IT1313

Operating Systems and Administration

Chapter 6

Process Scheduling

Instructional Learning Outcomes:

- After this lesson, you should be able to :
  - Explain how scheduling works in the management of processes.
  - Apply the various scheduling strategies and how to evaluate them.

Scheduling

- Task of managing CPU sharing among a pool of ready processes/threads.
- Possible only with context switching facility.
- The <u>*scheduler*</u> chooses one of the ready threads to allocate to the CPU when it is available.
- The <u>*scheduling policy*</u> determines when it is time for a thread to be removed from the CPU and allocate it to another thread.
- The <u>*scheduling mechanism*</u> determines how the process manager can determine it is time to interrupt the CPU, and how a thread can be allocated to and removed from the CPU.

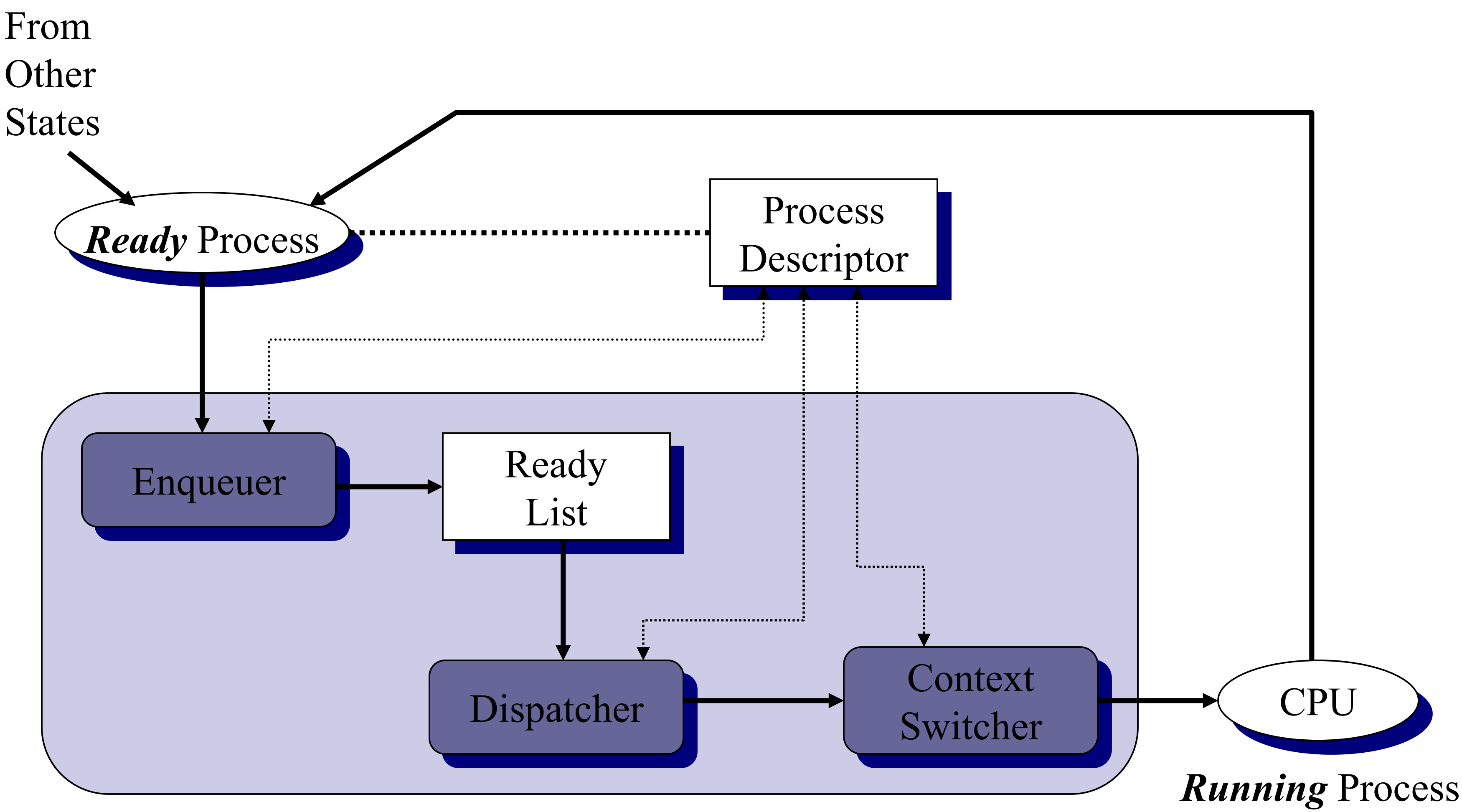
Scheduling Mechanisms

- The scheduling mechanism is composed of three logical parts : the enqueuer, the dispatcher and the context switcher.
  - When a process/thread is changed into the ready state, the **enqueuer** enqueues the corresponding descriptor into a list of processes that are waiting for the CPU (or the ready list). The enqueuer may place the new process anywhere in the list, depending on the **scheduling policy**.

Scheduling Mechanisms

- When the scheduler switches the CPU from executing one process to another, the **context switcher** saves the contents of all CPU registers (PC, IR, condition status, processor status, and ALU status) for the thread being removed into its descriptor.
- The dispatcher is invoked after the current process is removed from the CPU. The **dispatcher** removes one of the threads from the ready list and then allocates it to the CPU by loading its CPU registers in the thread’s descriptor into the CPU.

The Scheduler



Scheduling Performance

- Scheduler and its scheduling policy can have a dramatic effect on the performance of a multi-programmed computer.
- The time a process takes to finish execution is dependent on how often it gets to execute on the CPU as despatched by the scheduler.
- Aim is to decrease average time a process takes to execute in the computer and to increase the throughput of a computer in terms of the number of processes get executed per unit time.

Scheduling Evaluation

- $W(p_i)$ = Total time P_i spent waiting in ready list (*wait time*)
- Let $T_{TRnd}(p_i)$ = Time from p_i first enter ready to last exit ready (*turnaround time*)

Nonpreemptive (voluntary) Schedulers

- First-Come-First-Served (FCFS)
  - Scheduler picks up first job to arrive in the ready list.
- Shortest Job First (SJF) (Nonpreemptive)
  - Scheduler picks up shortest job to arrive in the ready list.
- Priority (PR) (Nonpreemptive)
  - Scheduler picks up the job with highest priority in the ready list.

Preemptive (involuntary) Schedulers

- Shortest Job First (SJF) (Preemptive)
  - Scheduler picks up shortest job to arrive in the ready list.
- Priority (PR) (Preemptive)
  - Scheduler picks up the job with the highest priority in the ready list.
- Round Robin (RR)
  - Scheduler gives a short time-slice to each job.
- Multi-level Queues
  - Implement multiple queues (eg. Process all foreground processes before background processses, or 80% timeslice to foreground processes, 20% to background processes).

First-Come First-Served (FCFS)

- Concept
  - The process to request first will be allocated the CPU first
  - It is non-preemptive
  - Using a FCFS queue, PCB of new process will be linked to the tail of the queue
  - When CPU is free, process at the head of the FCFS queue will be allocated the CPU
- Advantages:
  - Simplest CPU scheduling algorithm
- Disadvantage
  - Convoy effect: short process behind long process and waiting for the long process to finished. This results in lower CPU and devices utilization.

Shortest-Job-First (SJF)

- Concept:
  - Each process is associated with the length of its service time. When the CPU is available, it is assigned to the process that has the smallest (remaining) service time.
- Two methods:
  - nonpreemptive - Once CPU is allocated to a process it cannot be preempted until it completes its service time.
  - preemptive - A current process is preempted if a new process arrives with service time length lesser than the current process's <u>remaining</u> service time.
  - This scheme is also known as the Shortest-Remaining-Time - First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Priority Scheduling

- Concept
  - A priority number (integer) is associated with each process. The CPU is allocated to the process with the highest priority. Equal- priority processes are scheduled in FCFS order.
    - preemptive
    - nonpreemptive
- Problem: Starvation
  - Low priority processes may never be executed.
- Solution: Aging
  - Increase the priority of the process as time progresses.

Round-Robin (RR)

- Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- New processes are added to the tail of the ready queue, which is a FCFS circular queue.
- Fast response time. It is good for time-sharing system.
- If service time is < 1 time quantum, process release CPU voluntarily.
- If service time > 1 time quantum, context switch will be executed.
- Performance. It depends on size of time quantum:
  - large quantum (q). This is equivalent to FIFO.
  - small quantum (q). It must be large with respect to the context switch, otherwise overhead is too high.

Multilevel Queue

- Processes are classified into groups based on some property of the process.
- Ready queue is partitioned into separate queues such as foreground (interactive), background (batch).
- The processes are permanently assigned to one queue.
- Each queue has its own scheduling algorithm. For example:
  - RR for foreground queue
  - FCFS for background.

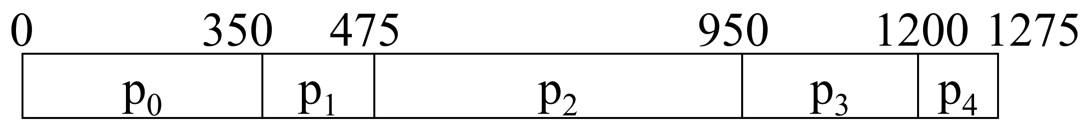
CPU Schedules

- There is no practical way to determine which policy is the best.
- To study policy performance, we need to consider simplified model, based on a fixed order of processes, which reflect the actual kind of processes a computer can expect.
- To do so, we are provided with :
  - A schedule of processes, their arrival time and CPU service time.
- We need to analyze and produce the following :
  - Gantt chart
  - Average waiting and turnaround time

CPU Schedules

- A Gantt chart is a ‘chart’ showing the actual execution of a series of processes by the CPU.
  - It shows the start and end of execution of each process.
  - It does not show the arrival time of each process.
- Waiting time is defined as the total time which a process spends waiting for the CPU in the ready list.
- Turnaround time is the time from the arrival of the process to its completion by the CPU.
  - It includes the time it spent waiting for the CPU.

CPU Schedules



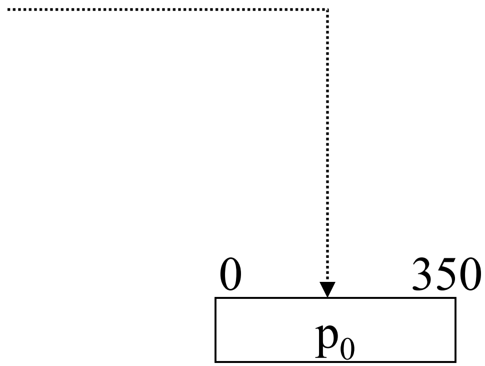
- The above example shows a typical Gantt chart.
- To find waiting time for p_2, we take start time for p_2 minus the arrival time of p_2.
  - Which is 475 – 0 = 475.
- Turnaround time is end time minus the arrival time,
  - which is 950 – 0 = 950.
- Average waiting time is *Total Waiting Time / Number of Processes*
- Average turnaround time is *Total Turanaround Time / Number of Processes*

CPU Schedules

- In the following slides, we will see how to compute the average turnaround time and average waiting time of the processes for the following scheduling algorithms:
  - FCFS
  - SJF (nonpreemptive)
  - Priority (nonpreemptive)
  - Round Robin
- We assume all the processes arrive at time 0.

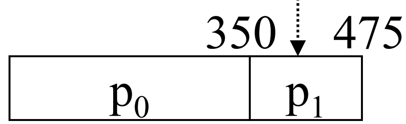
First-Come-First-Served

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



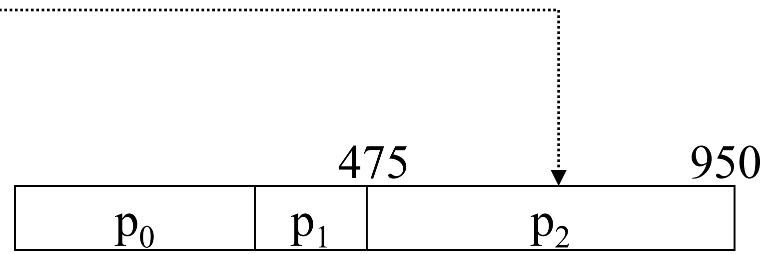
First-Come-First-Served

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



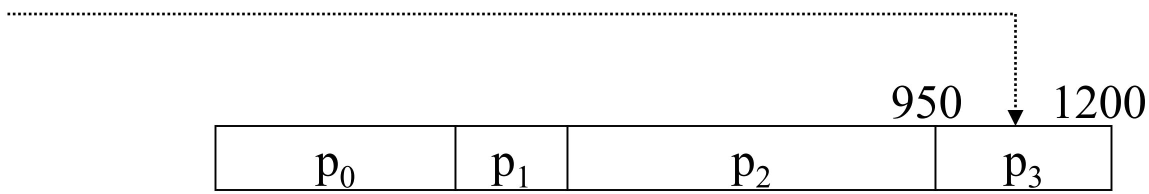
First-Come-First-Served

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



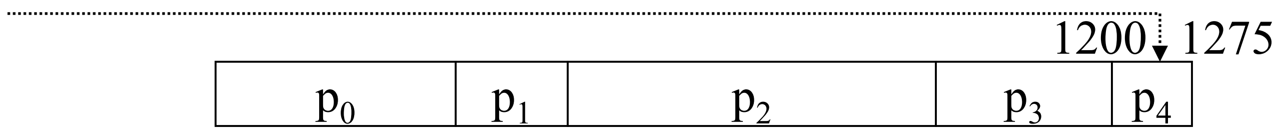
First-Come-First-Served

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



First-Come-First-Served

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



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

0	350	475	950	1200	1275
p_0	p_1	p_2	p_3	p_4	

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

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

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

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

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

$W(p_0) = 0$

$W(p_1) = T_{TRnd}(p_0) = 350$

$W(p_2) = T_{TRnd}(p_1) = 475$

$W(p_3) = T_{TRnd}(p_2) = 950$

$W(p_4) = T_{TRnd}(p_3) = 1200$

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

$T_{TRnd\ (avg)} = (350+475+950+1200+1275)/5 = 850$

Shortest Job First (Non-preemptive)

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

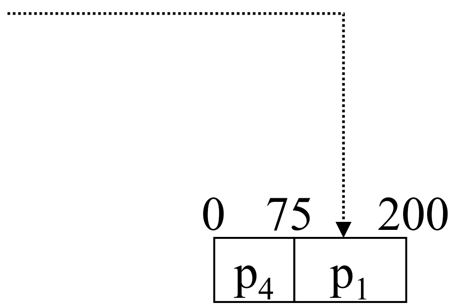
Assumption : All jobs arrive at the same time

0 75

p_4

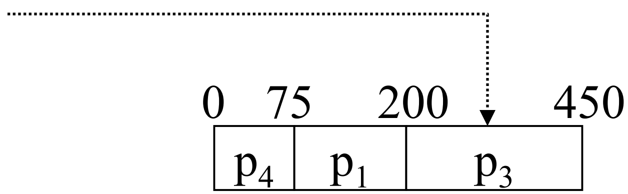
Shortest Job First (Non-preemptive)

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



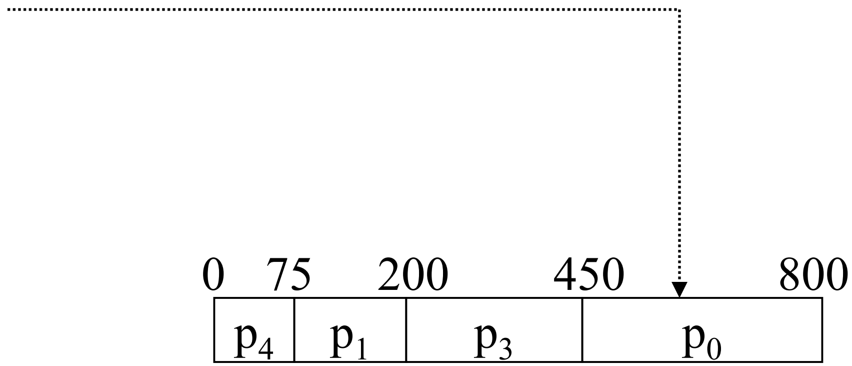
Shortest Job First (Non-preemptive)

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



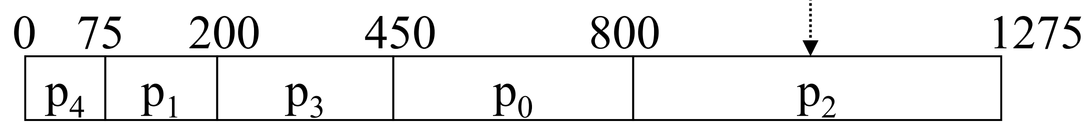
Shortest Job First (Non-preemptive)

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



Shortest Job First (Non-preemptive)

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



Shortest Job First (Non-preemptive)

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

0	75	200	450	800	1275
p_4	p_1	p_3	p_0	p_2	

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

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

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

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

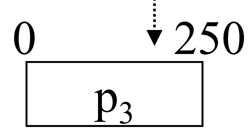
$T_{TRnd}(p_4) = \tau(p_4) = 75$ $\quad W(p_4) = 0$

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

$$T_{TRnd\ (avg)} = (800+200+1275+450+75)/5 = 560$$

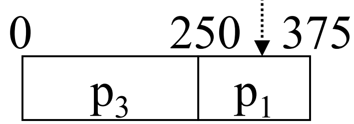
Priority Scheduling(Non-preemptive)

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



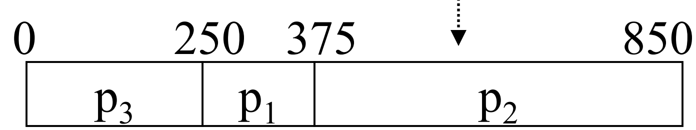
Priority Scheduling(Non-preemptive)

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



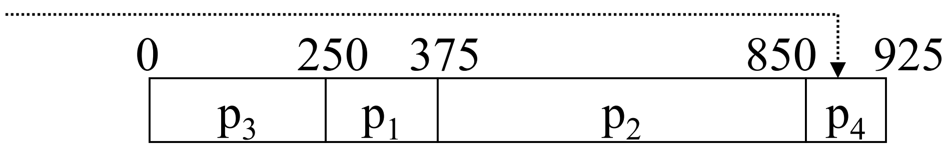
Priority Scheduling(Non-preemptive)

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



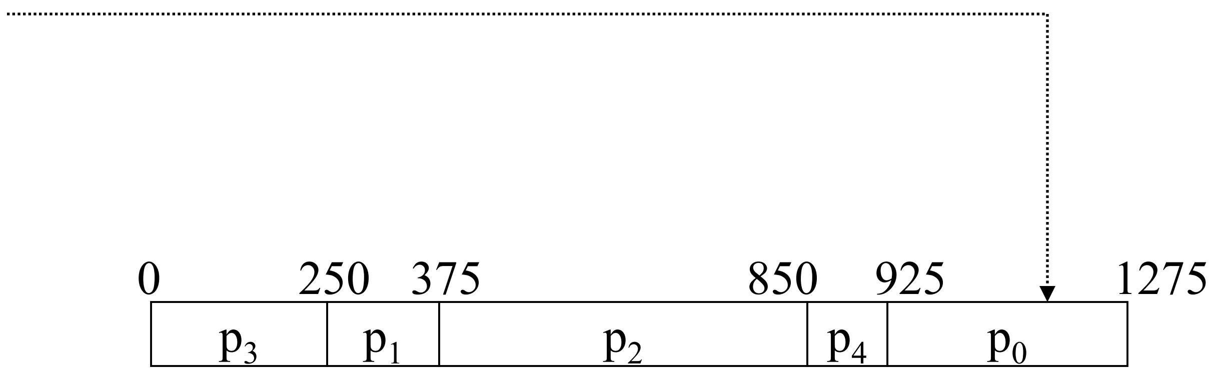
Priority Scheduling(Non-preemptive)

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



Priority Scheduling(Non-preemptive)

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



Priority Scheduling(Non-preemptive)

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

- *Reflects importance of external use*
- May cause starvation
- Can address starvation with aging

0	250	375	850	925	1275
p_3	p_1	p_2	p_4	p_0	

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

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

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

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

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

$W(p_0) = 925$

$W(p_1) = 250$

$W(p_2) = 375$

$W(p_3) = 0$

$W(p_4) = 850$

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

$T_{TRnd\ (avg)} = (1275+375+850+250+925)/5 = 735$

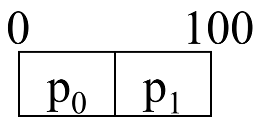
Round Robin (TQ=50)

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

0	50
p_0	

Round Robin (TQ=50)

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



Round Robin (TQ=50)

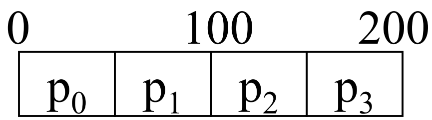
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

0 100

p_0	p_1	p_2

Round Robin (TQ=50)

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



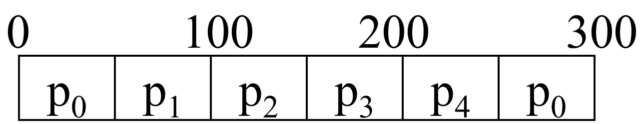
Round Robin (TQ=50)

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

Round Robin (TQ=50)

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



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

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

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

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 1050

p_0	p_2	p_3	p_0	p_2	p_3	p_0	p_2	p_0

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 1050 1150 1250 1275

p_0	p_2	p_3	p_0	p_2	p_3	p_0	p_2	p_0	p_2	p_2	p_2	p_2

Round Robin (TQ=50)

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

- Equitable (Fair)
- Most widely-used
- Fits naturally with interval timer

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 1050 1150 1250 1275

p_0	p_2	p_3	p_0	p_2	p_3	p_0	p_2	p_0	p_2	p_2	p_2	p_2

$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$
$T_{TRnd}(p2) = 1275$
$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

W(p0) = 0+200+175+125+100+100+50 =750
W(p1) = 50+200+175 =425
W(p2) = 100+200+150+100+100+100+50=800
W(p3) = 150+200+150+100+100=700
W(p4) = 200+200=400

$$\mathbf{W_{avg} = (750+425+800+700+400)/5 = 3075/5 = 615}$$

$$\mathbf{T_{TRnd\ (avg)} = (1100+550+1275+950+475)/5 = 870}$$

BSD 4.4 (Unix) 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

Windows Scheduling

- Involuntary CPU Sharing across threads
- Preemptive algorithms
- 32 Multi-Level Queues
  - Highest 16 levels are “real-time”
  - Next lower 15 are for system/user threads
    - Range determined by process base priority
  - Lowest level is for the idle thread

Conclusion

- Scheduling is a requirement for modern OS with multi-programming ability.
- Policy used has a dramatic effect on overall system performance.
- No single ‘pure’ scheduler is good for all kinds of jobs, so a combination is most often used to provide overall acceptable performance.