ARCOS Group
Universidad Carlos III de Madrid

Lesson 3
Process scheduling

Operating Systems
Computer Science and Engineering

Recommended reading

Base

1. Carretero 2020:
    1. Cap. 5
2. Carretero 2007:
    1. Cap. 3 and 4

Suggested

1. Tanenbaum 2006(en):
    1. Chap.3
2. Stallings 2005:
    1. 3.2, 3.3 and 3.5
3. Silberschatz 2006:
    1. 3.1 and 3.3

WARNING!

- This material is a script of the class but it is not the notes of the course.
- The books given in the bibliography together with what is explained in class represent the study material for the course syllabus.

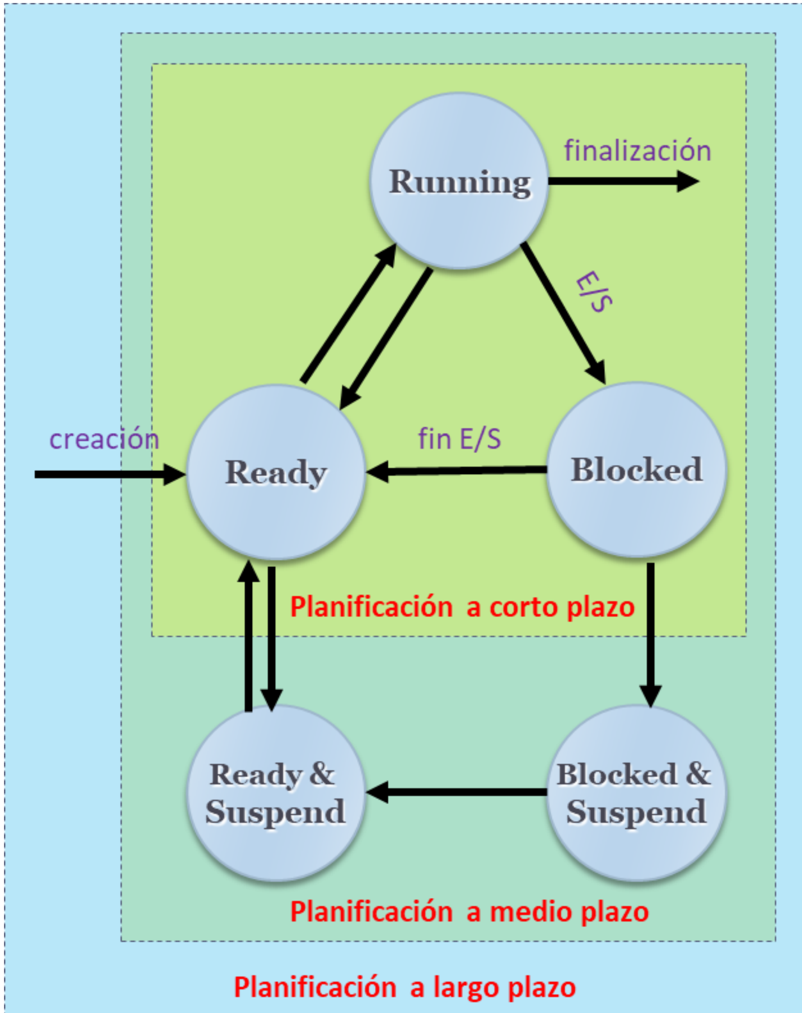
Contents

1. Basic concepts of operating system scheduling.
2. Scheduling and activation
3. Most common scheduling algorithms
   - FIFO, SJF, RR and PRIORITY.
4. Scheduling data structures in the kernel.
   - Scheduling in LINUX: aging.
5. System call for Process scheduling.

Contents

1. **Basic concepts of operating system scheduling.**
2. Scheduling and activation
3. Most common scheduling algorithms
   - FIFO, SJF, RR and PRIORITY.
4. Scheduling data structures in the kernel.
   - Scheduling in LINUX: aging.
5. System call for Process scheduling.

Process scheduling
scheduling **levels**



- Long-term
  - Add processes to be executed
    - Used for batch processing
- Mid-term
  - Processes to add/remove from main memory
- Short-term
  - Select the next process to be executed
    - Frequently invoked, fast

Process scheduling

goals of the scheduling algorithms (according to the system)

- All systems:
  - Fair – offers each process an equal share of the CPU
  - Expeditious – compliance with the distribution policy undertaken
  - Balanced – keep all parts of the system busy
- *Batch* systems:
  - Productivity – maximize the number of jobs per hour
  - Waiting time – minimize the time between issuance and completion of the job
  - CPU Usage – keep CPU busy all the time
- Interactive systems:
  - Response time – respond to requests as quickly as possible
  - Adjusted – meet user expectations
- Real-time systems:
  - Compliance with deadlines – prevent loss of data
  - Predictable – avoid quality degradation in multimedia systems

Process scheduling

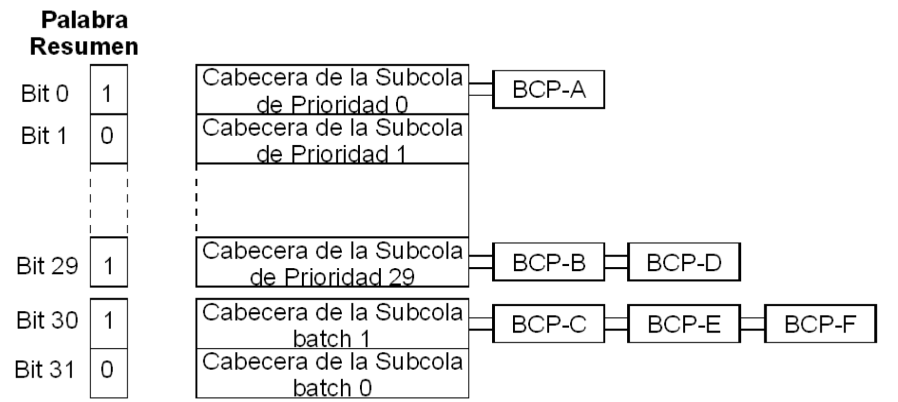
characteristics of scheduling algorithms (1/2)

- *Preemption:*
  - Without expulsion (non-appropriative):
    - The process conserves the CPU as long as desired.
    - Voluntary Context Switching (V.C.C.)
    - [A] Easy solution to resource sharing.
    - [D] A process can block the rest
    - Windows 3.1, Windows 95 (16 bits), NetWare, MacOS 9.x.
  - With expulsion (appropriative):
    - Requires a clock that periodically interrupts:
      - when the quantum of a process is spent it is switched to another one
    - (It adds) Involuntary Context Switching (C.C.I.)
    - [A] Improves interactivity
    - [D] Precise mechanisms for race conditions
    - AmigaOS (1985), Windows NT-XP-Vista-7, Linux, BSD, MacOS X

Process scheduling

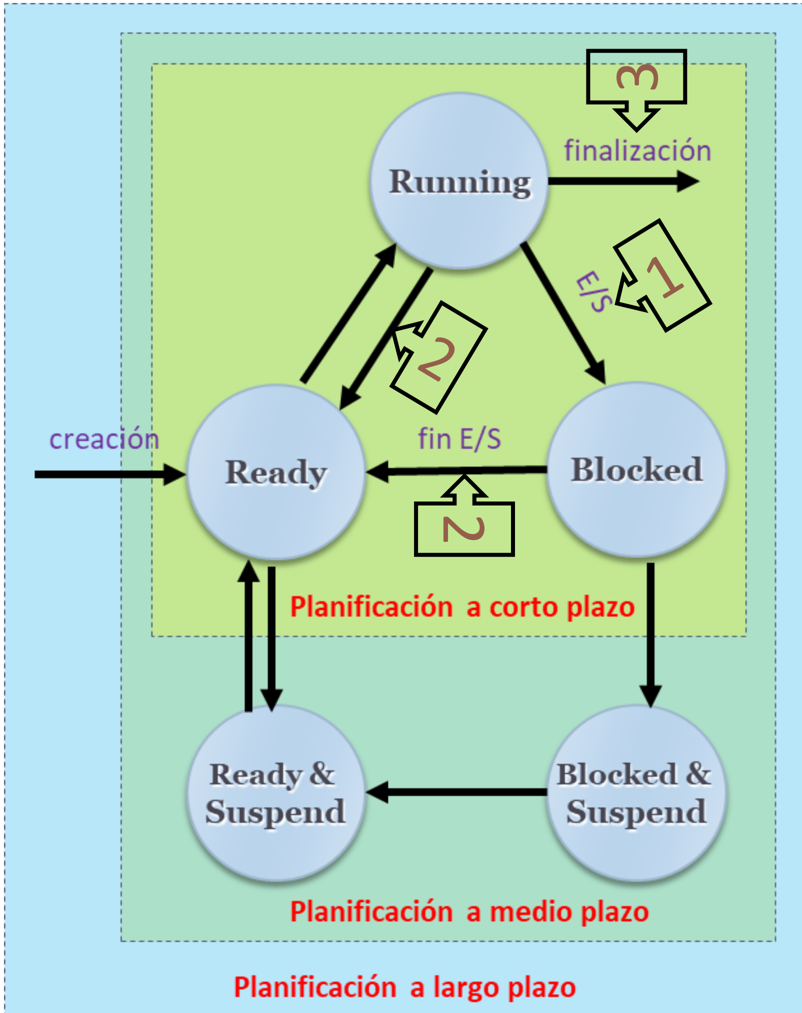
characteristics of scheduling algorithms (2/2)

- Classification of processes (BCP *) in queues:
  - Unclassified (single queue)
  - By type:
    - CPU-*bound*: + CPU usage bursts
    - IO-*bound*: + I/O standby bursts
  - By priority



- *CPU-aware*:
  - Affinity:
    - Processes have 'affinity' to a CPU: «better to return to the same CPU»
  - Symmetry:
    - Processes run on the CPU that have capabilities specific to that CPU

Process scheduling
scheduling **decision points**



- Possible transitions with rescheduling:
  1. Process is blocked (wait for event)
  2. when dealing with interruption:
     - Clock interrupt.
     - Interrupt of end of event waiting.
  3. End of process execution

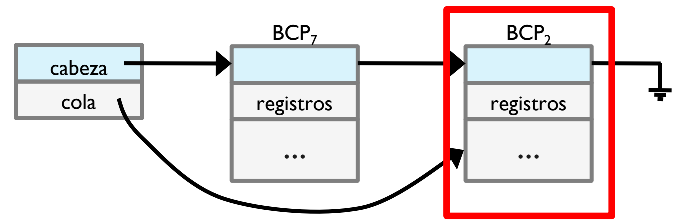
- Relationship between moment of decision and type:
  - Preemptive: **1**, **2** and **3**
  - NO preemptive: **1** and **3**

Contents

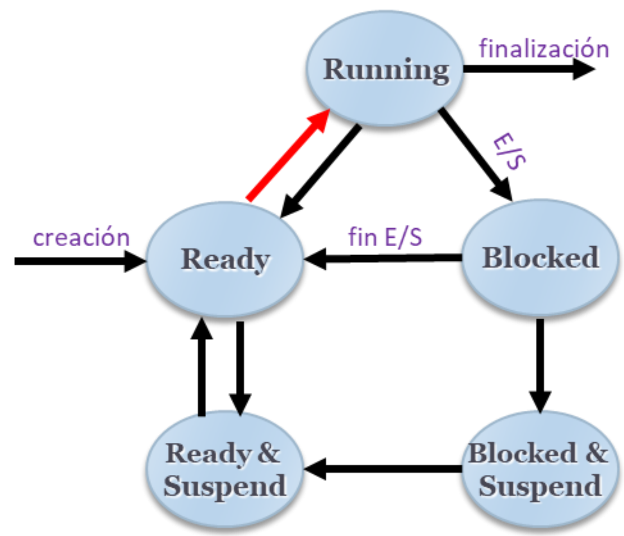
1. Basic concepts of operating system scheduling.
2. **Scheduling and activation**
3. Most common scheduling algorithms
   - FIFO, SJF, RR and PRIORITY.
4. Scheduling data structures in the kernel.
   - Scheduling in LINUX: aging.
5. System call for Process scheduling.

Scheduler and activator

- **Scheduler**:
  Selects the process to be executed from among those ready to be executed



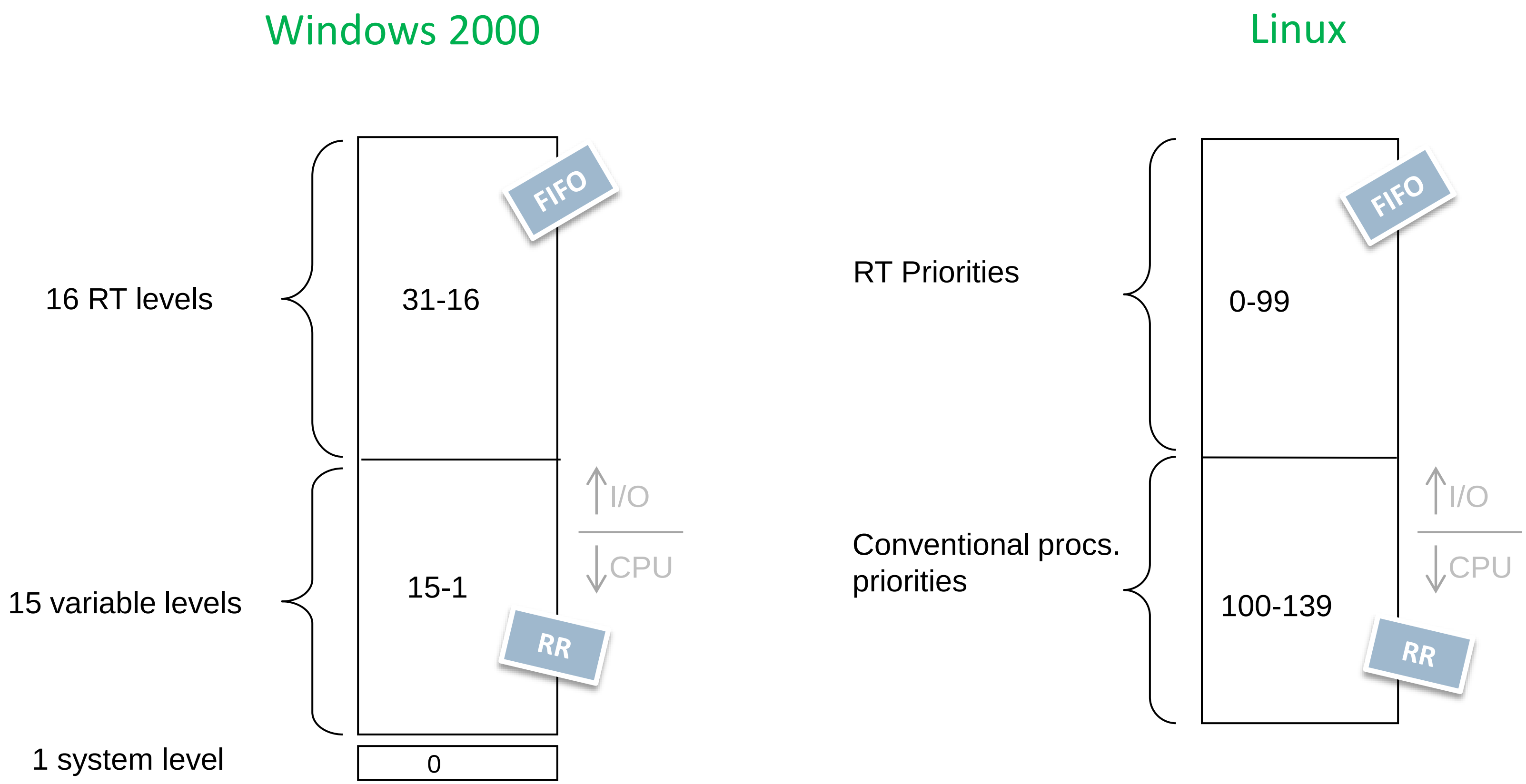
- **Activator**:
  Gives control to the process that the scheduler has selected (context switch - restore).



Policy versus mechanism

- Separation of <u>what</u> can be done from <u>how</u> it can be done
  - Normally, a process knows which thread has the highest priority, which thread will need the most I/O, etc.
- Use of parameterized scheduling algorithms
  - Kernel mechanism
- Parameters filled by the user processes
  - Policy set by user processes

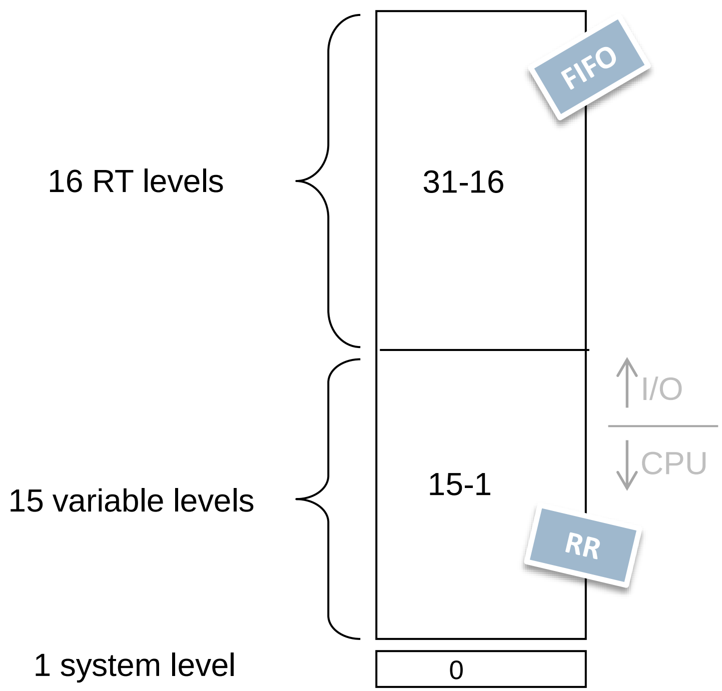
Multi-policy scheduling
Windows 2000 and Linux

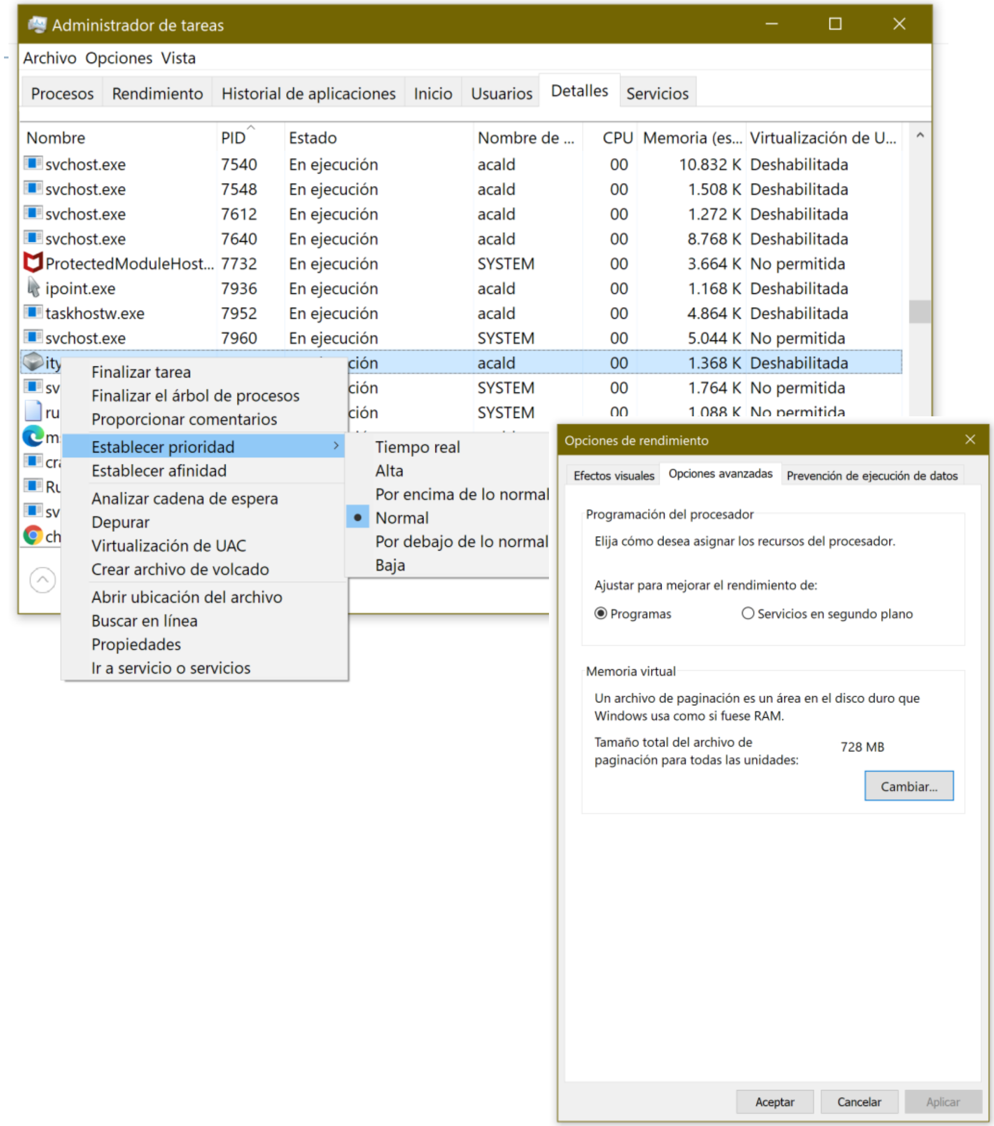


Multi-policy scheduling

Windows 2000 and Linux

Windows 2000

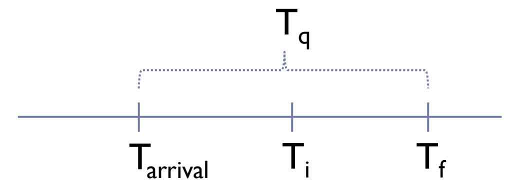




Contents

1. Basic concepts of operating system scheduling.
2. Scheduling and activation
3. **Most common scheduling algorithms**
   - **FIFO, SJF, RR and PRIORITY.**
4. Scheduling data structures in the kernel.
   - Scheduling in LINUX: aging.
5. System call for Process scheduling.

Scheduling: Measurements



Name	Calculations	Definition	Goal
CPU utilization	$U = TU / T$	Percentage of time CPU is used	Maximize
Productivity	$P = NW / T$	Number of jobs completed per unit of time	Maximize

Name	Calculations	Definition	Goal
$\mathbf{T_q}$ Turnaround time	$T_q = T_f - T_{arrival}$	Time a process spends in the system.	Minimize
$\mathbf{T_s}$ Service time	$T_s = T_{CPU} + T_{I/O}$	Time spent on productive tasks (CPU and Input/Output).	
$\mathbf{T_e}$ Waiting time	$T_e = T_q - T_s$	Time a process spends in waiting queues	
$\mathbf{T_n}$ Normalized turnaround time	$T_n = T_q / T_s$	Indicates the delay experienced. (return time / service time)	

Scheduling: main algorithms

	Name	How it works	Appropriative	Disadvantages
FCFS FIFO	*First to Come First to Serve*	First come, first serve	NO	• Penalizes short processes
SJF	*Shortest Job First*	Shortest job first	NO	• The duration of each job must be known in advance. • Possibility of starvation of long jobs (continuous arrival of short jobs).
RR	*Cíclico o Round-Robin*	Rotational shift	YES	• Context switches generate delay (even though *quantum time* >> *context switch time*)
Prio	*By priority*	Highest priority process is selected first	IFF blocked + priority higher than current	• If priority is fixed then starvation problem • Aging mechanisms

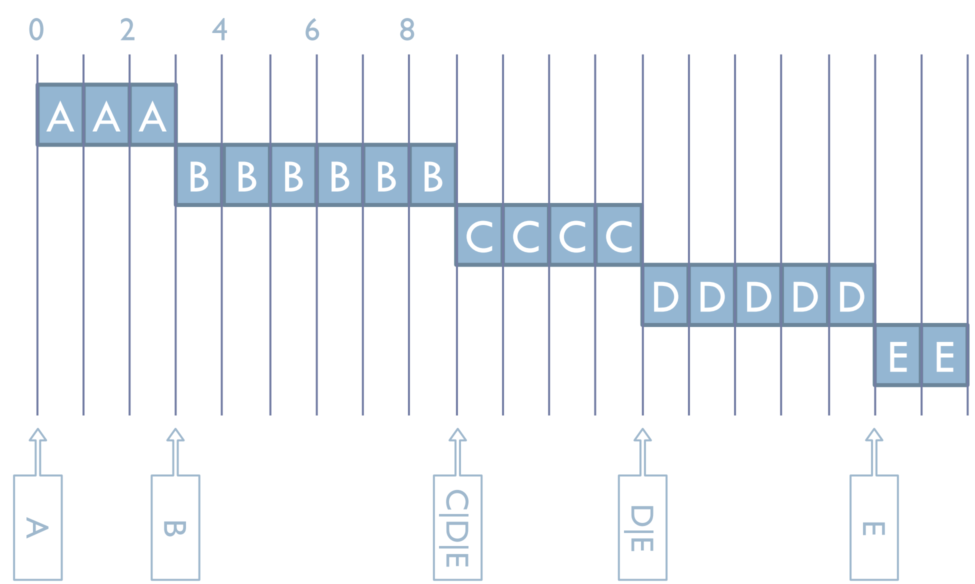
FCFS scheduling

- **Initials**: FCFS (A.K.A. FIFO)
- **Name**: First to Come First to Serve
- **How it works**: First come, first serve
- **Appropriative**: NO
- **Disadvantages**:
  - Penalize short processes

Example FCFS (1/2)

- While there are processes:
  - The process with the lowest arrival T. in the system is selected.
  - This process is executed during the service T.

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Example FCFS (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3					
B	2	6					
C	4	4					
D	6	5					
E	8	2					

1. Fill the start time (T_i) and end (T_f)
   1. T_i is **0** initially and then the $\mathbf{T_f}$ of the previous run.
   2. T_f is T_i + service time.
   3. FIFO: look $\mathbf{T_i}$ and take the next $\mathbf{T_i}$
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

Example FCFS (2/2)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3	**0**	**3**	**3**		
B	2	6	**3**	**9**	**7**		
C	4	4	**9**	**13**	**9**		
D	6	5	**13**	**18**	**12**		
E	8	2	**18**	**20**	**12**		

1. Fill the start time (T_i) and end (T_f)
   1. T_i is **0** initially and then the $\mathbf{T_f}$ of the previous run.
   2. T_f is T_i + service time.
   3. FIFO: look $\mathbf{T_i}$ and take the next $\mathbf{T_i}$
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

Example FCFS (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	3	3		
B	2	6	3	9	7		
C	4	4	9	13	9		
D	6	5	13	18	12		
E	8	2	18	20	12		

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

Example FCFS (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	3	3	**0**	**3/3=1**
B	2	6	3	9	7	**1**	**7/6=1.16**
C	4	4	9	13	9	**5**	**9/4=1.25**
D	6	5	13	18	12	**7**	**12/5=2.4**
E	8	2	18	20	12	**10**	**12/2=6**

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

Example FCFS (2/2)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	9	7	1	7/6=1.16
C	4	4	9	13	9	5	9/4=1.25
D	6	5	13	18	12	7	12/5=2.4
E	8	2	18	20	12	10	12/2=6

- T_e: Average waiting time: **4.6**
- T_n: Normalized average turnaround time: **2.5**

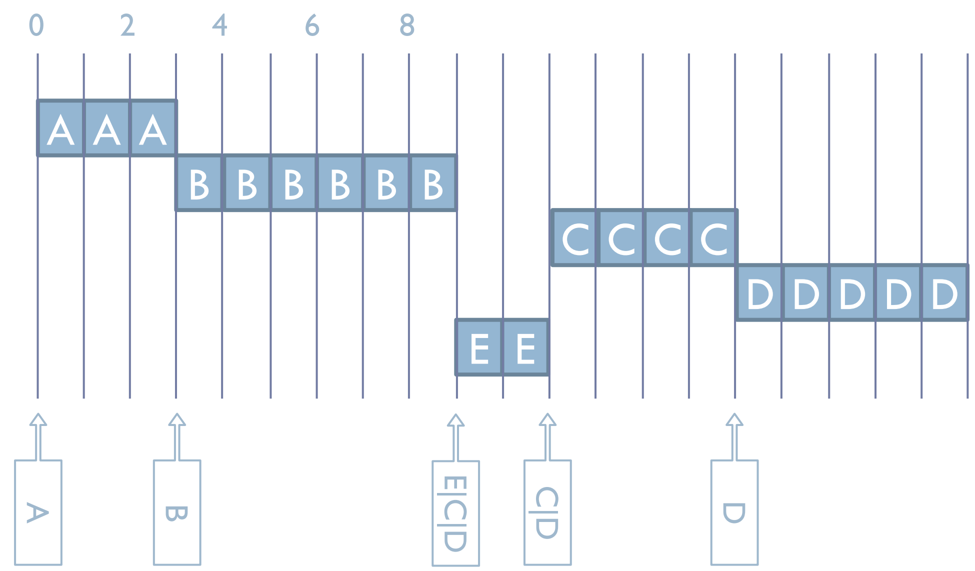
SJF scheduling

- **Initials**: SJF
- **Name**: Shortest Job First
- **How it works**: First the shortest job: the shortest job is selected
- **Appropriative**: NO
- **Disadvantages**:
  - The duration of each job must be known in advance.
  - Possibility of starvation of long jobs (continuous arrival of short jobs)

SJF Example (1/2)

- While there are processes:
  - The process with the lowest service T. in the system is selected.
  - This process is executed during the service T.

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



SJF Example (2/2)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3					
B	2	6					
C	4	4					
D	6	5					
E	8	2					

1. Fill the start time (T_i) and end (T_f)
   1. T_i is **0** first and then is **T_f** of previous process executed.
   2. T_f is T_i + service time
   3. SJF: look T_f and take the first process with lower or equal T_i
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

SJF Example (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	**0**	**3**	**3**		
B	2	6	**3**	**9**	**7**		
C	4	4	**11**	**15**	**11**		
D	6	5	**15**	**20**	**14**		
E	8	2	**9**	**11**	**3**		

1. Fill the start time (T_i) and end (T_f)
   1. T_i is **0** first and then is $\mathbf{T_f}$ of previous process executed.
   2. T_f is T_i + service time
   3. SJF: look T_f and take the first process with lower or equal T_i
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

SJF Example (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	3	3		
B	2	6	3	9	7		
C	4	4	11	15	11		
D	6	5	15	20	14		
E	8	2	9	11	3		

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

SJF Example (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	3	3	**0**	**3/3=1**
B	2	6	3	9	7	**1**	**7/6=1.16**
C	4	4	11	15	11	**7**	**11/4=2.75**
D	6	5	15	20	14	**9**	**14/5=2.8**
E	8	2	9	11	3	**1**	**3/2=1.5**

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

SJF Example (2/2)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	9	7	1	7/6=1.16
C	4	4	11	15	11	7	11/4=2.75
D	6	5	15	20	14	9	14/5=2.8
E	8	2	9	11	3	1	3/2=1.5

- T_e: Average waiting time: **~~4.6~~ 3.6**
- T_n: Normalized average turnaround time: **~~2.5~~ 1.84**

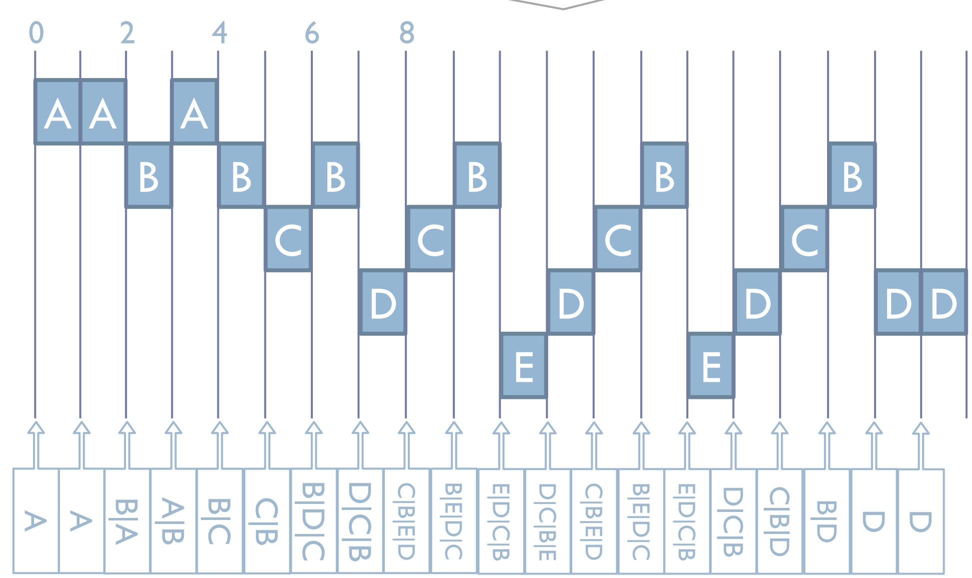
Cyclical or Round-Robin

- **Initials**: RR
- **Name**: *Cyclical or Round-Robin*
- **How it works**: Rotational shift
  - There is a FIFO queue with the processes ready to be executed
  - The first process to run on the processor is selected
  - The process will run until:
    - Ends its time quantum/slice and returns to the end of the ready queue.
    - It is blocked by an event and goes to the end of the corresponding blocked queue.
      - If process is blocked and event arrives then it passes to the end of the ready queue.
    - Ends the execution of the process within the time slice.
- **Appropriative**: YES
- **Disadvantages**:
  - Context switches generate delay
    (even if slice is longer than the context switch time)

Round-Robin example (q=1)

- While there are processes:
  - The first process ready to run is selected from the list
  - This process is executed during q=1 (slice) or end of execution
    - Those who arrive during the slice are put at the end of the ready list (and arrival time)
  - The executed process is put at the end of the ready list (if there is T_s)

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Round-Robin example (q=1)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3					
B	2	6					
C	4	4					
D	6	5					
E	8	2					

1. **Draw** and fill in start time (T_i) and end time (T_f)
   1. T_i is 0 first and then the moment it is scheduled for the first time.
   2. T_f is the instant when it has been fully executed.
   3. RR: [process arrival -> first in list], take first process in list + execute slice (q) + is placed last in the list
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

Round-Robin example (q=1)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3	**0**	**4**	**4**		
B	2	6	**2**	**18**	**16**		
C	4	4	**5**	**17**	**13**		
D	6	5	**7**	**20**	**14**		
E	8	2	**10**	**15**	**7**		

1. **Draw** and fill in start time (T_i) and end time (T_f)
   1. T_i is 0 first and then the moment it is scheduled for the first time.
   2. T_f is the instant when it has been fully executed.
   3. RR: [process arrival -> first in list], take first process in list + execute slice (q) + is placed last in the list
2. Fill the turnaround time $T_q = T_f - T_{arrival}$

Round-Robin example (q=1)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3	0	4	4		
B	2	6	2	18	16		
C	4	4	5	17	13		
D	6	5	7	20	14		
E	8	2	10	15	7		

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

Round-Robin example (q=1)

Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	4	4	**1**	**4/3=1.33**
B	2	6	2	18	16	**10**	**16/6=2.66**
C	4	4	5	17	13	**9**	**13/4=3.25**
D	6	5	7	20	14	**9**	**14/5=2.8**
E	8	2	10	15	7	**5**	**7/2=3.5**

1. Fill waiting time: $T_e = T_q - T_s$
2. Fill the normalized turnaround time: $T_n = T_q / T_s$

Round-Robin example (q=1)

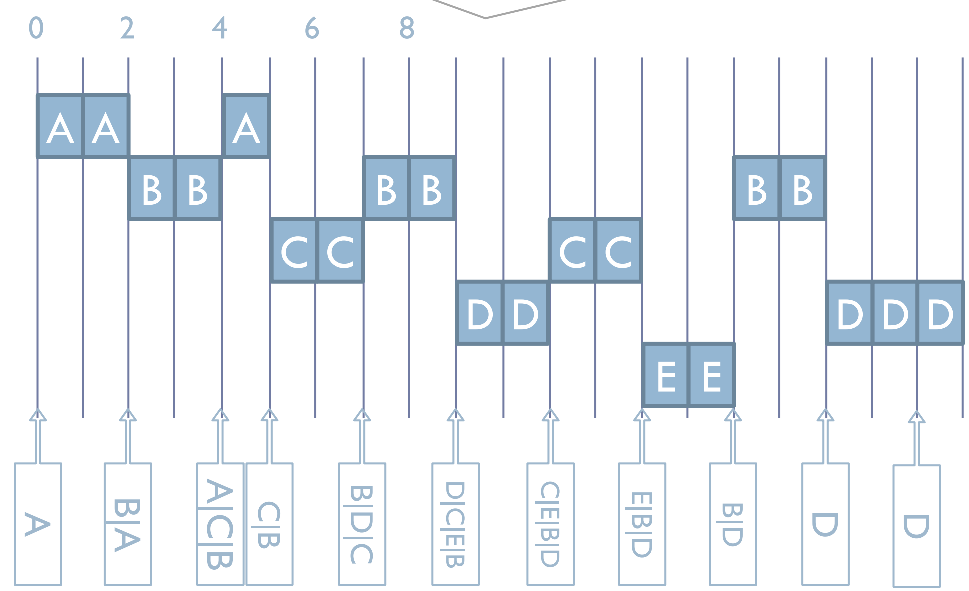
Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	4	4	1	4/3=1.33
B	2	6	2	18	16	10	16/6=2.66
C	4	4	5	17	13	9	13/4=3.25
D	6	5	7	20	14	9	14/5=2.8
E	8	2	10	15	7	5	7/2=3.5

- T_e: Average waiting time: **~~4.6~~ ~~3.6~~ 6.8**
- T_n: Normalized average turnaround time: **~~2.5~~ ~~1.84~~ 2.71**

Round-Robin example (q=2)

- While there are processes:
  - The first process ready to run is selected from the list
  - This process is executed during q=2 (slice) or end of execution
    - Those who arrive during the slice are put at the end of the ready list (and arrival time)
  - The executed process is put at the end of the ready list (if there is T_s)

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Round-Robin example (q=2)

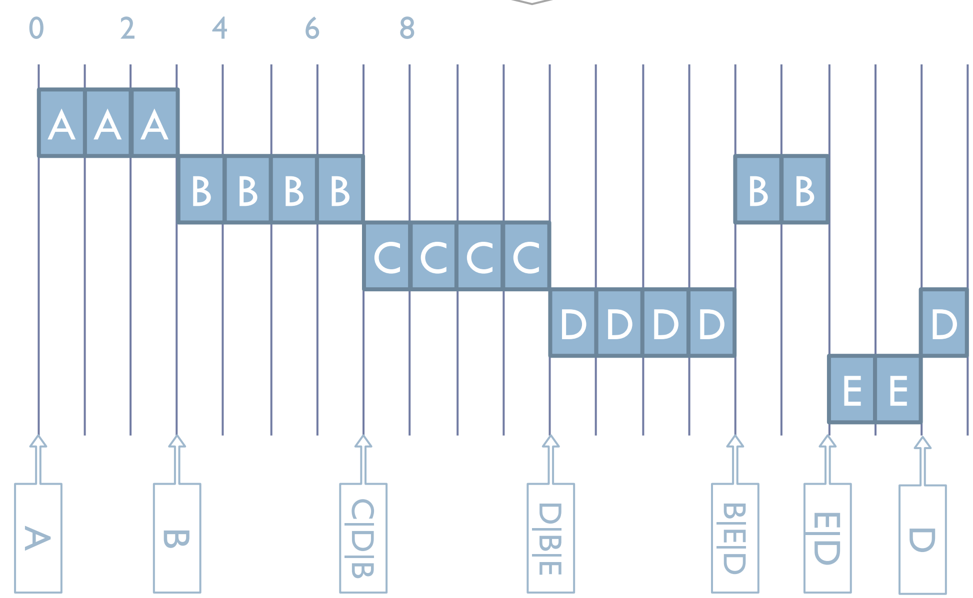
Process	Arrival	Service	Start	End	Turnaround	Waiting	Normalized turnaround
A	0	3	0	5	5	2	5/3=1.66
B	2	6	2	17	15	9	15/6=2.5
C	4	4	5	13	9	5	9/4=2.25
D	6	5	9	20	14	9	14/5=2.8
E	8	2	13	15	7	5	7/2=3.5

- T_e: Average waiting time: **~~4.6~~ ~~3.6~~ 6**
- T_n: Normalized average turnaround time: **~~2.5~~ ~~1.84~~ 2.54**

Round-Robin example (q=4)

- While there are processes:
  - The first process ready to run is selected from the list
  - This process is executed during q=4 (slice) or end of execution
    - Those who arrive during the slice are put at the end of the ready list (and arrival time)
  - The executed process is put at the end of the ready list (if there is T_s)

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Round-Robin example (q=4)

Process	Arrival	Service	Start	End	Turnaro und	Waitin g	Normalized turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	17	15	9	15/6=2.5
C	4	4	7	11	7	3	7/4=1.75
D	6	5	11	20	14	9	14/5=2.8
E	8	2	17	19	11	9	11/2=5.5

- T_e: Average waiting time: ~~**4.6**~~ ~~**3.6**~~ **6**
- T_n: Normalized average turnaround time: ~~**2.5**~~ ~~**1.84**~~ **2.71**

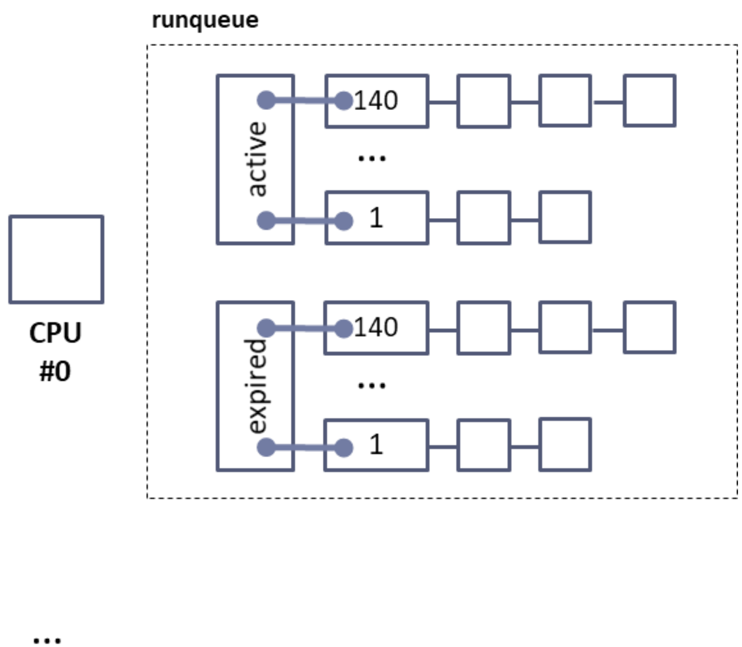
Assignment by priority

- **Initials**: Prio
- **Name**: By priorities
- **How it works**: Highest priority first
  - Each process has a priority assigned to it
  - There is a FIFO queue for each priority
  - The first process in the queue with the highest priority is selected from all the queues
- **Appropriative**: NO
- **Disadvantages**:
  - If fixed priority, then problem of starvation
    - Solution: apply aging mechanisms so that those of lower priority with more time "grow" in priority temporarily.

Contents

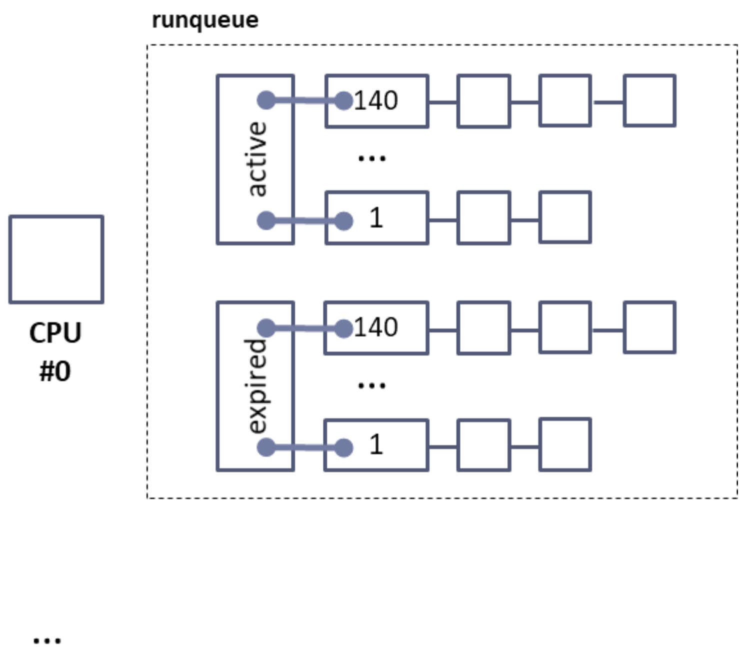
1. Basic concepts of operating system scheduling.
2. Scheduling and activation
3. Most common scheduling algorithms
   - FIFO, SJF, RR and PRIORITY.
4. **Scheduling data structures in the kernel.**
   - **Scheduling in LINUX: aging.**
5. System call for Process scheduling.

Scheduling: data structures
Linux



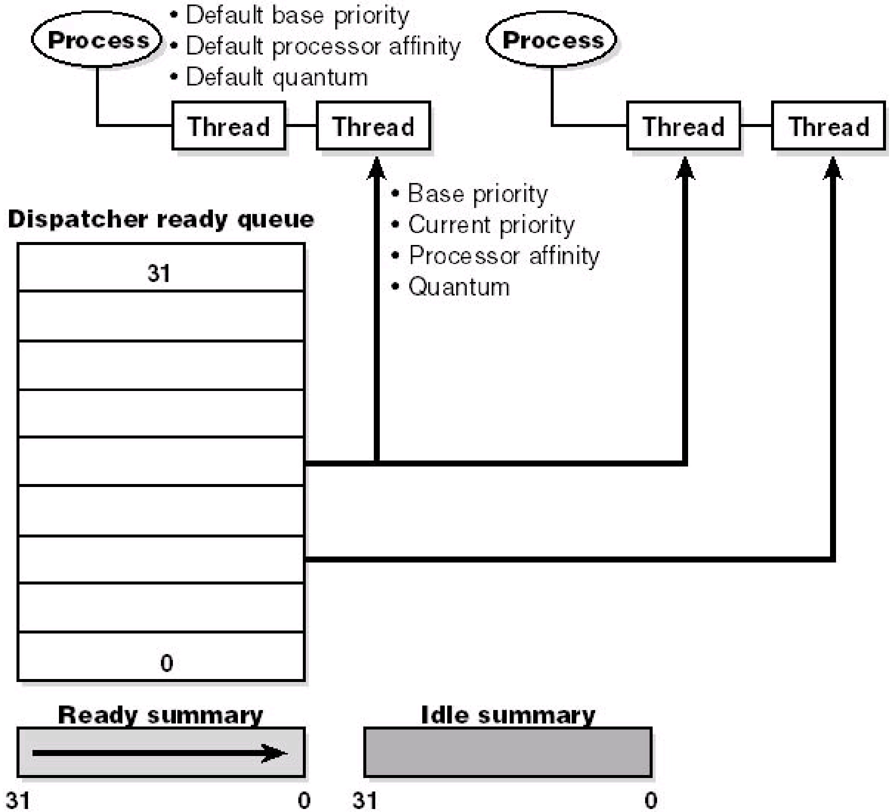
- Kernel/sched.c
- Each processor has its own *runqueue*
- Each *runqueue* has two priority vectors:
  - Active and Expired
- Each priority vector has 140 lists:
  - One per level of priority
  - 100 real-time levels included

Scheduling: management
Linux



- The scheduler chooses the processes from the list of assets according to their priority
- When the slice of a process expires, move it to the expired list
  - Priority and slice are recalculated:
    - Possibility of "aging"
- When the active list is empty, the scheduler exchanges the active and expired lists
- If a process is sufficiently interactive it will remain in the active list

Scheduling: data structures
Windows 2000

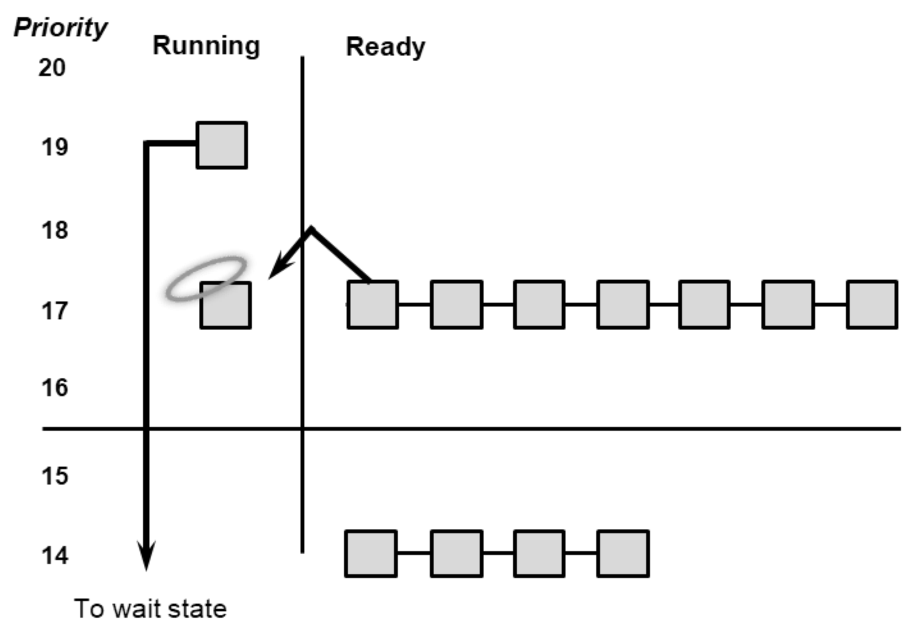


- *Dispatcher database:*
  - Database of threads ready to run and which process they belong to
- *Dispatcher ready queue*
  - One queue per priority level
- *Ready summary*
  - One bit per level
  - If $bit_i = 1 \rightarrow$ a thread at this level
  - Increases the search speed
- *Idle summary*
  - One bit per processor
  - If bit $= 1 \rightarrow$ processor is free

Inside Microsoft Windows 2000 (Microsoft Programming Series)

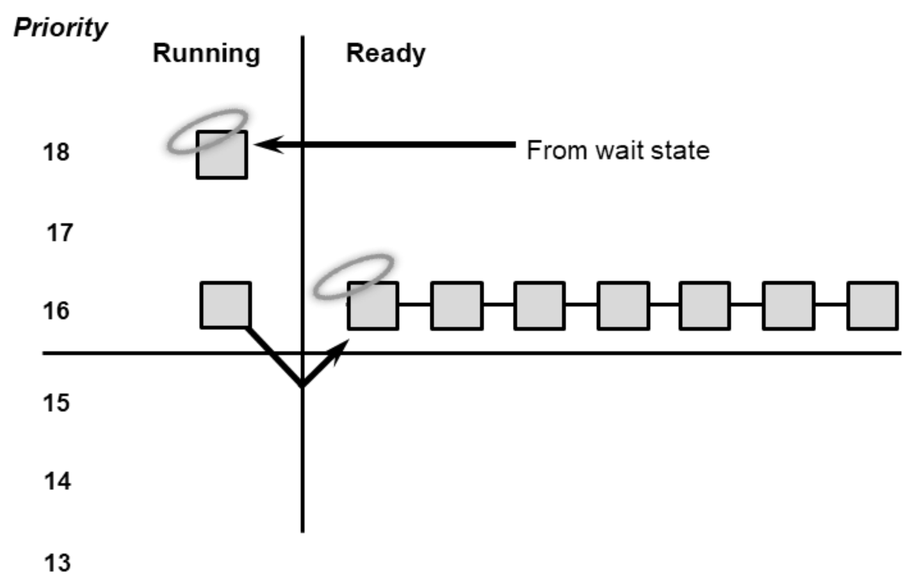
Scheduling: scenarios (1/3)
Windows 2000

- Voluntary context switching:
  - Enters the waiting state for some object:
    - event, mutex, semaphore, I/O request, etc.
  - At the end of waiting, move to the end of the ready queue + *temporary priority boost.*
  - T slice/quantum is maintained



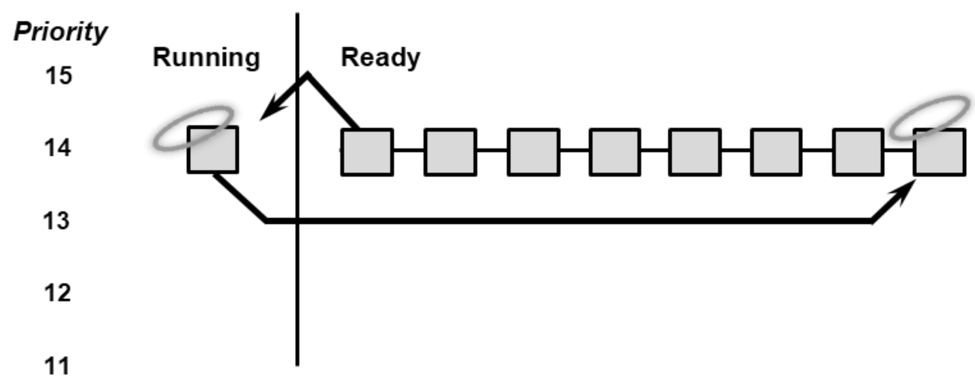
Scheduling: scenarios (2/3)
Windows 2000

- Expulsion:
  - A lower priority T thread is ejected when a higher priority T thread becomes ready to execute
  - T is at the head of its priority queue
  - T slice: if RT then is restarted otherwise, it is kept



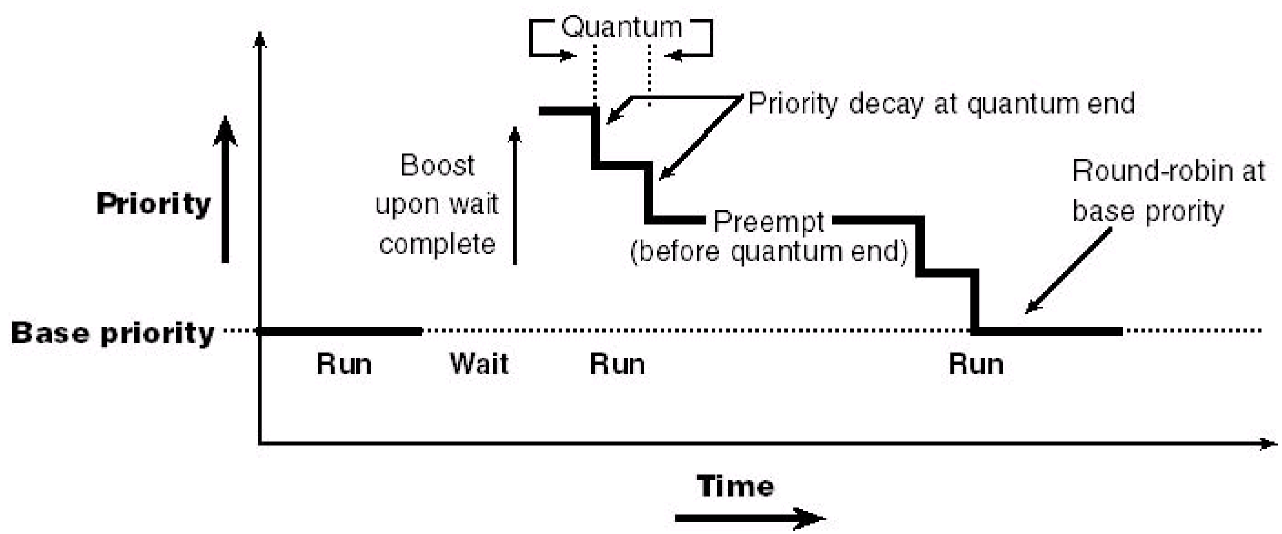
Scheduling: scenarios (3/3)
Windows 2000

- End of slice/quantum:
  - A T thread uses up its time slice (quantum)
  - Scheduler's actions:
    - Reducing the priority of T → another thread goes on to execute
    - Do not reduce the priority → T goes to the last in the queue of its level (if empty, returns again)
  - T slice/quantum: is restarted



Scheduling: *temporary priority boost*
Windows 2000

- *Priority boost:*
  - Priority is increased at certain times (0-15 levels only):
    - When an I/O operation is completed
    - When a wait operation ends
    - When the thread has been in the ready queue for a "long time" without running:
      - The *balance set manager* kernel thread increases the priority for «aging»
      - Samples 1 time per second the ready queue and if T.state=READY more than 300 ticks (~3 or 4 seconds) then
        T.priority = 15
        T.slice = 2 * usual_slice



Inside Microsoft Windows 2000 (Microsoft Programming Series)

Windows Scheduler

summary

- Main characteristics:
  - Priority-based and use of time quantums.
  - Appropriate scheduling.
  - Scheduling with processor affinity.
- Scheduling by threads and not by processes.
  - A thread may lose the processor if a higher-priority thread is ready.
- Scheduling decisions:
  - New threads → Ready.
  - Blocked threads receiving event → Ready.
  - Thread leaves CPU if its quantum ends, terminates or blocks.

Contents

1. Basic concepts of operating system scheduling.
2. Scheduling and activation
3. Most common scheduling algorithms
   - FIFO, SJF, RR and PRIORITY.
4. Scheduling data structures in the kernel.
   - Scheduling in LINUX: aging.
5. **System call for Process scheduling.**

System call for Process scheduling

- `sched_setscheduler/sched_getscheduler`
  - Set/return the scheduling policy and parameters of a specified thread.
- `sched_setparam/sched_getparam`
  - Set/Fetch the scheduling parameters of a specified thread.
- `sched_get_priority_max/sched_get_priority_min`
  - Return the maximum/minimum priority available in a specified scheduling policy.
- `sched_rr_get_interval`
  - Fetch the quantum used for threads that are scheduled under the "round-robin"
- `sched_yield`
  - Cause the caller to relinquish the CPU, so that some other thread be executed.
- `sched_setaffinity/sched_getaffinity`
  - (Linux-specific) Set/Get the CPU affinity of a specified thread.
- `sched_setattr/sched_getattr`
  - Set/Fetch the scheduling policy and parameters of a specified thread. This (Linux-specific) system call provides a superset of the functionality of sched_set/getscheduler and sched_set/getparam.

ARCOS Group
Universidad Carlos III de Madrid

Lesson 3
Process scheduling

Operating Systems
Computer Science and Engineering