

# Operating Systems

Christopher Kruegel  
Department of Computer Science  
UC Santa Barbara  
<http://www.cs.ucsb.edu/~chris/>  
Fall 2009

---

# Scheduling

UC Santa Barbara

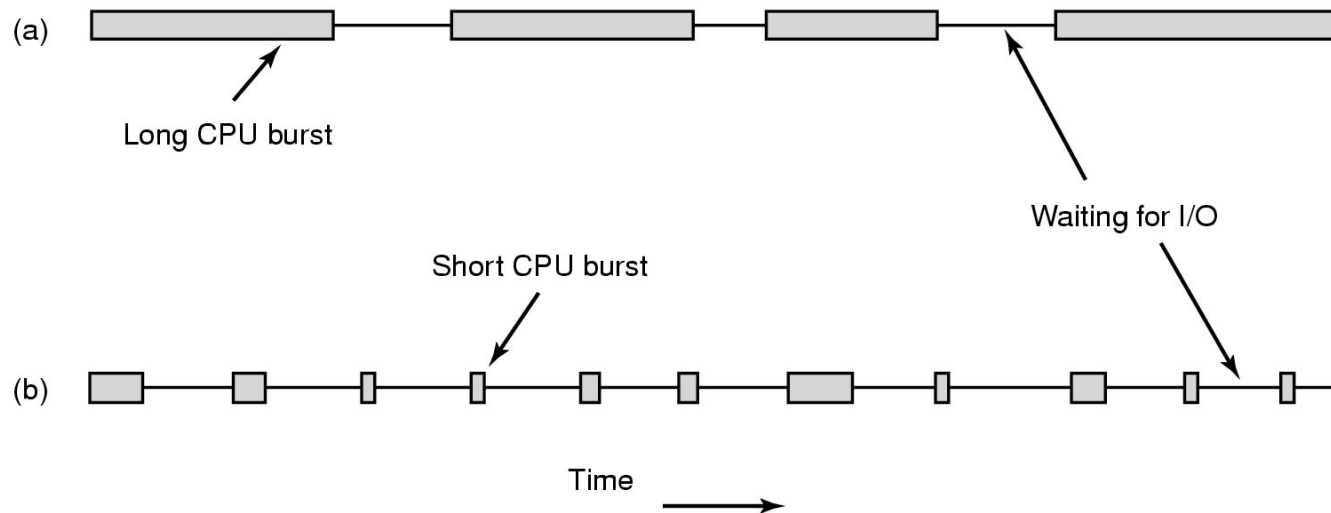
---

- Many processes to execute, but one CPU
- OS time-multiplexes the CPU by operating context switching
  - Between user processes
  - Between user processes and the operating system
- Operation carried out by *scheduler* following a *scheduling algorithm*
- Switching is expensive
  - Switch from user to kernel model
  - Save the state of the current process (including memory map)
  - Select a process for execution (scheduler)
  - Restore the saved state of the new process

# CPU-bound and I/O-bound Processes

UC Santa Barbara

- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process (a)
  - an I/O bound process (b)



# When To Schedule

---

*UC Santa Barbara*

- Must schedule
  - a process blocks (I/O, semaphore, etc)
  - a process exits
- May schedule
  - new process is created (parent and child are both ready)
  - I/O interrupt
  - clock interrupt

# Scheduling Algorithms

UC Santa Barbara

- Non-preemptive
  - CPU is switched when process
    - has finished
    - executes a yield()
    - blocks
- Preemptive
  - CPU is switched independently of the process behavior
    - A clock interrupt is required
- Scheduling algorithms should enforce
  - Fairness
  - Policy
  - Balance

# Scheduling in Batch Systems

UC Santa Barbara

---

- Goals
  - Throughput:  
maximize jobs per hour
  - Turnaround time:  
minimize time between submission and termination
  - CPU utilization  
keep processor busy
- Examples
  - First-come first-served
  - Shortest job first
  - Shortest remaining time next

# First-Come First-Served

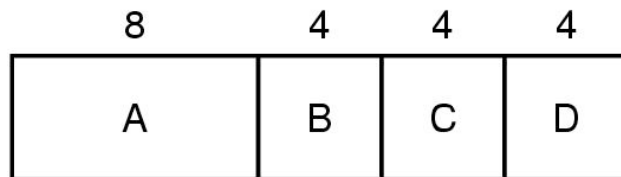
UC Santa Barbara

---

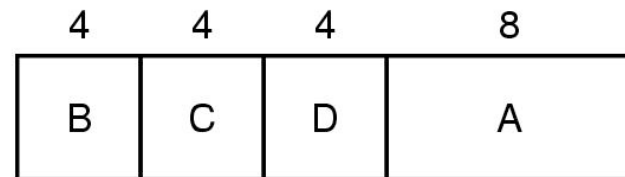
- Processes are inserted in a queue
- The scheduler picks up the first process, executes it to termination or until it blocks, and then picks the next one
- Very simple
- Disadvantage
  - I/O-bound processes could be slowed down by CPU-bound ones

# Shortest Job First

- This algorithm assumes that running time for all the processes to be run is known in advance
- Scheduler picks the shortest job first
- Optimizes turnaround time
  - a) Turnaround is A=8, B=12, C=16, D=20 (avg. 14)
  - b) Turnaround is B=4, C=8, D=12, A=20 (avg. 11)
- Problem: what if new jobs arrive?



(a)



(b)

# Shortest Remaining Time Next

UC Santa Barbara

- This algorithm also assumes that running time for all the processes to be run is known in advance
- The algorithm chooses the process whose remaining run time is the shortest
- When a new job arrive, its remaining run time is compared to the one of the process running
- If current process has more remaining time than the run time of new process, the current process is preempted and the new one is run

# Scheduling in Interactive Systems

UC Santa Barbara

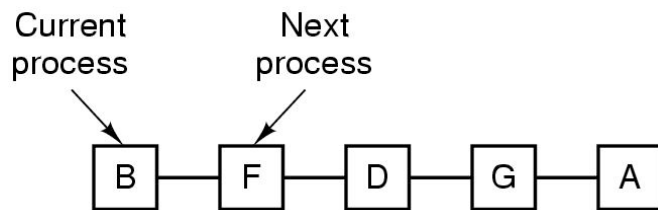
---

- Goals
  - Response time:  
minimize time needed to react to requests
  - Proportionality:  
meet user expectations
- Examples
  - Round robin
  - Priority scheduling
  - Lottery scheduling

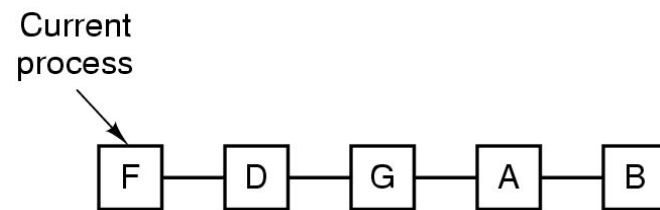
# Round Robin Scheduling

UC Santa Barbara

- Each process is assigned a *quantum*
- The process
  - Suspends before the end of the quantum or
  - Is preempted at the end of the quantum
- Scheduler maintains a list of ready processes



(a)



(b)

# Round Robin Scheduling

UC Santa Barbara

---

- Parameters:
  - Context switch (e.g., 1 msec)
  - Quantum length (e.g., 25 msec)
- If quantum is too small, a notable percentage of the CPU time is spent in switching contexts
- If quantum is too big, response time can be very bad

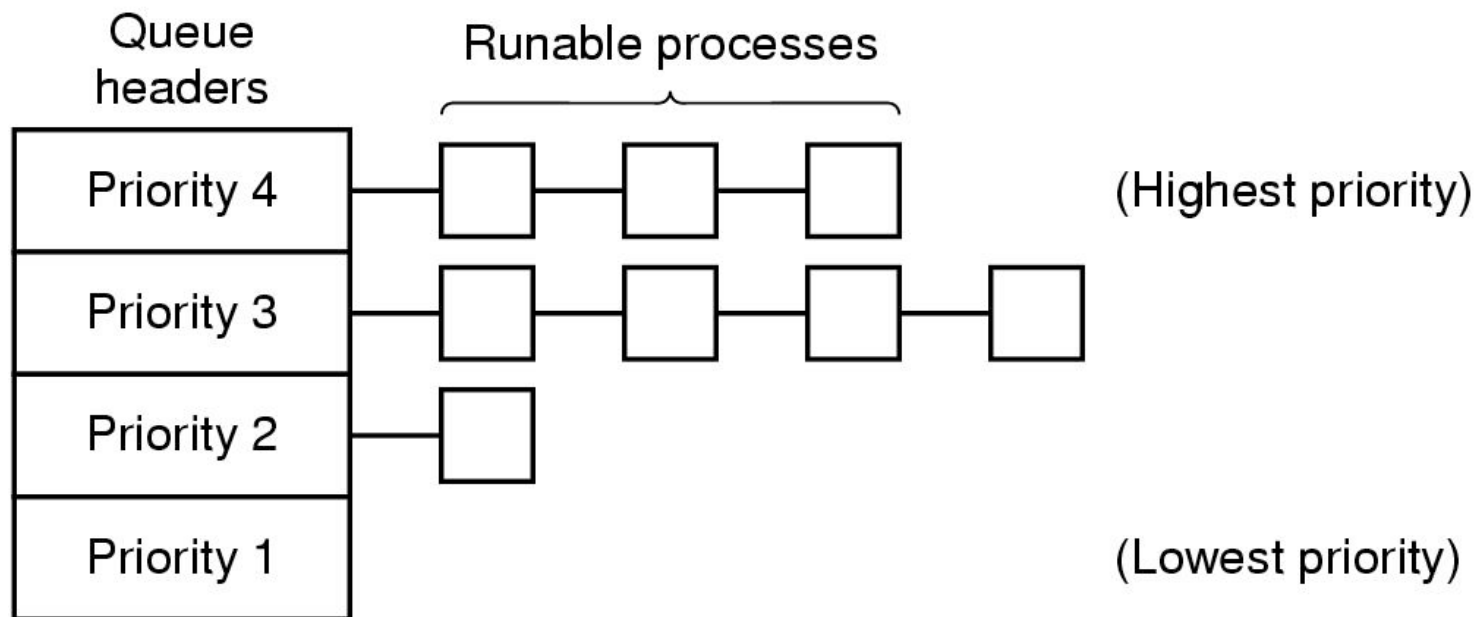
# Priority Scheduling

UC Santa Barbara

- Each process is assigned a priority
- The process with the highest priority is allowed to run
- I/O bound processes should be given higher priorities
- Problem: low priority processes may end up starving...
- First solution: As the process uses CPU, the corresponding priority is decreased
- Second solution: Set priority as the inverse of the fraction of quantum used
- Third solution: Used priority classes (starvation is still possible)

# Priority Scheduling

UC Santa Barbara



# Lottery Scheduling

UC Santa Barbara

---

- OS gives “lottery tickets” to processes
- Scheduler picks a ticket randomly and gives CPU to the winner
- Higher-priority processes get more tickets
- Advantage:
  - processes may exchange tickets
  - it is possible to fine tune the share of CPU that a process receives
  - easy to implement

# Scheduling

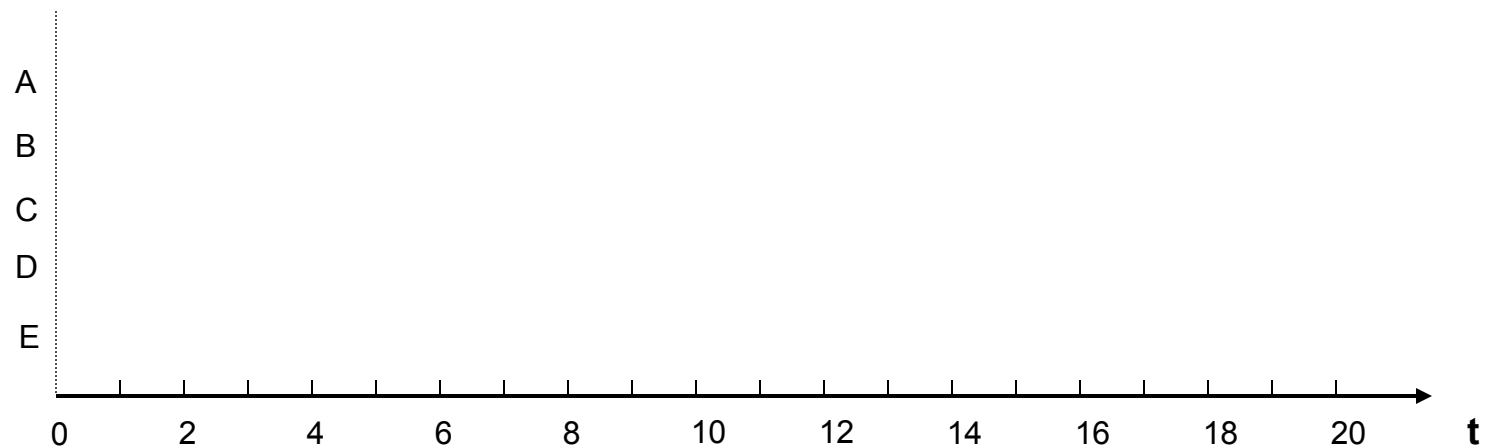
- Example
  - non-preemptive priority scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3

# Scheduling

UC Santa Barbara

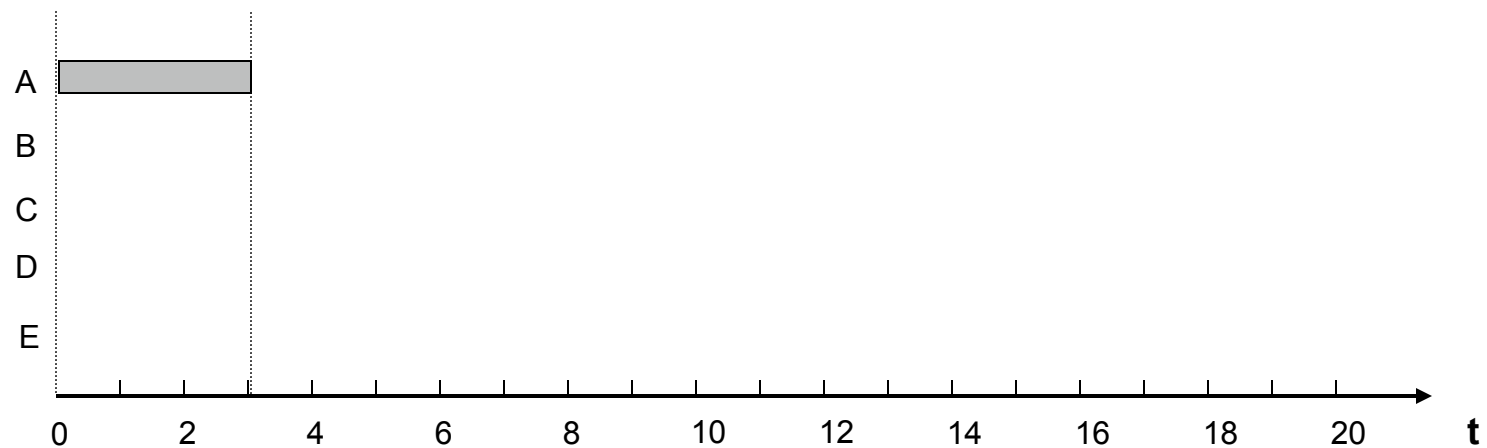
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

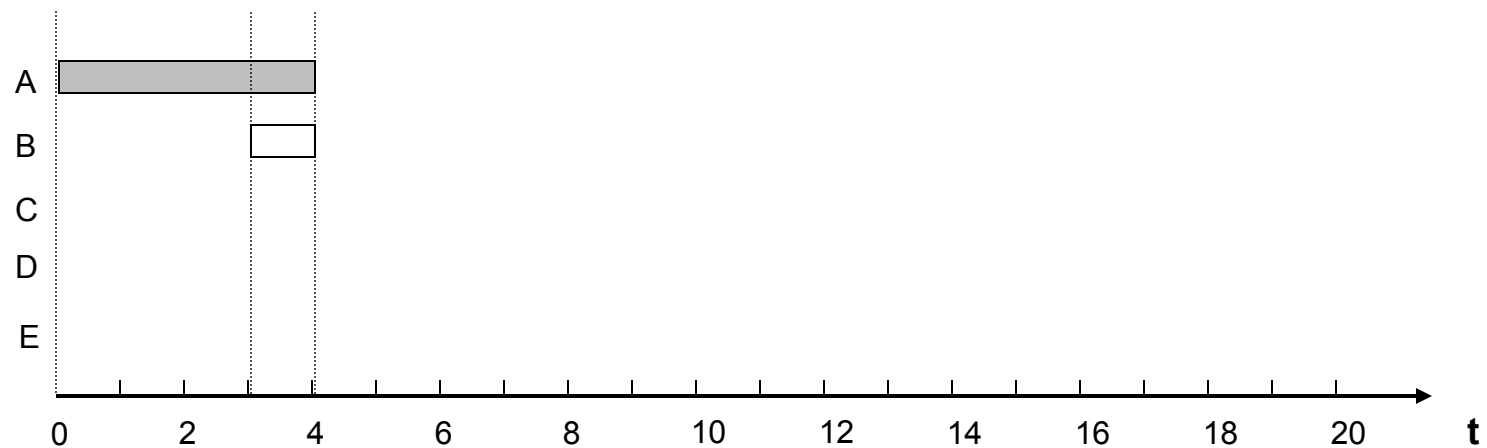
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

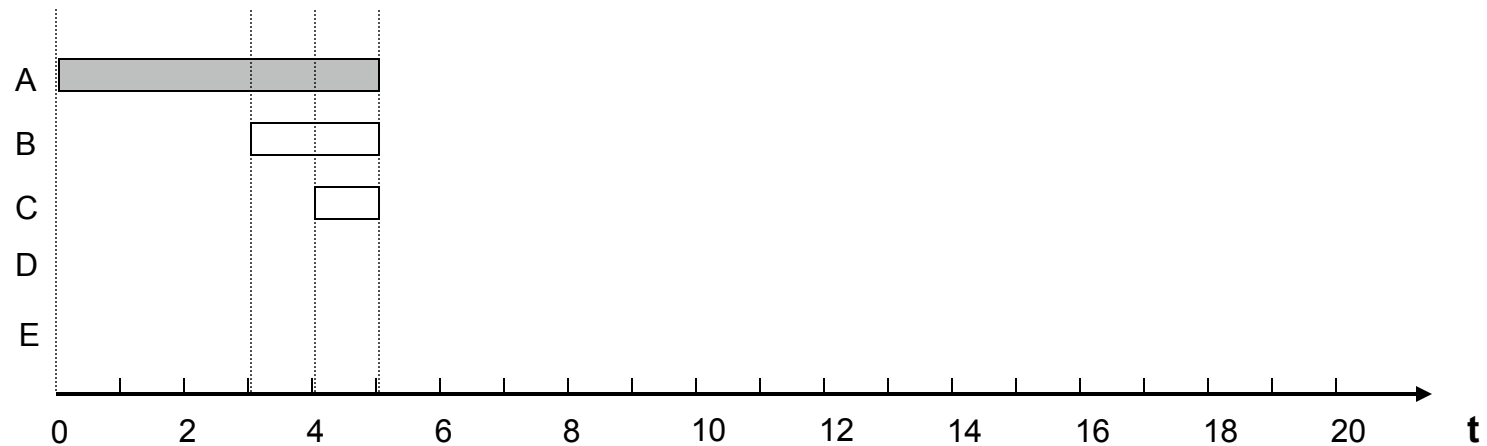
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

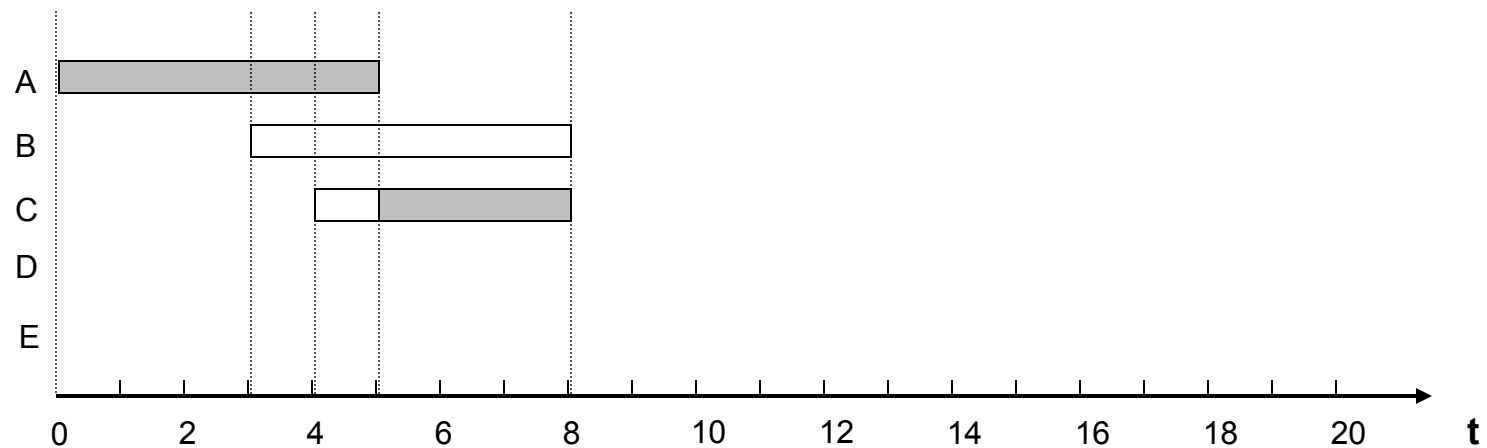
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

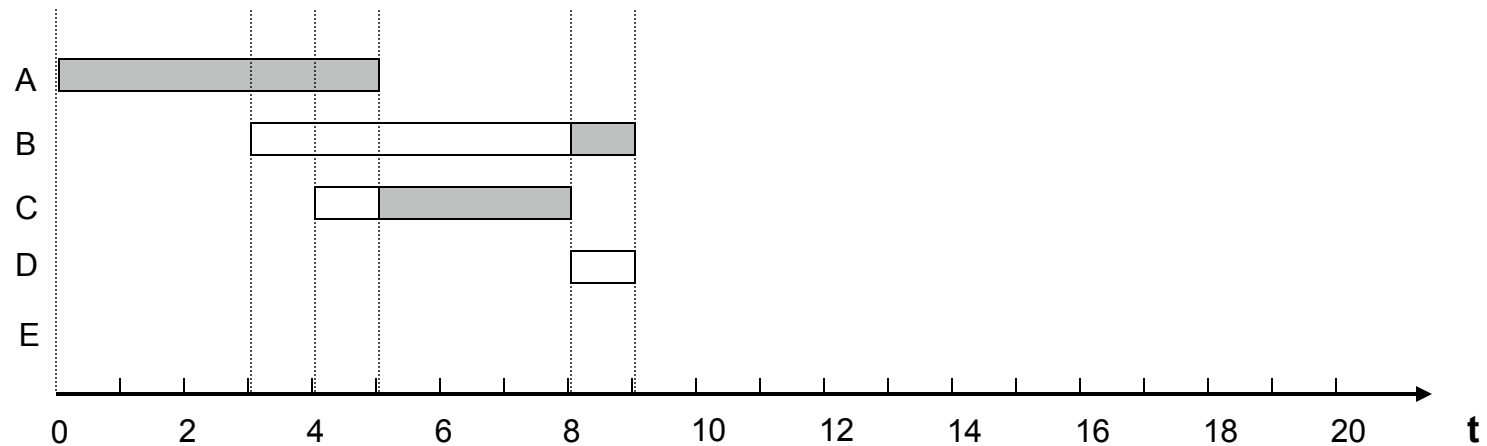
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

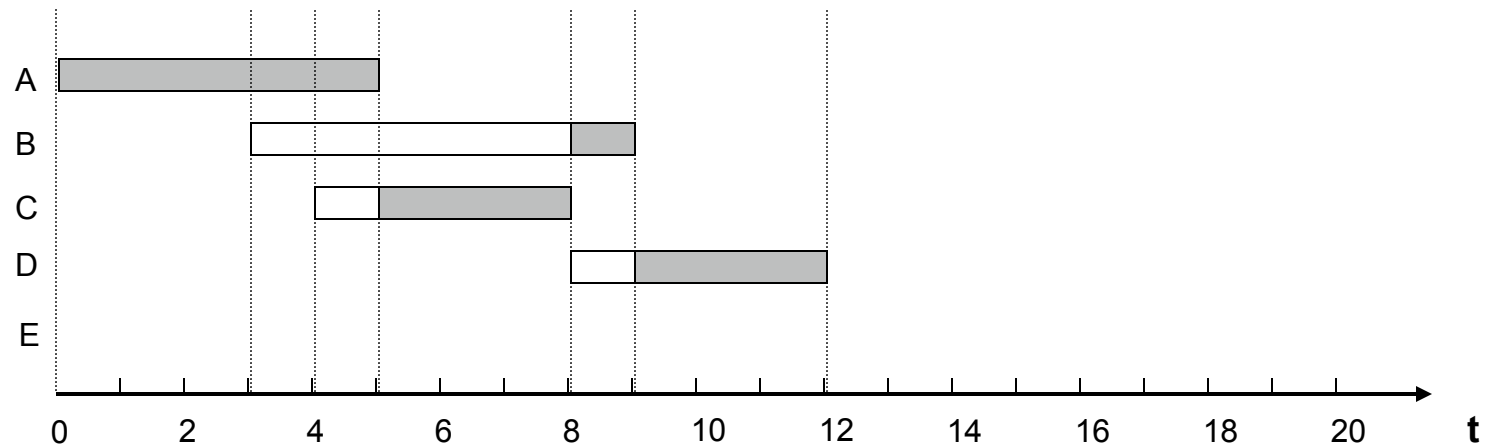
UC Santa Barbara

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

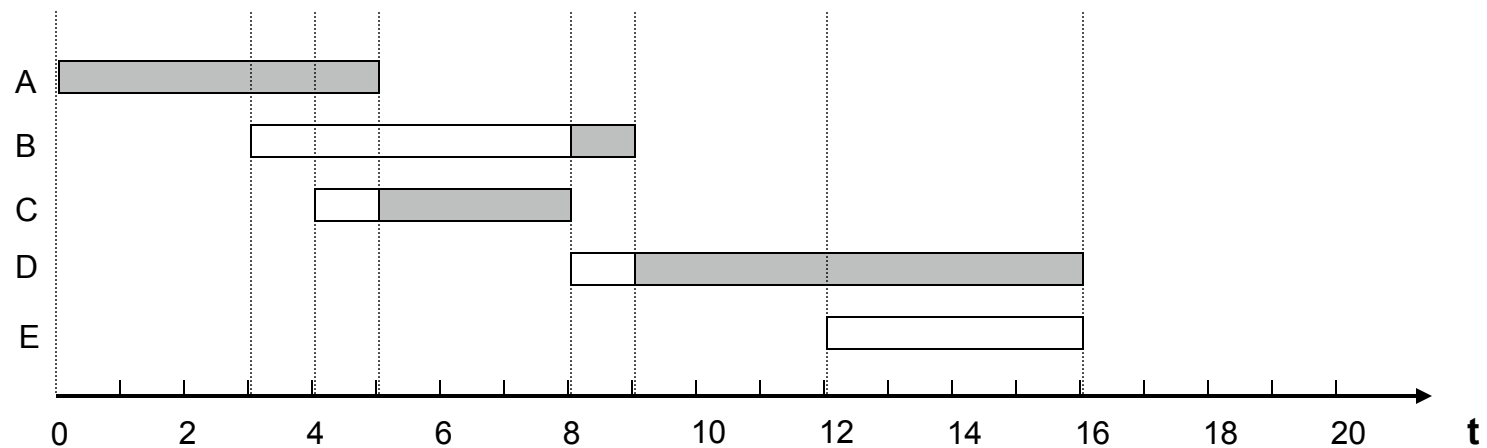
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

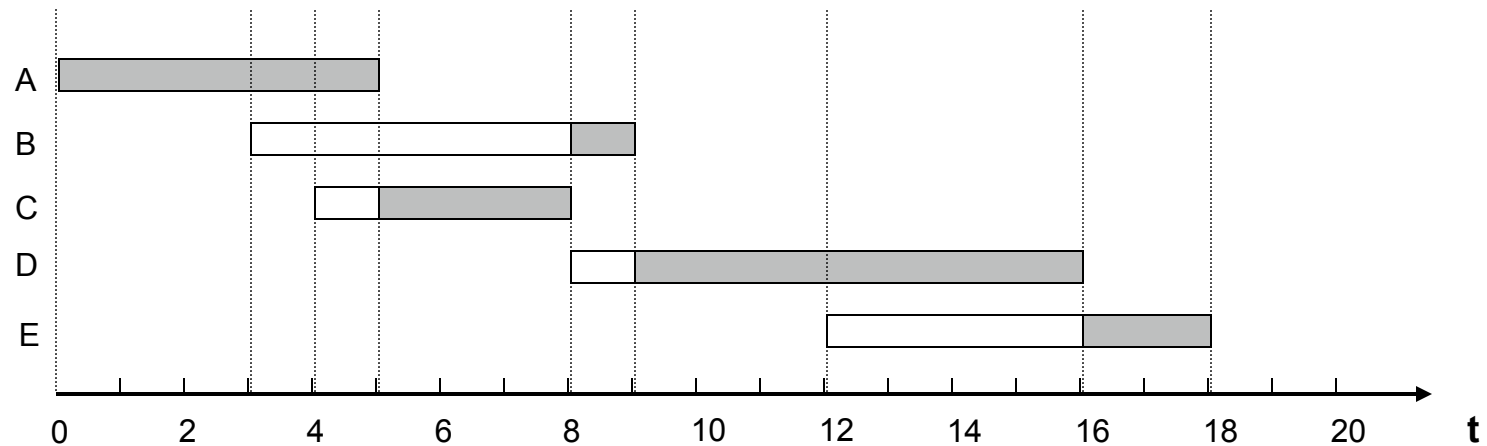
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

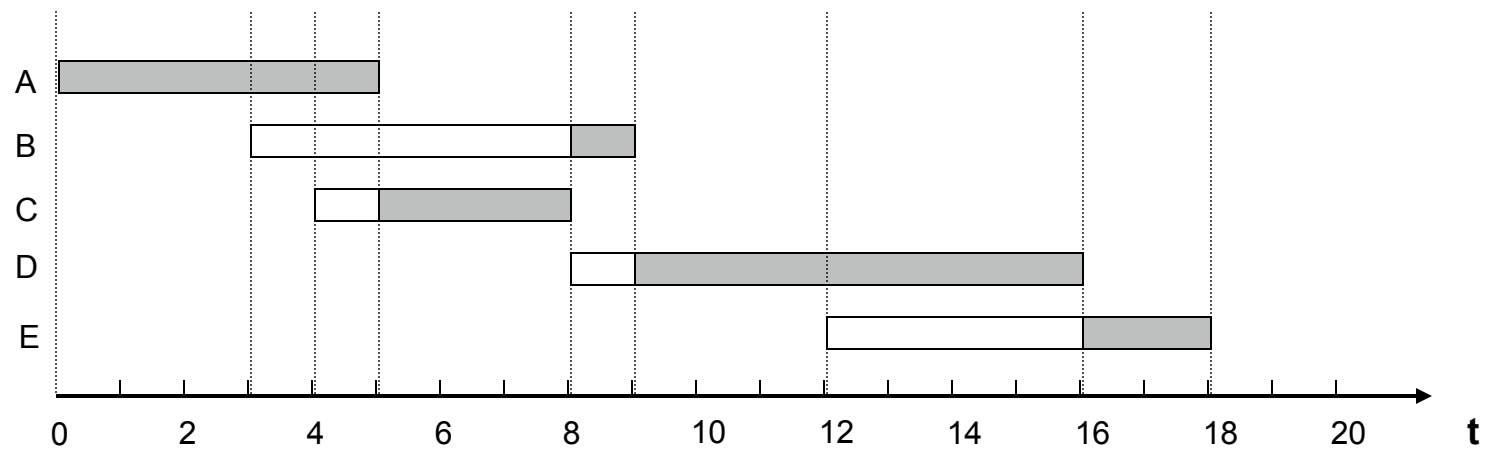
UC Santa Barbara

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara



Process	A	C	B	D	E
Time (RUNNING)	0	5	8	9	16

# Scheduling

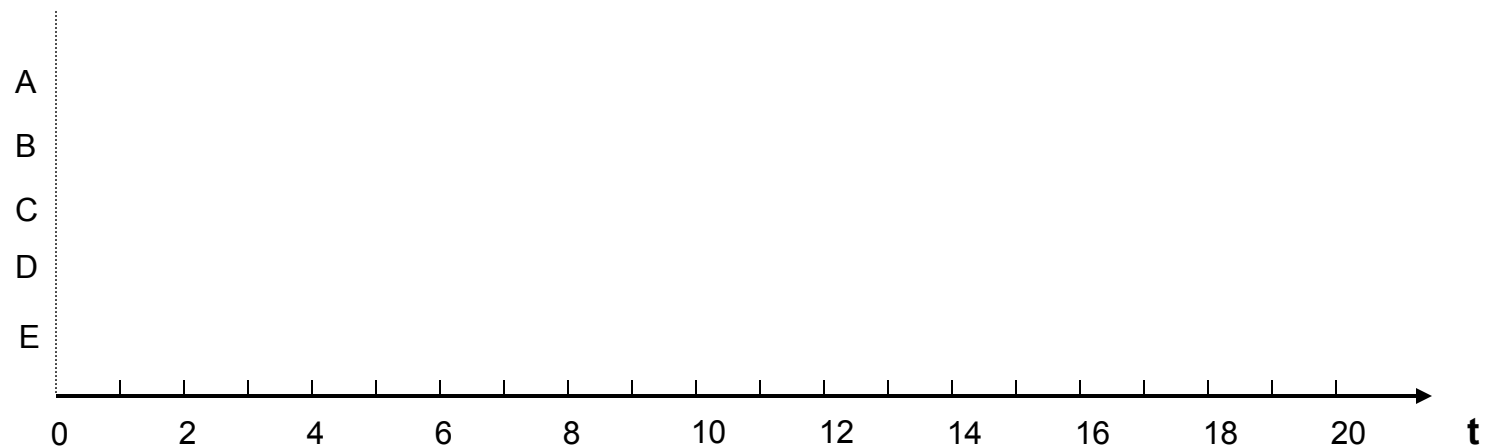
- Example
  - *preemptive* priority scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3

# Scheduling

UC Santa Barbara

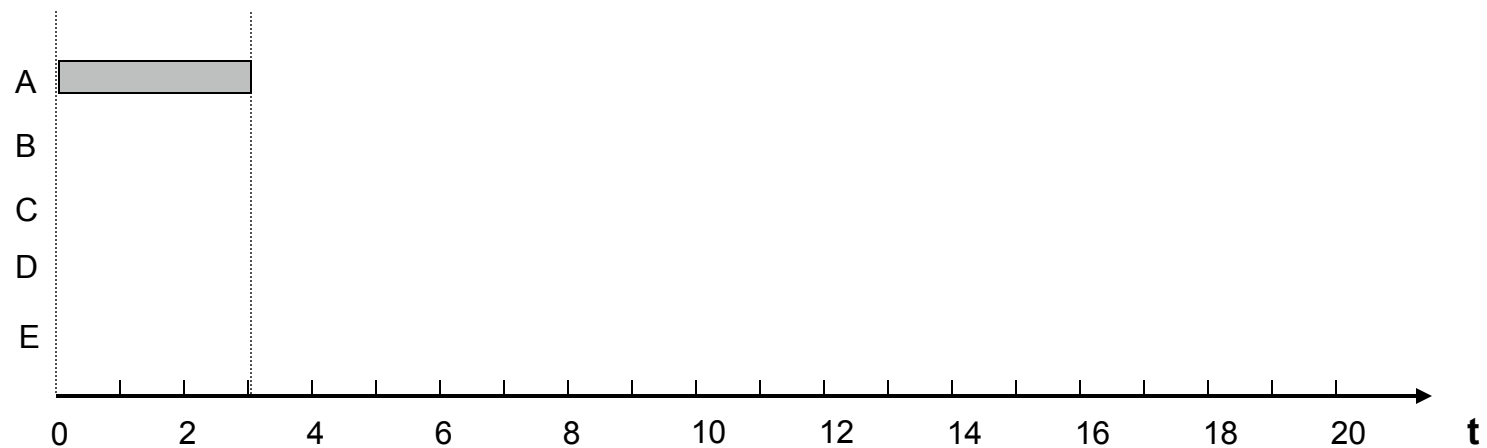
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

UC Santa Barbara

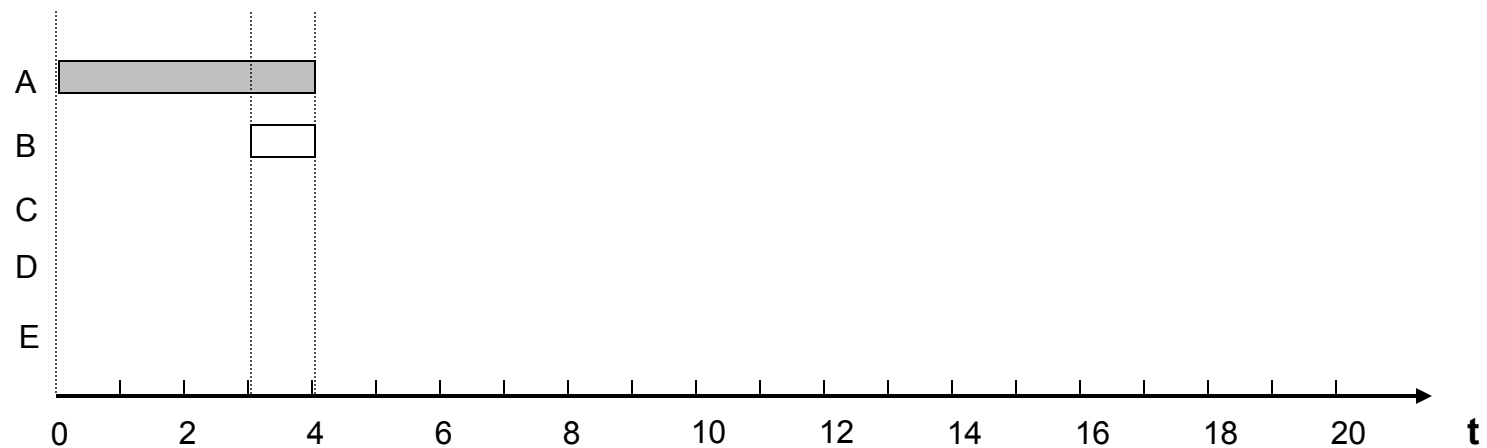
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

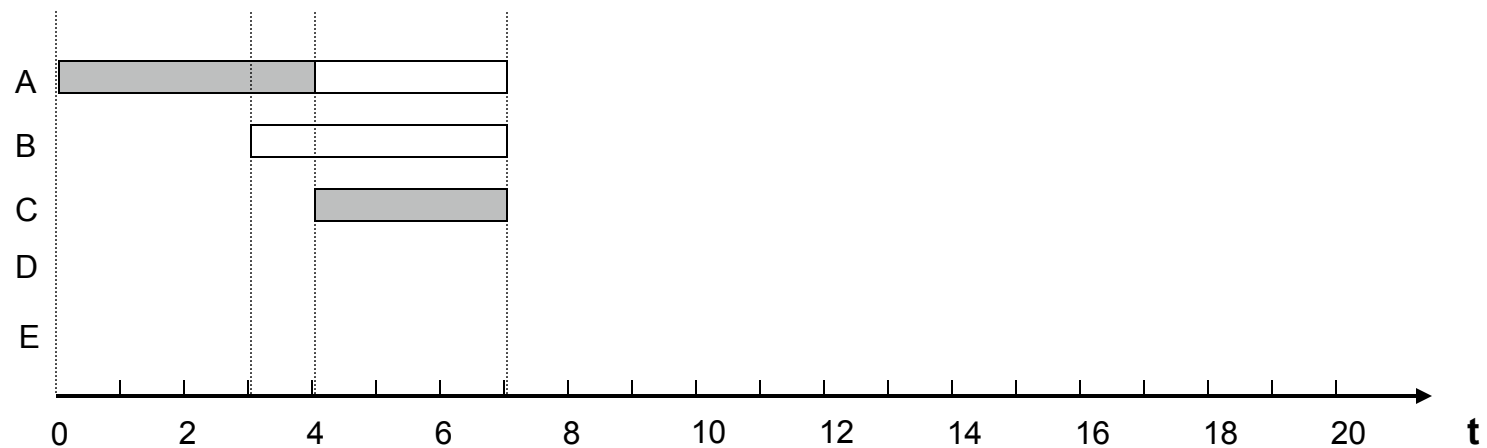
UC Santa Barbara

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

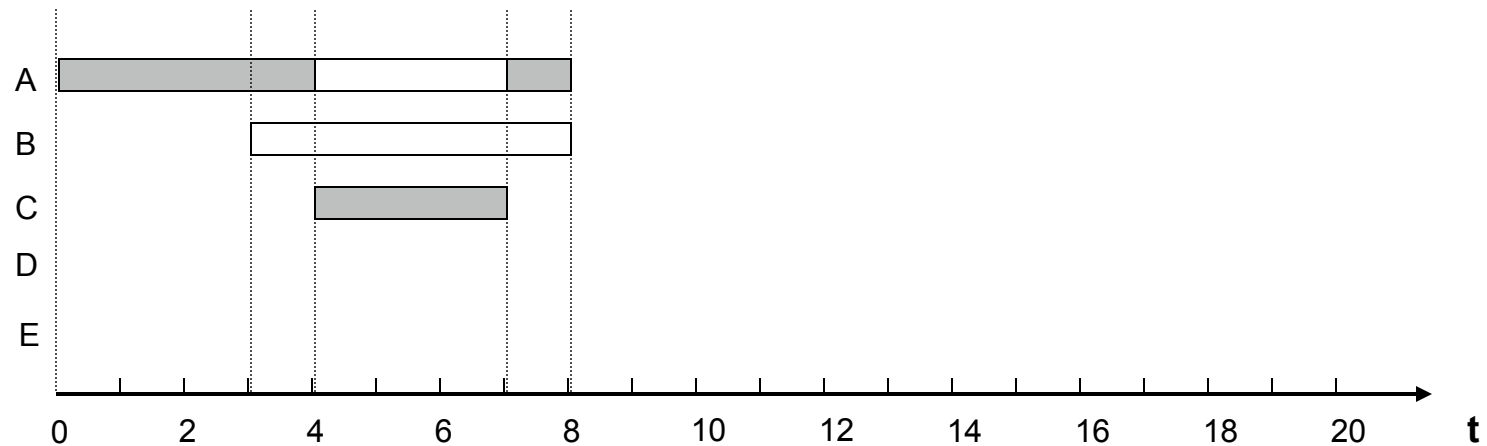
Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

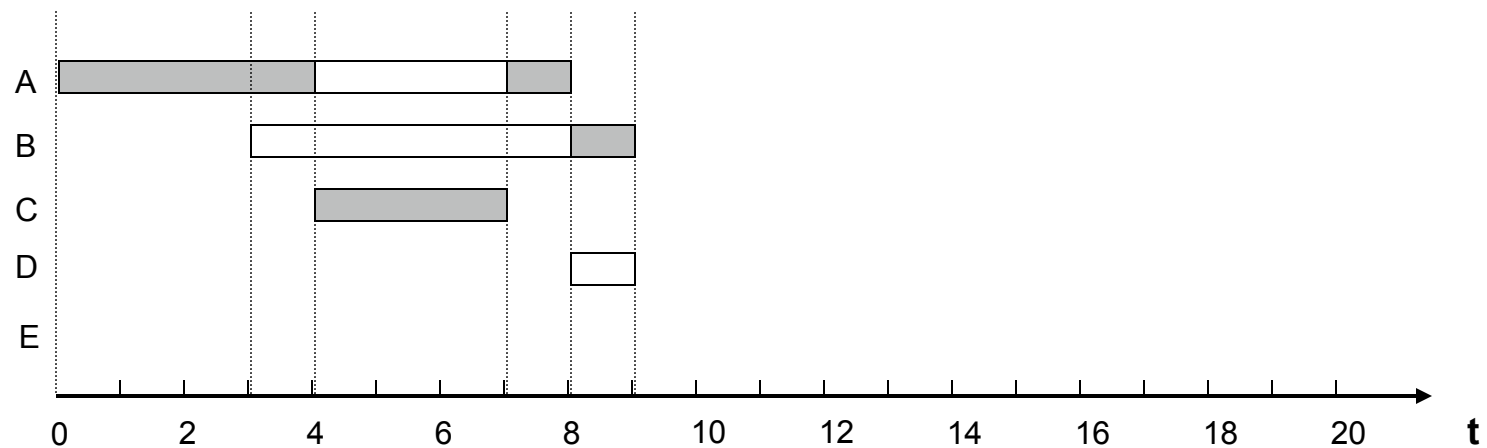
UC Santa Barbara

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



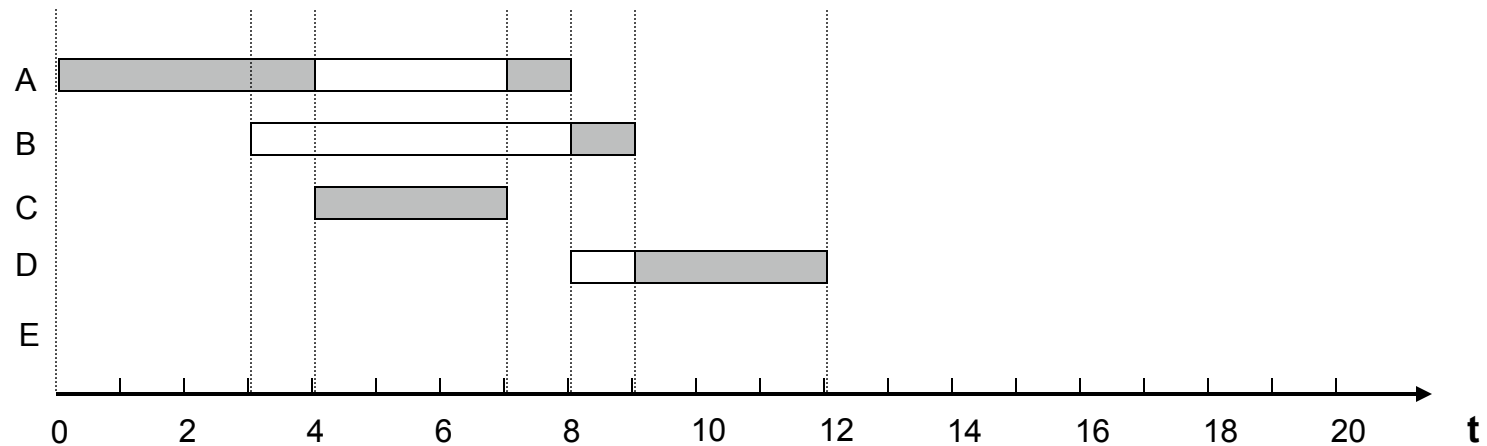
# Scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



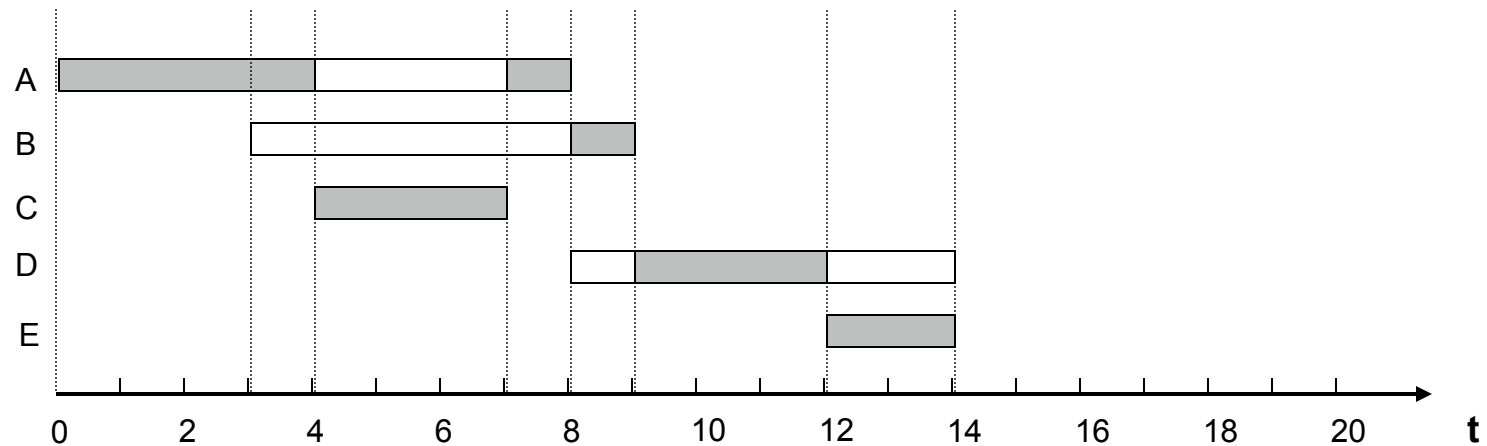
# Scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



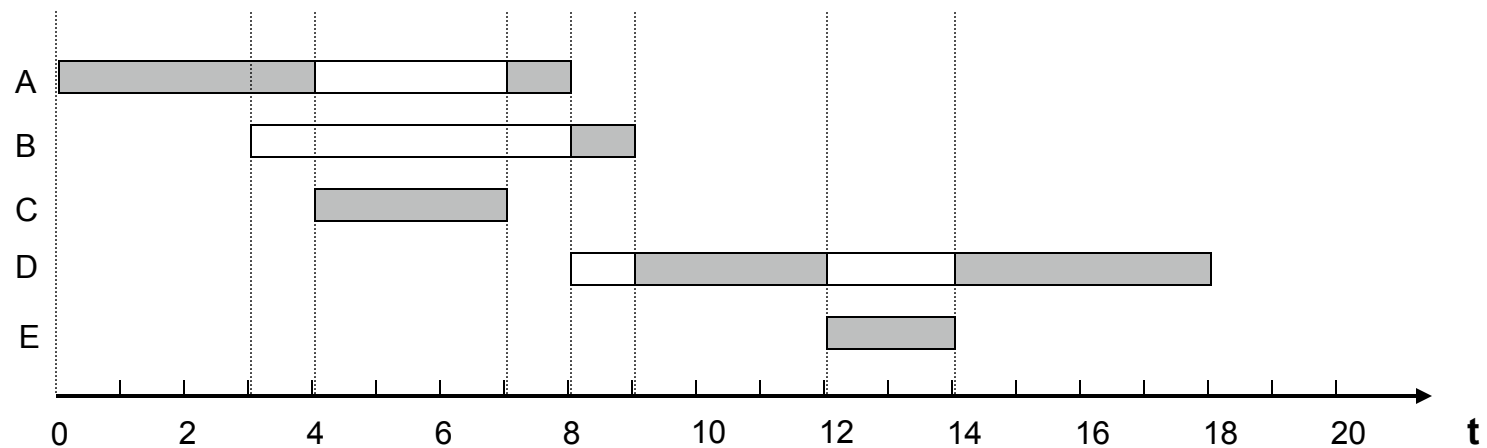
# Scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



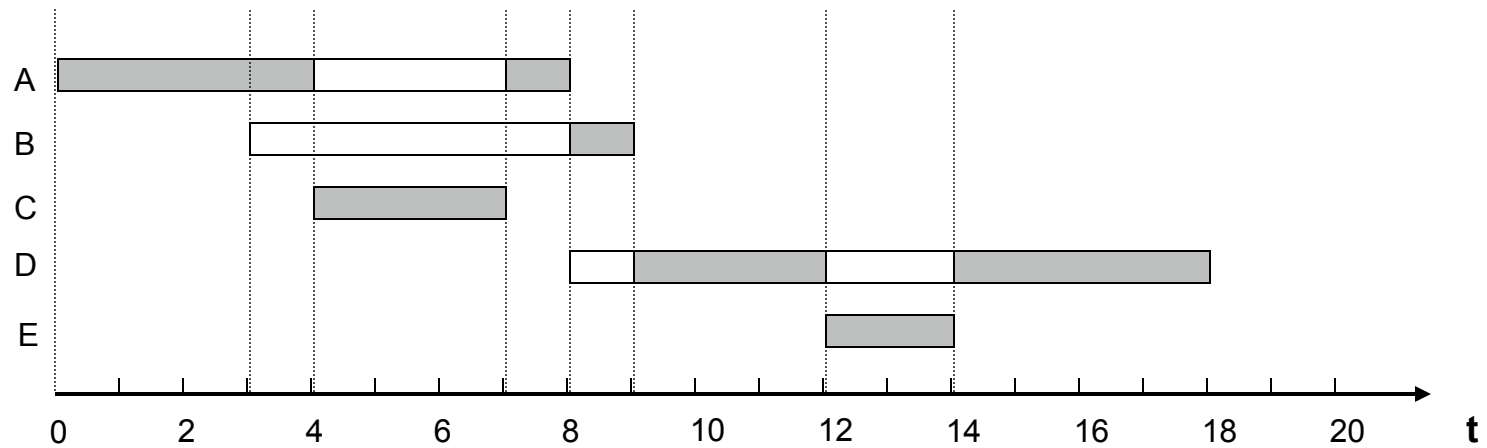
# Scheduling

Process	Start	Runtime	Priority
A	0	5	2
B	3	1	1
C	4	3	4
D	8	7	0
E	12	2	3



# Scheduling

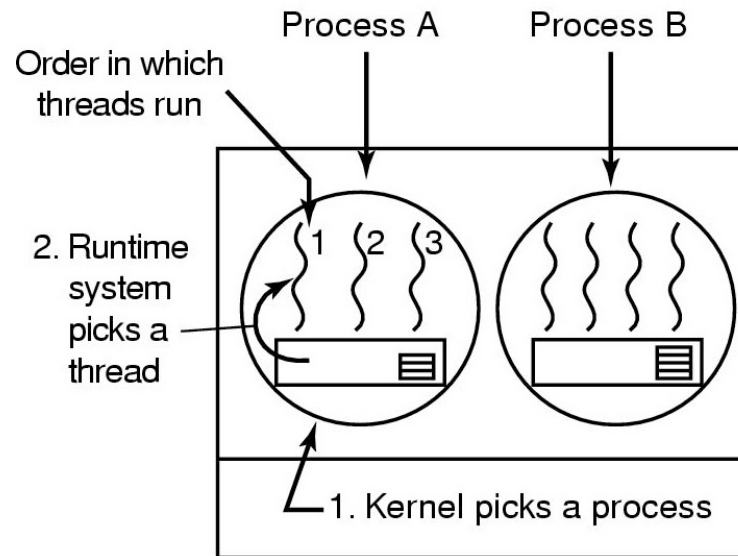
UC Santa Barbara



Process	A	C	A	B	D	E	D
Time (RUNNING)	0	4	7	8	9	12	14

# Thread Scheduling

- If threads are implemented in user space, only one process' threads are run inside a quantum
- Possible scheduling of user-level threads
  - 48-msec process quantum
  - Threads run 8 msec/CPU burst

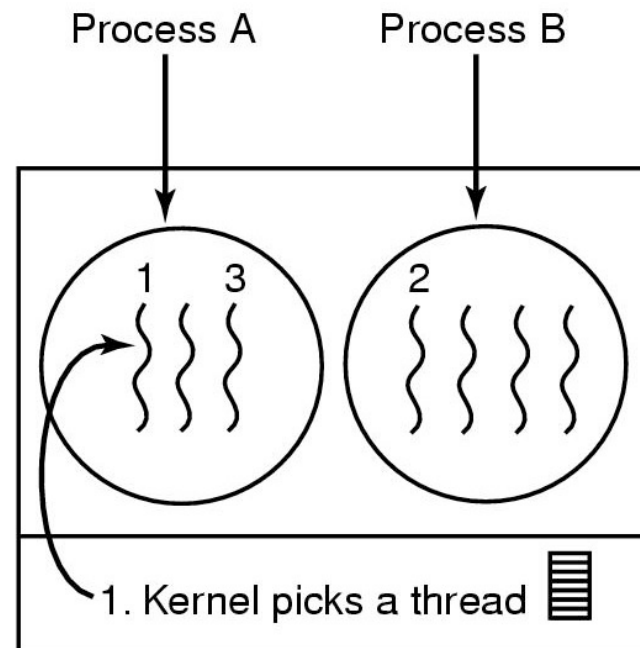


Possible: A1, A2, A3, A1, A2, A3  
Not possible: A1, B1, A2, B2, A3, B3

# Thread Scheduling

UC Santa Barbara

- If threads are implemented in the kernel, threads can be interleaved
- Kernel may decide to switch to a thread belonging to the same process for efficiency reasons (memory map does not change)



Possible: A1, A2, A3, A1, A2, A3

Also possible: A1, B1, A2, B2, A3, B3

# Policy versus Mechanism

UC Santa Barbara

---

- Sometimes an application may want to influence the scheduling of cooperating processes (same user, or children processes) to achieve better overall performance
- Separate what is allowed to be done with how it is done
  - process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - Mechanism in the kernel
- Parameters filled in by user processes
  - Policy set by user process

# Linux - CFS

UC Santa Barbara

- Completely fair scheduler (CFS)

Ingo Molnar:

80% of CFS's design can be summed up in a single sentence: CFS basically models an "ideal, precise multi-tasking CPU" on real hardware.

On real hardware, we can run only a single task at once, so while that one task runs, the other tasks that are waiting for the CPU are at a disadvantage - the current task gets an unfair amount of CPU time. In CFS this fairness imbalance is expressed and tracked via the per-task `p->wait_runtime` (nanosec-unit) value. "wait\_runtime" is the amount of time the task should now run on the CPU for it to become completely fair and balanced.