Scheduler Components

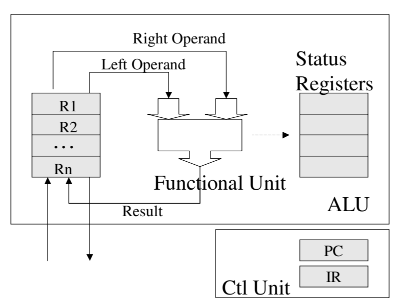


Context Switch

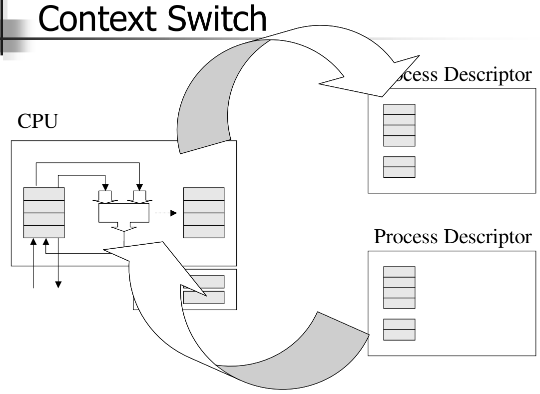
- Processes are switched out using <u>Context Switching</u>
- Context Switch:
  - **Save** pertinent info for current process
    - PC, Register, Status, etc.
  - **Update** PC, Register, Status, etc.
    - with info for process selected to run
- Switching User Process
  - 2 Context switches (CTX)

Process 1 running
CTX
Dispatcher : selects next process
CTX
Process 2 running

Process Context



Context Switch



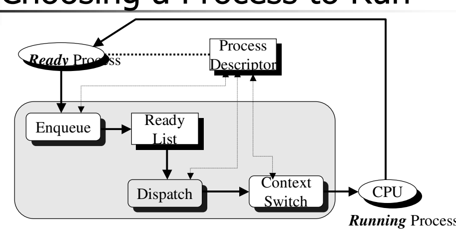
Invoking the Scheduler

- Need a *mechanism* to call the scheduler
- Voluntary call
  - Process blocks itself
  - Calls the scheduler
- Involuntary call
  - External force (interrupt) blocks the process
  - Calls the scheduler

Contemporary Scheduling

- Involuntary CPU sharing – timer interrupts
  - *Time quantum* determined by interval timer – usually fixed size for every process using the system
  - Sometimes called the *time slice length*

Choosing a Process to Run



- *Mechanism* never changes
- Strategy = *policy* the dispatcher uses to select a process from the ready list
- Different policies for different requirements

Policy Considerations

- Policy can control/influence:
  - CPU utilization
  - Average time a process waits for service
  - Average amount of time to complete a job
- Could strive for any of:
  - Equitability
  - Favor very short or long jobs
  - Meet priority requirements
  - Meet deadlines

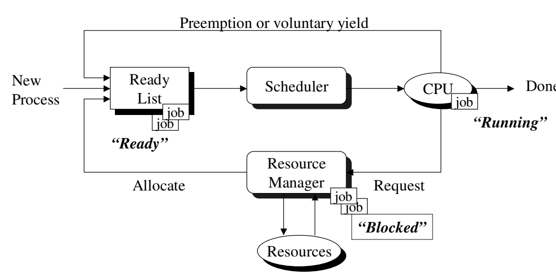
Optimal Scheduling

- Suppose the scheduler knows each process p_i's *service time*, $\tau(p_i)$ -- or it can estimate each $\tau(p_i)$:
- Policy can optimize on any criteria, e.g.,
  - CPU utilization
  - Waiting time
  - Deadline
- To find an *optimal schedule*:
  - Have a finite, fixed # of p_i
  - Know $\tau(p_i)$ for each p_i
  - Enumerate all schedules, then choose the best

However ...

- The $\tau(p_i)$ are almost certainly just estimates
- General algorithm to choose optimal schedule is $O(n^2)$
- Other processes may arrive while these processes are being serviced
- Usually, optimal schedule is only a *theoretical benchmark* – scheduling policies try to *approximate* an optimal schedule

Model of Process Execution



Selection Strategies

- Motivation
  - To "optimize" some aspect of system behavior
- Considerations
  - Priority of process
    - External : assigned
    - Internal : aging
  - Fairness : no starvation
  - Overall Resource Utilization

  …

Selection Strategies…

- Considerations…
  - Turnaround time
    - Average time / job
  - Throughput
    - Jobs / time unit
  - Response time
  - System availability
  - Deadlines

Talking About Scheduling ...

- Let $P = \{p_i \mid 0 \leq i < n\}$ = set of processes
- Let $S(p_i) \in$ {running, ready, blocked}
- Let $\tau(p_i)$ = Time process needs to be in running state (the *service time*)
- Let $W(p_i)$ = Time p_i is in ready state before first transition to running (*wait time*)
- Let $T_{TRnd}(p_i)$ = Time from p_i first enter ready to last exit ready (*turnaround time*)
- Batch *Throughput rate* = inverse of avg T_{TRnd}
- Timesharing response time = $W(p_i)$

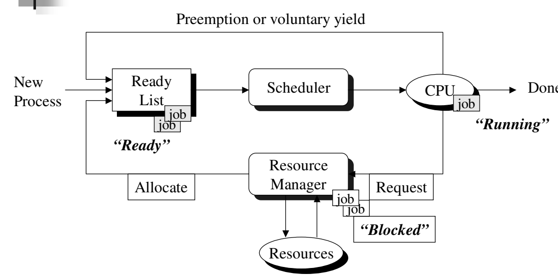
Definition & Terms

- Time Quantum
  - Amount of time between timer interrupts
  - Also called Time Slice
- Service Time τ (P_i)
  - Amount of time process needs to be in Running state (acquired CPU) before it is completed
- Wait Time W (P_i)
  - Time a process spends waiting in the Ready state before its *first* transition to the Running state

Definition & Terms…

- Turnaround Time T (P_i)
  - Amount of time between moment process first enters Ready state and the moment the process exits Running state for the last time (completed)
- Service time, Wait time & Turnaround time are measurable metrics used to compare scheduling algorithms

Simplified Model



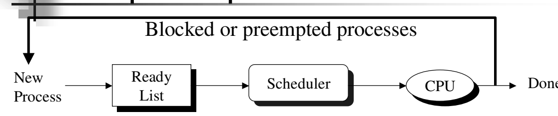
- Simplified, but still provide analysis result
- Easy to analyze performance

Classes of Scheduling Algorithms

- 2 major classes
  - Non-preemptive
    - Run to completion
  - Preemptive
    - Process with highest priority always gets CPU

    Recall : Several ways to establish priority

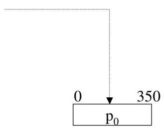
Nonpreemptive Schedulers



- Try to use the simplified scheduling model
- Only consider running and ready states
- Ignores time in blocked state:
  - "New process created when it enters ready state"
  - "Process is destroyed when it enters blocked state"
  - Really just looking at "small phases" of a process

First-Come-First-Served

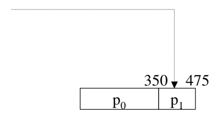
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0) = 350$ $W(p_0) = 0$

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

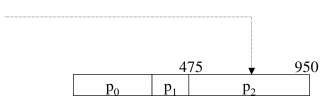


$T_{TRnd}(p_0) = \tau(p_0) = 350$ $W(p_0) = 0$

$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$ $W(p_1) = T_{TRnd}(p_0) = 350$

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



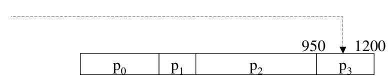
$T_{TRnd}(p_0) = \tau(p_0) = 350$ $W(p_0) = 0$

$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$ $W(p_1) = T_{TRnd}(p_0) = 350$

$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$ $W(p_2) = T_{TRnd}(p_1) = 475$

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0) = 350$ $W(p_0) = 0$

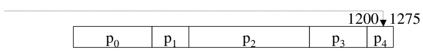
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$ $W(p_1) = T_{TRnd}(p_0) = 350$

$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$ $W(p_2) = T_{TRnd}(p_1) = 475$

$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$ $W(p_3) = T_{TRnd}(p_2) = 950$

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0) = 350$ — $W(p_0) = 0$

$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$ — $W(p_1) = T_{TRnd}(p_0) = 350$

$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$ — $W(p_2) = T_{TRnd}(p_1) = 475$

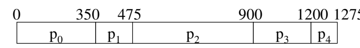
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$ — $W(p_3) = T_{TRnd}(p_2) = 950$

$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$ — $W(p_4) = T_{TRnd}(p_3) = 1200$

FCFS Average Wait Time

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- *Easy to implement*
- Ignores service time, etc
- Not a great performer



$T_{TRnd}(p_0) = \tau(p_0) = 350$ — $W(p_0) = 0$

$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$ — $W(p_1) = T_{TRnd}(p_0) = 350$

$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$ — $W(p_2) = T_{TRnd}(p_1) = 475$

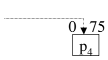
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$ — $W(p_3) = T_{TRnd}(p_2) = 950$

$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$ — $W(p_4) = T_{TRnd}(p_3) = 1200$

$W_{avg} = (0+350+475+950+1200)/5 = 2974/5 = 595$

Shortest Job Next

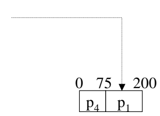
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_4) = \tau(p_4) = 75$ — $W(p_4) = 0$

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

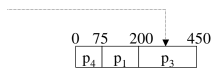


$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ — $W(p_1) = 75$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ — $W(p_4) = 0$

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



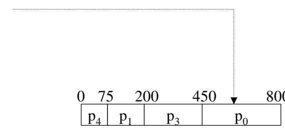
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $\quad W(p_1) = 75$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $\quad W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $\quad W(p_4) = 0$

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $\quad W(p_0) = 450$

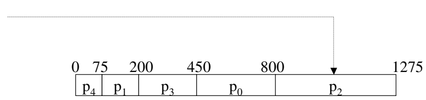
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $\quad W(p_1) = 75$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $\quad W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $\quad W(p_4) = 0$

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $\quad W(p_0) = 450$

$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $\quad W(p_1) = 75$

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$ $\quad W(p_2) = 800$

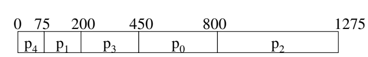
$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $\quad W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $\quad W(p_4) = 0$

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- *<u>Minimizes wait time</u>*
- May starve large jobs
- Must know service times



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $\quad W(p_0) = 450$

$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $\quad W(p_1) = 75$

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$ $\quad W(p_2) = 800$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $\quad W(p_3) = 200$

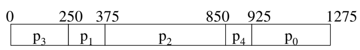
$T_{TRnd}(p_4) = \tau(p_4) = 75$ $\quad W(p_4) = 0$

$W_{avg} = (450+75+800+200+0)/5 = 1525/5 = 305$

Priority Scheduling

i	$\tau(p_i)$	Pri
0	350	5
1	125	2
2	475	3
3	250	1
4	75	4

- *<u>Reflects importance of external use</u>*
- May cause starvation
- Can address starvation with aging



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_4)+\tau(p_2)+\tau(p_1))+\tau(p_3) = 350+75+475+125+250 = 1275$

$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_3) = 125+250 = 375$

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_1)+\tau(p_3) = 475+125+250 = 850$

$T_{TRnd}(p_3) = \tau(p_3) = 250$

$T_{TRnd}(p_4) = \tau(p_4)+ \tau(p_2)+ \tau(p_1)+\tau(p_3) = 75+475+125+250 = 925$

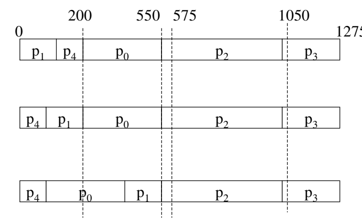
$W(p_0) = 925$
$W(p_1) = 250$
$W(p_2) = 375$
$W(p_3) = 0$
$W(p_4) = 850$

$W_{avg} = (925+250+375+0+850)/5 = 2400/5 = 480$

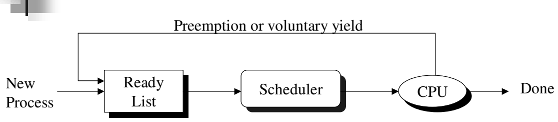
Deadline Scheduling

i	$\tau(p_i)$	Deadline
0	350	575
1	125	550
2	475	1050
3	250	(none)
4	75	200

- *<u>Allocates service by deadline</u>*
- May not be feasible



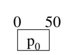
Preemptive Schedulers



- Highest priority process is guaranteed to be running at all times
  - Or at least at the beginning of a time slice
- Dominant form of contemporary scheduling
- But complex to build & analyze

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100

p_0	p_1

$W(p_0) = 0$
$W(p_1) = 50$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100

p_0	p_1	p_2

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200

p_0	p_1	p_2	p_3

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200

p_0	p_1	p_2	p_3	p_4

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200 300

p_0	p_1	p_2	p_3	p_4	p_0

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200 300 400 475

p_0	p_1	p_2	p_3	p_4	p_0	p_1	p_2	p_3	p_4

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$T_{TRnd}(p_4) = 475$
$W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200 300 400 475 550

p_0	p_1	p_2	p_3	p_4	p_0	p_1	p_2	p_3	p_4	p_0	p_1

$W(p_0) = 0$
$T_{TRnd}(p_1) = 550$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$T_{TRnd}(p_4) = 475$
$W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100 200 300 400 475 550 650

p_0	p_1	p_2	p_3	p_4	p_0	p_1	p_2	p_3	p_4	p_0	p_1	p_2	p_3

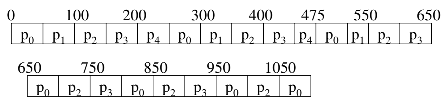
650 750 850 950

p_0	p_2	p_3	p_0	p_2	p_3

$W(p_0) = 0$
$T_{TRnd}(p_1) = 550$
$W(p_1) = 50$
$W(p_2) = 100$
$T_{TRnd}(p_3) = 950$
$W(p_3) = 150$
$T_{TRnd}(p_4) = 475$
$W(p_4) = 200$

Round Robin (TQ=50)

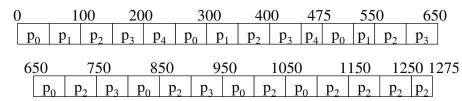
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = 1100$ $W(p_0) = 0$
$T_{TRnd}(p_1) = 550$ $W(p_1) = 50$
$W(p_2) = 100$
$T_{TRnd}(p_3) = 950$ $W(p_3) = 150$
$T_{TRnd}(p_4) = 475$ $W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

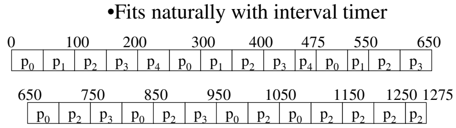


$T_{TRnd}(p_0) = 1100$ $W(p_0) = 0$
$T_{TRnd}(p_1) = 550$ $W(p_1) = 50$
$T_{TRnd}(p_2) = 1275$ $W(p_2) = 100$
$T_{TRnd}(p_3) = 950$ $W(p_3) = 150$
$T_{TRnd}(p_4) = 475$ $W(p_4) = 200$

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- Equitable
- Most widely-used
- Fits naturally with interval timer



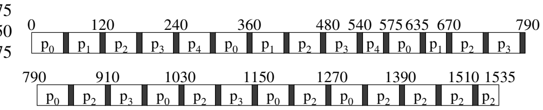
$T_{TRnd}(p_0) = 1100$ $W(p_0) = 0$
$T_{TRnd}(p_1) = 550$ $W(p_1) = 50$
$T_{TRnd}(p_2) = 1275$ $W(p_2) = 100$
$T_{TRnd}(p_3) = 950$ $W(p_3) = 150$
$T_{TRnd}(p_4) = 475$ $W(p_4) = 200$

$T_{TRnd-avg} = (1100+550+1275+950+475)/5 = 4350/5 = 870$
$W_{avg} = (0+50+100+150+200)/5 = 500/5 = 100$

RR with Overhead=10 (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

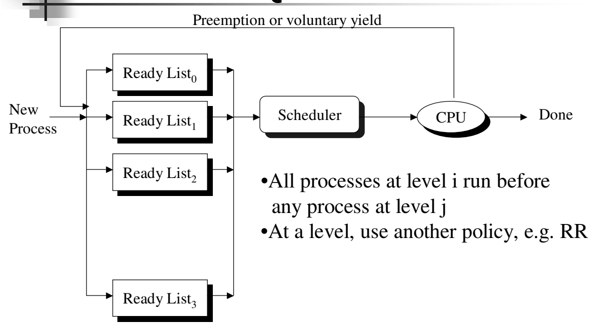
- Overhead must be considered



$T_{TRnd}(p_0) = 1320$ $W(p_0) = 0$
$T_{TRnd}(p_1) = 660$ $W(p_1) = 60$
$T_{TRnd}(p_2) = 1535$ $W(p_2) = 120$
$T_{TRnd}(p_3) = 1140$ $W(p_3) = 180$
$T_{TRnd}(p_4) = 565$ $W(p_4) = 240$

$T_{TRnd-avg} = (1320+660+1535+1140+565)/5 = 5220/5 = 1044$
$W_{avg} = (0+60+120+180+240)/5 = 600/5 = 120$

Multi-Level Queues



- All processes at level i run before any process at level j
- At a level, use another policy, e.g. RR

Contemporary Scheduling

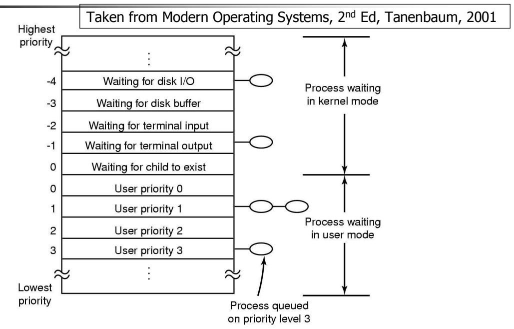
- Involuntary CPU sharing -- timer interrupts
  - *Time quantum* determined by interval timer -- usually fixed for every process using the system
  - Sometimes called the *time slice length*
- *Priority*-based process (job) selection
  - Select the highest priority process
  - Priority reflects policy
- With *preemption*
- Usually a variant of *Multi-Level Queues*

BSD 4.4 Scheduling

- Involuntary CPU Sharing
- Preemptive algorithms
- 32 Multi-Level Queues
  - Queues 0-7 are reserved for system functions
  - Queues 8-31 are for user space functions
  - `nice` influences (but does not dictate) queue level

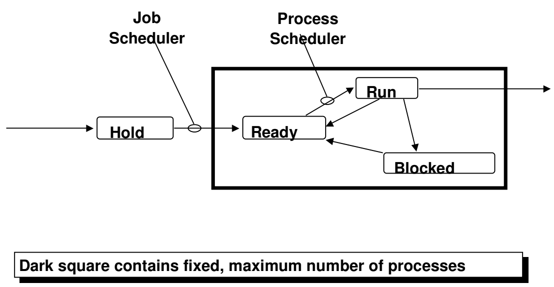
UNIX Scheduler

Taken from Modern Operating Systems, 2nd Ed, Tanenbaum, 2001



The UNIX scheduler is based on a multilevel queue structure

Process Life Cycle



Job and Process Scheduler

Job Scheduler

- Controls when jobs will be allowed to contend the CPU
- Most popular techniques

  FIFO First in, first out

  SJF Shortest job first

Process Scheduler

- Controls when individual jobs (processes) will actually get the CPU
- Only interesting in multi-programming
- Most popular technique is Round Robin
  - Give each process one time slice in turn until complete

Turnaround and Weighted Turnaround Time

Let: N be number of jobs

A_i be arrival time of i-th job

F_i be finish time of i-th job

Turnaround time for i^{th} job: $T_i = F_i - A_i$

Average turnaround time for i^{th} job: $T = \Sigma T_i / N$

Weighted turnaround time for i^{th} job:

$$WT_i = (F_i - A_i) / (\text{Service-time})_i$$

Average Weighted Turnaround time:

$$WT = \Sigma WT_i / N$$

Processor Sharing (PS) "Theoretical" Scheduling Algorithm

- Limit of RR as time quantum goes to zero.
- Like giving each CPU cycle to a different process, in round robin fashion.
- *N* processes scheduled by PS
  - Each job runs on dedicated *N*-fold slower CPU.
  - Thus, READY = RUNNING.
- CPU Time "shared" equally among processes

Scheduling Example 2

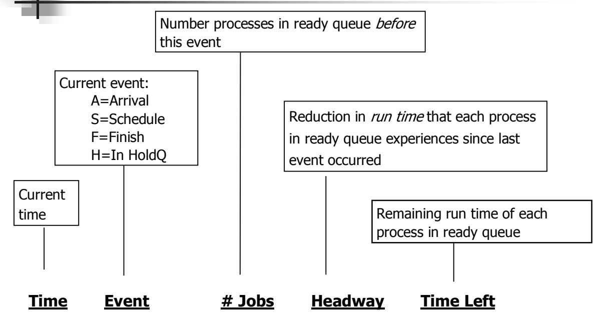
Assume:

Multiprogramming FIFO Job Scheduling

Processor Sharing Process Scheduling

Job	Arrives	Run Time
1	10.0	0.3
2	10.2	0.5
3	10.4	0.1
4	10.5	0.4
5	10.8	0.1

Definitions



Example 2 Continued

Time	Event	# Jobs	Headway	Time Left
10.0	1 A,S			1 0.3
10.2	2 A,S	1	0.2	1 0.1
				2 0.5
10.4	1 F	2	0.1	2 0.4
	3 A,S			3 0.1
10.5	4 A,S	2	0.05	2 0.35
				3 0.05
				4 0.4
10.65	3 F	3	0.05	2 0.3
				4 0.35

Example 2 Continued…

Time	Event	# Jobs	Headway	Time Left
10.8	5 A,S	2	0.075	2 0.225
				4 0.275
				5 0.1
11.1	5 F	3	0.1	2 0.125
				4 0.175
11.35	2 F	2	0.125	4 0.05
11.40	4 F	1	0.05	

T and W for Example 2

Job	Run	Start	Finish	Ti	WTi
1	0.3	10.0	10.4	0.4	1.33
2	0.5	10.2	11.35	1.15	2.3
3	0.1	10.4	10.65	0.25	2.5
4	0.4	10.5	11.4	0.9	2.25
5	0.1	10.8	11.1	0.3	3.0
	1.4			**3.0**	**11.38**

T = 0.6 **WT = 2.276**

Check:

Because CPU was never idle, 1.4 + 10.0 must equal time of last event (11.4)

Scheduling Example 4

Assume:

FIFO Job Scheduling 100 K Main Memory

5 Tape Drives Processor Sharing Process Scheduling

Job	Arrives	Run Time	Memory	Tapes
1	1.0	0.5	30	2
2	1.2	1.0	50	1
3	1.3	1.5	50	1
4	1.4	2.0	20	2
5	1.7	0.5	30	3
6	2.1	1.0	30	2

Example 4 Continued

Time	Event	# Jobs	HWay	MM	Tapes	Time Left
1.0	**1 A,S**			**70**	**3**	**1 0.5**
1.2	**2 A,S**	**1**	**0.2**	**20**	**2**	**1 0.3**
						2 1.0
1.3	**3 A,H**	**2**	**0.05**	**20**	**2**	**1 0.25**
						2 0.95
1.4	**4 A,S**	**2**	**0.05**	**0**	**0**	**1 0.2**
						2 0.9
						4 2.0
1.7	**5 A,H**	**3**	**0.1**	**0**	**0**	**1 0.1**
						2 0.8
						4 1.9
2.0	**1 F**	**3**	**0.1**	**30**	**2**	**2 0.7**
						4 1.8

Example 4 Continued …

Time	Event	# Jobs	HWay	MM	Tapes	Time Left
2.1	**6 A,S**	**2**	**0.05**	**0**	**0**	**2 0.65**
						4 1.75
						6 1.0
4.05	**2 F**	**3**	**0.65**	**50**	**1**	**4 1.1**
	3 S			**0**	**0**	**6 0.35**
						3 1.5
5.1	**6 F**	**3**	**0.35**	**30**	**2**	**4 0.75**
						3 1.15
6.6	**4 F**	**2**	**0.75**	**50**	**4**	**3 0.4**
	5 S			**20**	**1**	**5 0.5**
7.4	**3 F**	**2**	**0.4**	**70**	**2**	**5 0.1**
7.5	**5 F**	**1**	**0.1**	**100**	**5**	

T and W for Example 4

Job	Run	Arrives	Finish	Ti	WTi
1	0.5	1.0	2.0	1.0	2.0
2	1.0	1.2	4.05	2.85	2.85
3	1.5	1.3	7.4	6.1	4.06
4	2.0	1.4	6.6	5.2	2.6
5	0.5	1.7	7.5	5.8	11.6
6	2.1	2.1	5.1	3.0	3.0
				23.95	**26.11**

T = 3.99 WT = 4.35