Chapter 5: Scheduling

- Definitions for CPU scheduling
- Picking criteria used to schedule processes
- Choosing an algorithm to schedule processes
- Advanced kinds of CPU scheduling
  - » Multiprocessor CPU scheduling
  - » Real-time scheduling
- Evaluating scheduling algorithms

Definitions Used in CPU Scheduling

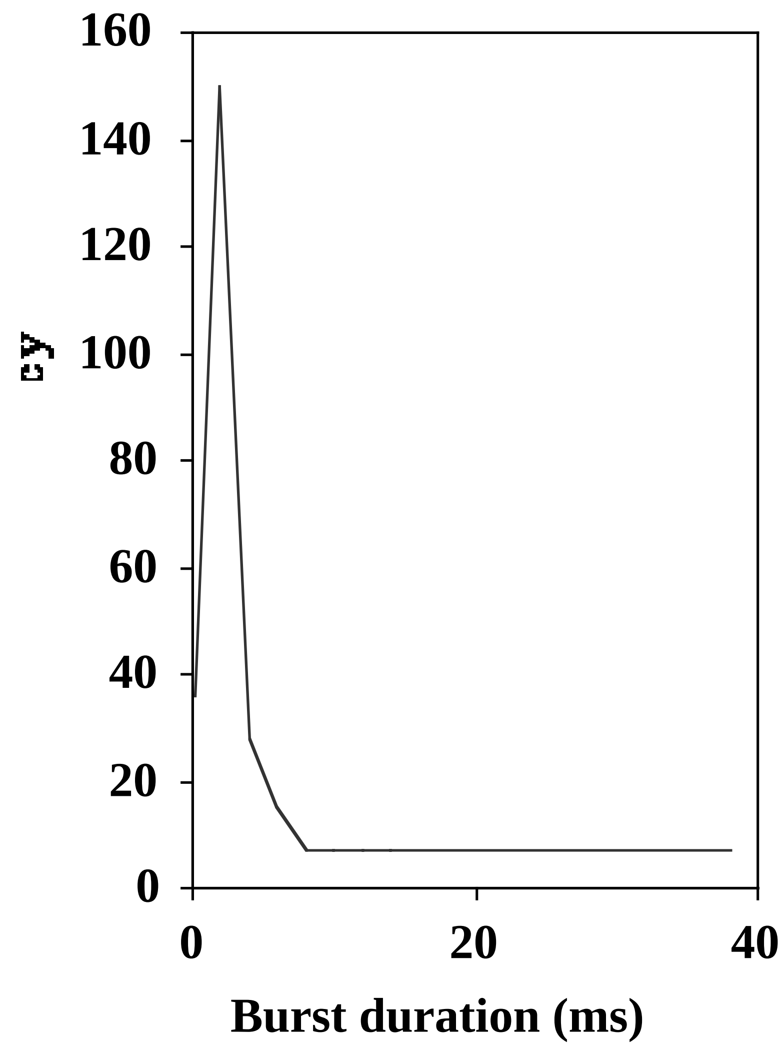
- Multiprogramming: the ability to switch between multiple programs (processes) on a single CPU
- CPU burst
  - » Period of CPU time used by a process between I/O requests
  - » Duration varies depending on code being run
- CPU scheduler
  - » Selects the process to run next using one of several methods (scheduling algorithms) to select the process
  - » Tells the dispatcher to run the process
- Dispatcher
  - » Switches the process context
  - » Switches to user mode (if necessary)
  - » Returns to where the CPU left off

Types of Scheduling

- CPU scheduler makes decisions when any process
  - » Switches from running to waiting state
  - » Terminates
  - » Switches from running to ready state
  - » Switches from waiting to ready state
- First two mechanisms are non-preemptive
  - » Process must voluntarily give up the CPU
  - » No unexpected switch from running to ready
- Second two mechanisms are preemptive
  - » Process can be forced to give up the CPU
  - » Each ready process gets a fraction of the CPU

CPU Burst Duration

- Multiprogramming provides best response time
  - » Utilization is good
  - » Short processes get good response time
- Processes execute in bursts
  - » Alternate CPU usage and I/O usage (process is in wait state)
  - » CPU bursts tend to be short (most under 20 ms)



Criteria for Scheduling

- Overall goals:
  - » Keep the CPU as busy as possible
  - » Provide good response time for users
- Statistics of interest
  - » Throughput: number of processes that finish per unit time
  - » Turnaround time: total (wall clock) time to run a particular process
  - » Waiting time: time a process has been waiting in the ready queue
    - – Doesn’t include time waiting for I/O
  - » Response time: time it takes from when a request was submitted until the first response is produced
    - – Not the time until process finishes!
    - – Often, response is necessary if process is long-running

What Makes a Good CPU Schedule?

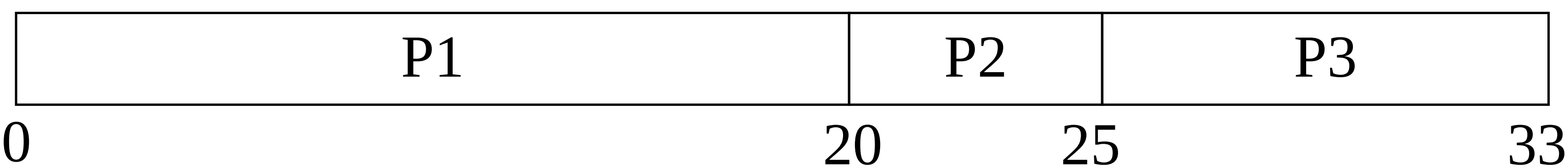
- Maximum CPU utilization: CPU always used
- Minimum response time: makes users happy
- Minimum waiting time
  - » Makes users happy
  - » Reduces number of processes in process table
- Maximum throughput: more processes finishing is better
- Minimum turnaround time: users like their programs to finish faster

CPU Scheduling Algorithms

- Use various characteristics of processes to pick the one to run next
  - » Process priority
  - » Length of time a process has been running
  - » Time it has left
  - » When it last ran
  - » Others?
- Try to optimize one or more of the scheduling criteria
  - » Generally impossible to optimize *all* criteria at once
  - » Pick criteria to optimize based on type of computer system

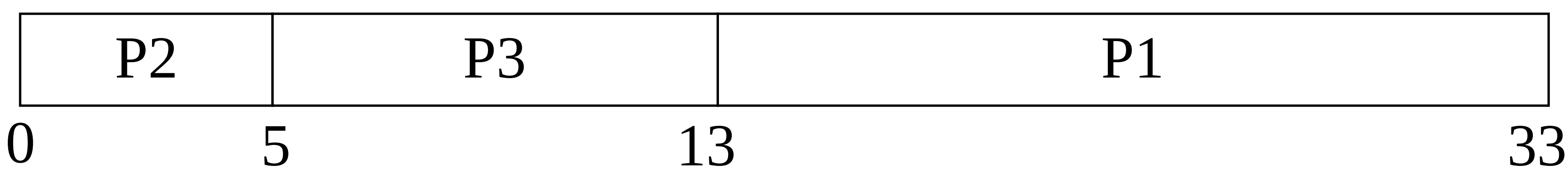
First-Come, First-Served (FCFS)

- Run processes in the order they were started
- Non-preemptive: processes run until they terminate or make an I/O request
- Example: three processes
  - » P1: burst time 20
  - » P2: burst time 5
  - » P3: burst time 8
- Processes arrive in order P1, P2, P3
- Waiting time for P1=0, P2=20, P3=25
- Average waiting time = (0+20+25)/3 = 15



First-Come, First-Served (FCFS)

- Example: three processes
  - » P1: burst time 20
  - » P2: burst time 5
  - » P3: burst time 8
- Processes arrive in order P2, P3, P1
- Waiting time for P1=13, P2=0, P3=5
- Average waiting time = (5+0+13)/3 = 6
- Exhibits the <u>convoy effect</u>
  - » Waiting time much lower
  - » Short processes not stuck behind long process

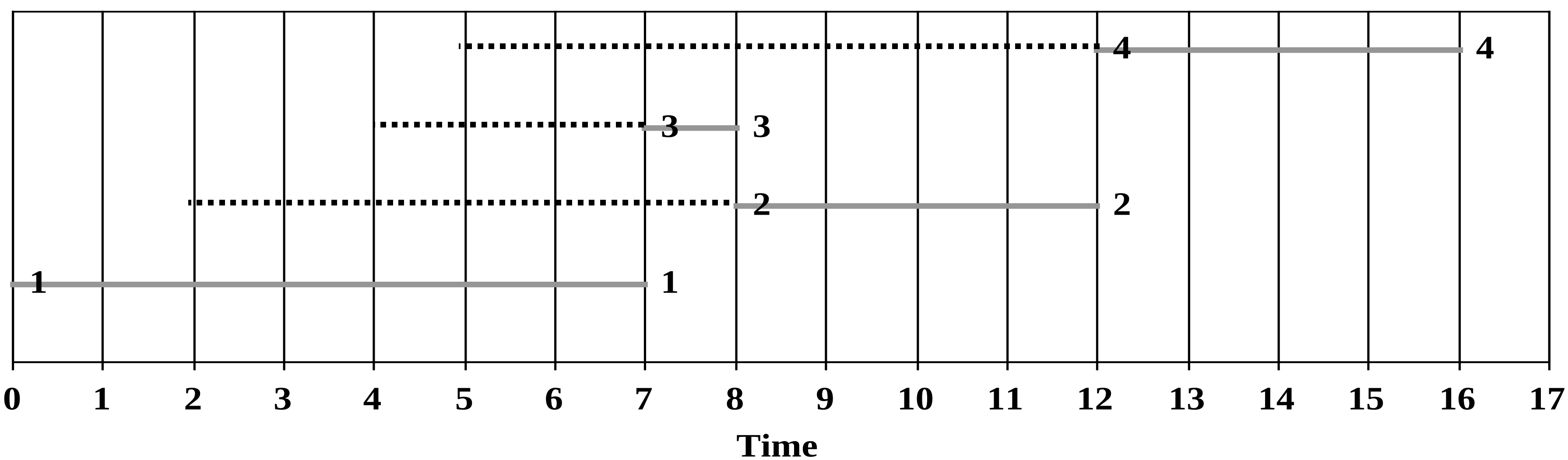


Shortest Job First (SJF)

- Pick the process whose CPU burst is shortest
  - » Requires knowledge of burst times in advance of schedule
- Two possibilities:
  - » Non-preemptive: the process keeps the CPU until it finishes its CPU burst
  - » Preemptive: if a new process arrives, preempt if the new process's burst time is shorter than the *remainder* of the current process' burst time. Also known as Shortest-Time-Remaining-First (SRTF)
- SJF is optimal for waiting time
  - » Minimum average waiting time for a given set of processes
  - » Not necessarily optimal for other criteria

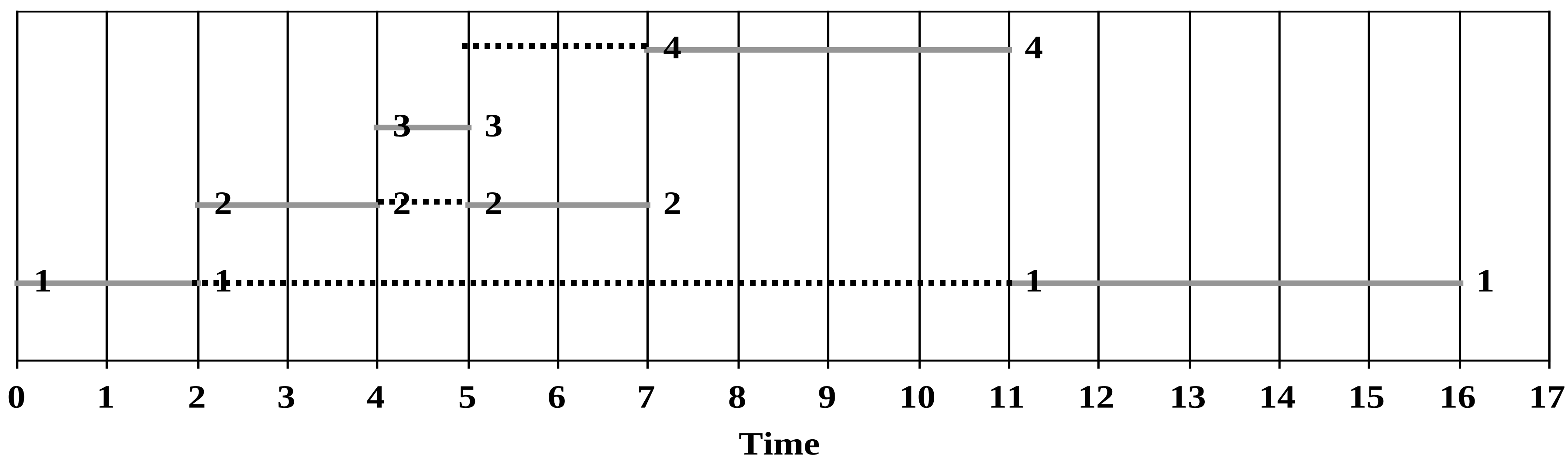
Non-Preemptive SJF

- Four processes:
  - » P1 arrives at 0.0, burst time 7
  - » P2 arrives at 2.0, burst time 4
  - » P3 arrives at 4.0, burst time 1
  - » P4 arrives at 5.0, burst time 4
- Average waiting time = (0 + 6 + 3 + 7)/4 = 4



Preemptive SJF

- Four processes:
  - » P1 arrives at 0.0, burst time 7
  - » P2 arrives at 2.0, burst time 4
  - » P3 arrives at 4.0, burst time 1
  - » P4 arrives at 5.0, burst time 4
- Average waiting time = (9 + 0 + 1 + 2)/4 = 3



How Long Is The Next CPU Burst?

- Don't know in advance
  - Estimate the length using previous behavior
  - Use a formula based on one or more previous bursts
- Use exponential averaging to determine length
  - t_n = actual length of n^{th} CPU burst
  - x_n = predicted length of nth CPU burst
  - Averaging factor k, 0 <= k <= 1
  - Predict $x_{n+1} = k\ t_n + (1-k)\ x_n$

Using Exponential Averaging

- Only the last burst counts
  - » Predict next burst is same length as previous burst
  - » k = 1
- Recent history doesn’t count
  - » Use same prediction every time, and never correct it
  - » k = 0
- In general, more recent bursts count more
  - » Value of older bursts “decays” over time
  - » Rate of decay depends on k
  - » Larger values of k mean faster decay

Priority Scheduling

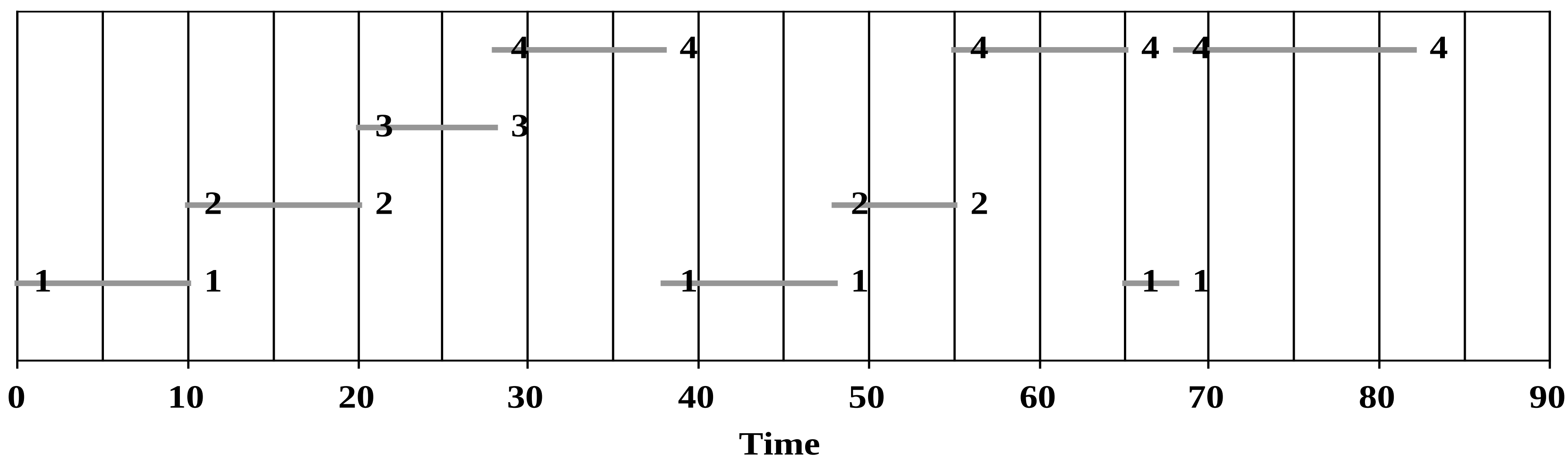
- Each process is assigned a priority (integer)
- CPU runs the ready process with the highest priority (smaller integers are higher priority)
- Can be done either
  - » Non-preemptive (highest priority runs until end of burst)
  - » Preemptive (check priorities when processes become ready)
- All algorithms are some form of priority scheduling
  - » SJF: priority = predicted CPU burst
  - » FCFS: priority = arrival time at CPU
- Starvation: low-priority processes may never complete
  - » Problem: high-priority processes keep low-priority ones from finishing
  - » Solution: priority increases as process ages (waits)

Round Robin (RR)

- Each process gets a small amount of CPU time (called a time quantum)
  - » Process preempted after its quantum is up
  - » Process then added to ready queue
- CPU time divided evenly between processes
  - » n processes in the ready queue
  - » Time quantum is q
  - » Each process gets $1/n$ of the CPU time in chunks of q time units at a time
  - » Waiting time is at most $(n-1)q$ time units
- Performance
  - » q is large -> FCFS
  - » q is small -> too much overhead doing context switches

Example: RR with Quantum 10

- Four processes:
  - » P1 has burst time 23
  - » P2 has burst time 17
  - » P3 has burst time 8
  - » P4 has burst time 34
- Higher average turnaround than SRTF, but better response time



Multilevel Queuing

- Several different ready queues, for example
  - » Foreground (interactive) / background (batch)
  - » System tasks / user tasks
- Each queue has its own scheduling algorithm
  - » Foreground -> RR, background -> FCFS
  - » System tasks: SJF, user tasks -> RR
- Scheduling must be done between queues
  - » Fixed priority
    - – All processes in higher-priority queues have priority
    - – Processes in lower-priority queues can starve
  - » Time-slicing
    - – Time is shared between queues (possibly unevenly)
    - – Each queue can allocate its time to its own processes

Multilevel Feedback Queuing

- Processes may be moved from one queue to another
  - » Long-running processes get lower priority
  - » Processes that have been waiting a lot get higher priority
- Multilevel feedback queue scheduling:
  - » Number of queues
  - » Scheduling algorithm for each queue
  - » Movement algorithms for processes
    - – When does a process get upgraded?
    - – When does a process get downgraded?
  - » Method to decide which queue a process starts in

Multilevel Feedback Queue Example

- Three queues
  - » Q_0: time quantum = 10 ms
  - » Q_1 : time quantum = 50 ms
  - » Q_2 : FCFS (may be preempted)
- Scheduling
  - » Processes enter Q_0, where they get up to 50 ms
  - » If a process in Q_0 doesn't complete in 50 ms, it's moved to Q_1
  - » Processes in are given 500 ms to finish
  - » If a process in Q_1 doesn't complete in 500 ms, it's moved to Q_2
  - » Processes in Q_2 run in the order in which they were demoted to Q_2
  - » Processes in Q_2 only use time unused by Q_0 and Q_1

Scheduling for Multiple CPUs

- Some systems have several CPUs that can be scheduled as a group
  - » Similar to uniprocessor scheduling, but can run more than one process at a time
  - » May be trickier issues: some processes may prefer a certain CPU
- Issues to consider
  - » Goal: <u>load sharing</u> - utilize all processors evenly
  - » Homogenous multiprocessing: all CPUs are the same
    - – Some CPUs may be "closer" to the data they need to run a particular process
  - » Asymmetric multiprocessing: processors have different jobs (graphics processing / data processing)

Real-Time Scheduling

- Some processes have time constraints on their completion time or progress
- Hard real-time systems
  - » Critical tasks must complete within a certain time
  - » Tasks that don't complete within their allotted time are useless
  - » Examples
    - – Medical systems
    - – Aircraft electronics
- Soft real-time systems
  - » Tasks must complete within a certain time to be useful
  - » Not fatal if all tasks don't complete fast enough
  - » Example: video games
- Real-time scheduling may be combined other algorithms

Performance Evaluation of Schedulers

