CHAPTER 3: PROCESSES AND PROCESS SCHEDULING

Dr\ Eman Monir

CHAPTER 3: PROCESSES

- PROCESS CONCEPT
- PROCESS SCHEDULING
- OPERATIONS ON PROCESSES
- INTERPROCESS COMMUNICATION
- EXAMPLES OF IPC SYSTEMS
- COMMUNICATION IN CLIENT-SERVER SYSTEMS

OBJECTIVES

- TO INTRODUCE THE NOTION OF A PROCESS -- A PROGRAM IN EXECUTION, WHICH FORMS THE BASIS OF ALL COMPUTATION
- TO DESCRIBE THE VARIOUS FEATURES OF PROCESSES, INCLUDING SCHEDULING, CREATION AND TERMINATION, AND COMMUNICATION
- TO EXPLORE INTERPROCESS COMMUNICATION USING SHARED MEMORY AND MESSAGE PASSING
- TO DESCRIBE COMMUNICATION IN CLIENT-SERVER SYSTEMS

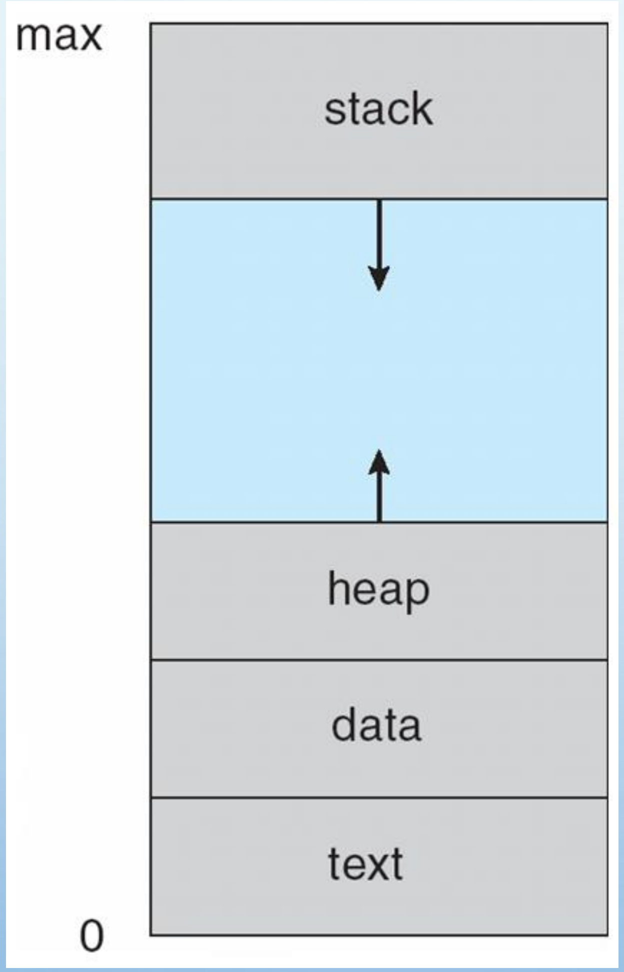
PROCESS CONCEPT

- AN OPERATING SYSTEM EXECUTES A VARIETY OF PROGRAMS:
  - BATCH SYSTEM – JOBS
  - TIME-SHARED SYSTEMS – USER PROGRAMS OR TASKS
- TEXTBOOK USES THE TERMS *JOB* AND *PROCESS* ALMOST INTERCHANGEABLY
- PROCESS – A PROGRAM IN EXECUTION; PROCESS EXECUTION MUST PROGRESS IN SEQUENTIAL FASHION
- MULTIPLE PARTS
  - THE PROGRAM CODE, ALSO CALLED TEXT SECTION
  - CURRENT ACTIVITY INCLUDING PROGRAM COUNTER, PROCESSOR REGISTERS
  - STACK CONTAINING TEMPORARY DATA
    - FUNCTION PARAMETERS, RETURN ADDRESSES, LOCAL VARIABLES
  - DATA SECTION CONTAINING GLOBAL VARIABLES
  - HEAP CONTAINING MEMORY DYNAMICALLY ALLOCATED DURING RUN TIME

PROCESS CONCEPT (CONT.)

- PROGRAM IS *PASSIVE* ENTITY STORED ON DISK (EXECUTABLE FILE), PROCESS IS *ACTIVE*
  - PROGRAM BECOMES PROCESS WHEN EXECUTABLE FILE LOADED INTO MEMORY
- EXECUTION OF PROGRAM STARTED VIA GUI MOUSE CLICKS, COMMAND LINE ENTRY OF ITS NAME, ETC
- ONE PROGRAM CAN BE SEVERAL PROCESSES
  - CONSIDER MULTIPLE USERS EXECUTING THE SAME PROGRAM

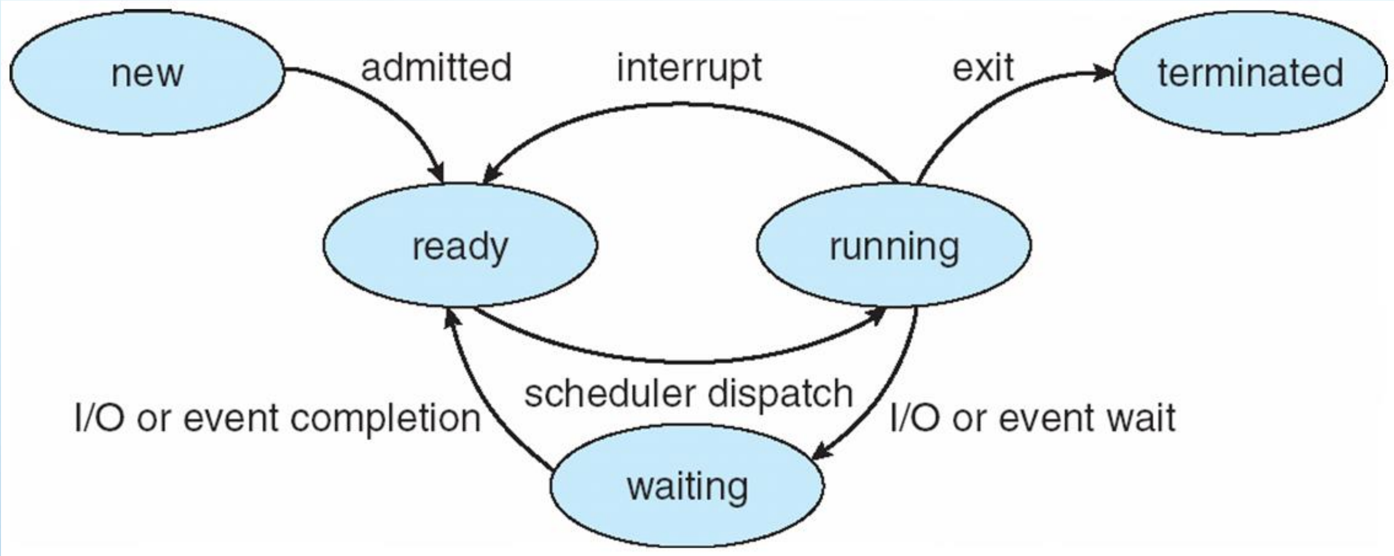
PROCESS IN MEMORY



PROCESS STATE

- AS A PROCESS EXECUTES, IT CHANGES **STATE**
  - **NEW**: THE PROCESS IS BEING CREATED
  - **RUNNING**: INSTRUCTIONS ARE BEING EXECUTED
  - **WAITING**: THE PROCESS IS WAITING FOR SOME EVENT TO OCCUR
  - **READY**: THE PROCESS IS WAITING TO BE ASSIGNED TO A PROCESSOR
  - **TERMINATED**: THE PROCESS HAS FINISHED EXECUTION

DIAGRAM OF PROCESS STATE

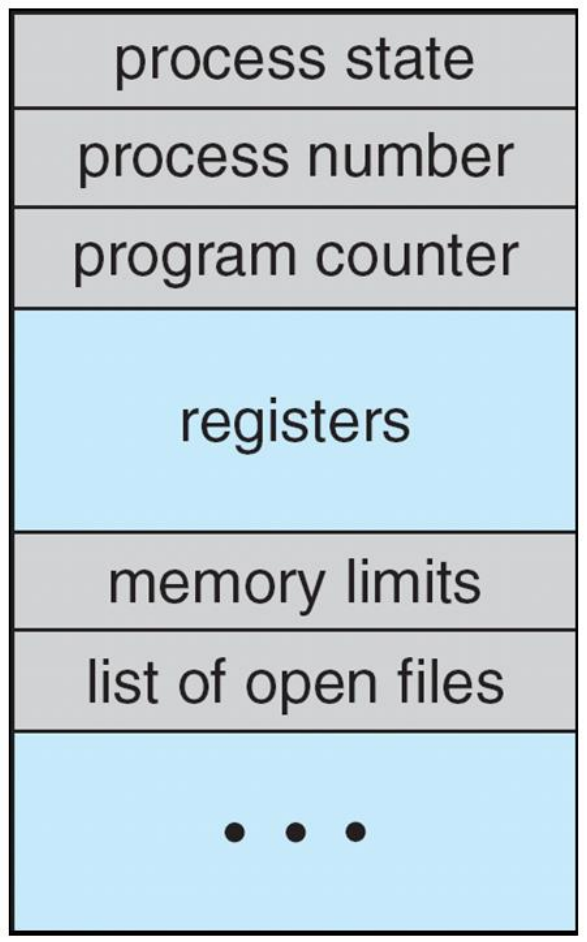


PROCESS CONTROL BLOCK (PCB)

INFORMATION ASSOCIATED WITH EACH PROCESS

(ALSO CALLED TASK CONTROL BLOCK)

- PROCESS STATE – RUNNING, WAITING, ETC
- PROGRAM COUNTER – LOCATION OF INSTRUCTION TO NEXT EXECUTE
- CPU REGISTERS – CONTENTS OF ALL PROCESS-CENTRIC REGISTERS
- CPU SCHEDULING INFORMATION– PRIORITIES, SCHEDULING QUEUE POINTERS
- MEMORY-MANAGEMENT INFORMATION – MEMORY ALLOCATED TO THE PROCESS
- ACCOUNTING INFORMATION – CPU USED, CLOCK TIME ELAPSED SINCE START, TIME LIMITS
- I/O STATUS INFORMATION – I/O DEVICES ALLOCATED TO PROCESS, LIST OF OPEN FILES



THREADS

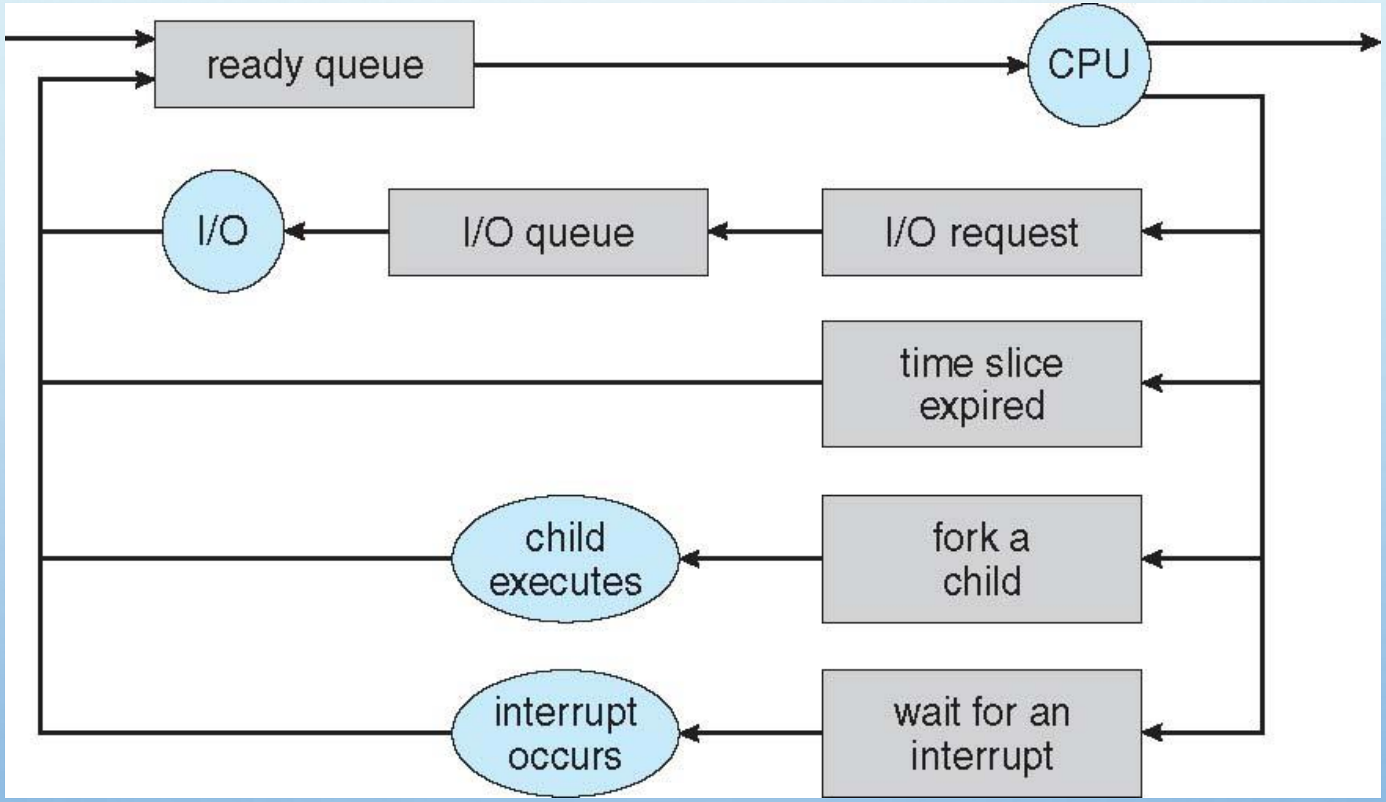
- SO FAR, PROCESS HAS A SINGLE THREAD OF EXECUTION
- CONSIDER HAVING MULTIPLE PROGRAM COUNTERS PER PROCESS
  - MULTIPLE LOCATIONS CAN EXECUTE AT ONCE
    - MULTIPLE THREADS OF CONTROL -> THREADS
- MUST THEN HAVE STORAGE FOR THREAD DETAILS, MULTIPLE PROGRAM COUNTERS IN PCB
- SEE NEXT CHAPTER

PROCESS SCHEDULING

- MAXIMIZE CPU USE, QUICKLY SWITCH PROCESSES ONTO CPU FOR TIME SHARING
- PROCESS SCHEDULER SELECTS AMONG AVAILABLE PROCESSES FOR NEXT EXECUTION ON CPU
- MAINTAINS SCHEDULING QUEUES OF PROCESSES
  - JOB QUEUE – SET OF ALL PROCESSES IN THE SYSTEM
  - READY QUEUE – SET OF ALL PROCESSES RESIDING IN MAIN MEMORY, READY AND WAITING TO EXECUTE
  - DEVICE QUEUES – SET OF PROCESSES WAITING FOR AN I/O DEVICE
  - PROCESSES MIGRATE AMONG THE VARIOUS QUEUES

REPRESENTATION OF PROCESS SCHEDULING

- **Queueing diagram** represents queues, resources, flows



SCHEDULERS

- SHORT-TERM SCHEDULER (OR CPU SCHEDULER) – SELECTS WHICH PROCESS SHOULD BE EXECUTED NEXT AND ALLOCATES CPU
  - SOMETIMES THE ONLY SCHEDULER IN A SYSTEM
  - SHORT-TERM SCHEDULER IS INVOKED FREQUENTLY (MILLISECONDS) $\Rightarrow$ (MUST BE FAST)
- LONG-TERM SCHEDULER (OR JOB SCHEDULER) – SELECTS WHICH PROCESSES SHOULD BE BROUGHT INTO THE READY QUEUE
  - LONG-TERM SCHEDULER IS INVOKED INFREQUENTLY (SECONDS, MINUTES) $\Rightarrow$ (MAY BE SLOW)
  - THE LONG-TERM SCHEDULER CONTROLS THE DEGREE OF MULTIPROGRAMMING
- PROCESSES CAN BE DESCRIBED AS EITHER:
  - I/O-BOUND PROCESS – SPENDS MORE TIME DOING I/O THAN COMPUTATIONS, MANY SHORT CPU BURSTS
  - CPU-BOUND PROCESS – SPENDS MORE TIME DOING COMPUTATIONS; FEW VERY LONG CPU BURSTS
- LONG-TERM SCHEDULER STRIVES FOR GOOD ***PROCESS MIX***

MULTITASKING IN MOBILE SYSTEMS

- SOME MOBILE SYSTEMS (E.G., EARLY VERSION OF IOS) ALLOW ONLY ONE PROCESS TO RUN, OTHERS SUSPENDED
- DUE TO SCREEN REAL ESTATE, USER INTERFACE LIMITS IOS PROVIDES FOR A
  - SINGLE FOREGROUND PROCESS- CONTROLLED VIA USER INTERFACE
  - MULTIPLE BACKGROUND PROCESSES– IN MEMORY, RUNNING, BUT NOT ON THE DISPLAY, AND WITH LIMITS
  - LIMITS INCLUDE SINGLE, SHORT TASK, RECEIVING NOTIFICATION OF EVENTS, SPECIFIC LONG-RUNNING TASKS LIKE AUDIO PLAYBACK
- ANDROID RUNS FOREGROUND AND BACKGROUND, WITH FEWER LIMITS
  - BACKGROUND PROCESS USES A SERVICE TO PERFORM TASKS
  - SERVICE CAN KEEP RUNNING EVEN IF BACKGROUND PROCESS IS SUSPENDED
  - SERVICE HAS NO USER INTERFACE, SMALL MEMORY USE

CONTEXT SWITCH

- WHEN CPU SWITCHES TO ANOTHER PROCESS, THE SYSTEM MUST SAVE THE STATE OF THE OLD PROCESS AND LOAD THE SAVED STATE FOR THE NEW PROCESS VIA A CONTEXT SWITCH
- CONTEXT OF A PROCESS REPRESENTED IN THE PCB
- CONTEXT-SWITCH TIME IS OVERHEAD; THE SYSTEM DOES NO USEFUL WORK WHILE SWITCHING
  - THE MORE COMPLEX THE OS AND THE PCB ➔ THE LONGER THE CONTEXT SWITCH
- TIME DEPENDENT ON HARDWARE SUPPORT
  - SOME HARDWARE PROVIDES MULTIPLE SETS OF REGISTERS PER CPU ➔ MULTIPLE CONTEXTS LOADED AT ONCE

OPERATIONS ON PROCESSES

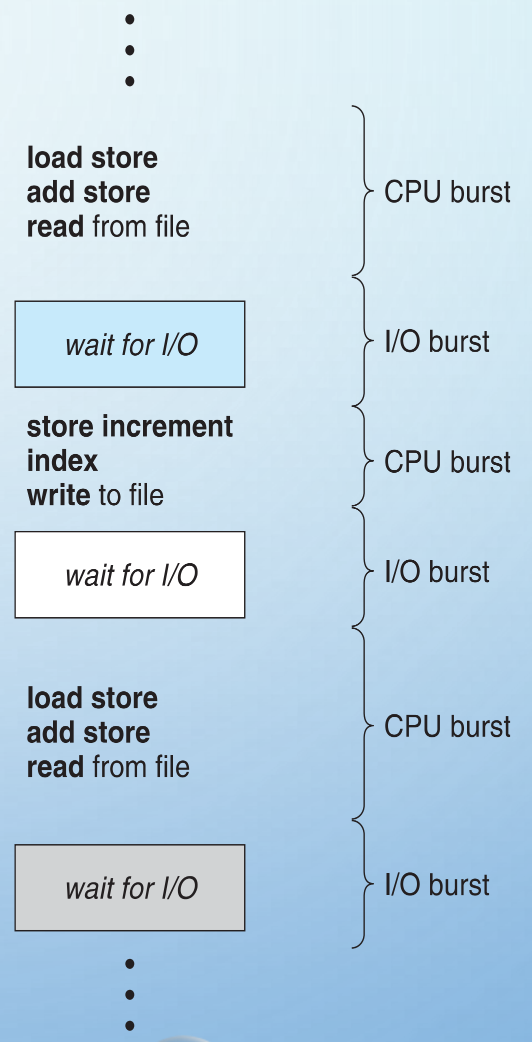
- SYSTEM MUST PROVIDE MECHANISMS FOR:
  - PROCESS CREATION,
  - PROCESS TERMINATION,
  - AND SO ON AS DETAILED NEXT

OBJECTIVES

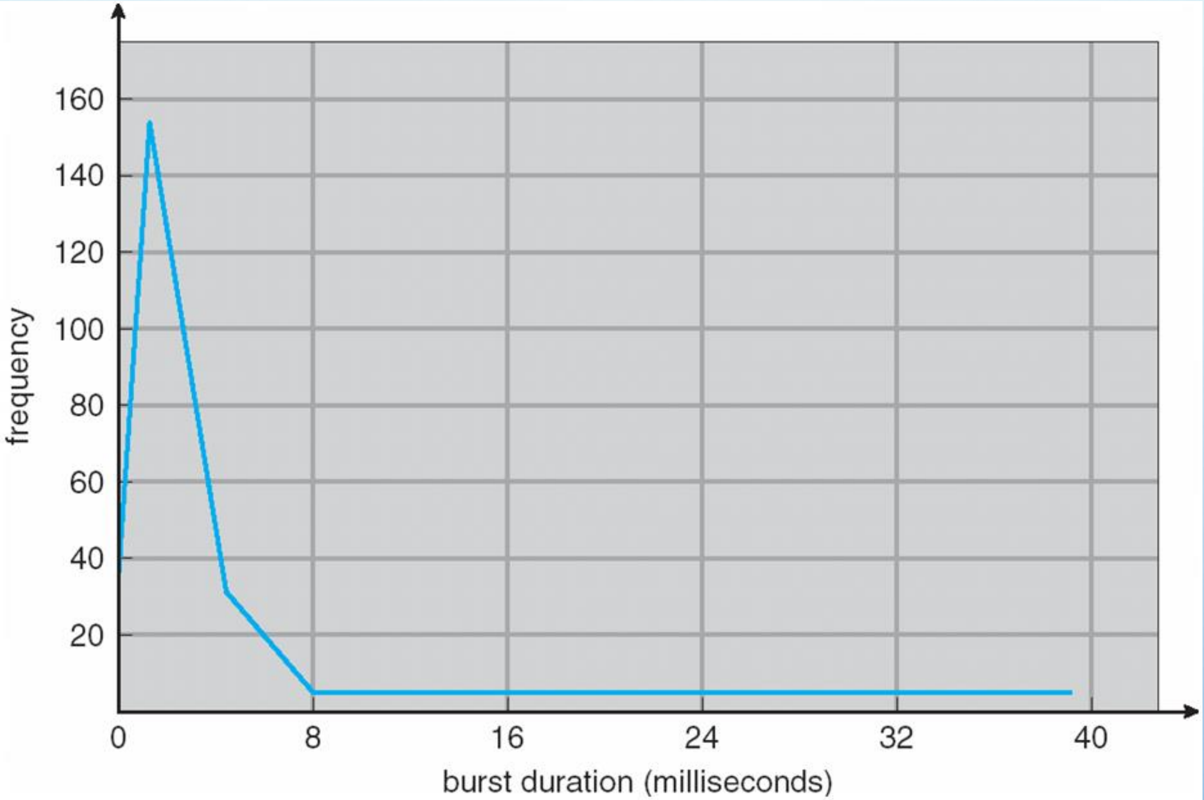
- TO INTRODUCE CPU SCHEDULING, WHICH IS THE BASIS FOR MULTIPROGRAMMED OPERATING SYSTEMS
- TO DESCRIBE VARIOUS CPU-SCHEDULING ALGORITHMS
- TO DISCUSS EVALUATION CRITERIA FOR SELECTING A CPU-SCHEDULING ALGORITHM FOR A PARTICULAR SYSTEM
- TO EXAMINE THE SCHEDULING ALGORITHMS OF SEVERAL OPERATING SYSTEMS

BASIC CONCEPTS

- MAXIMUM CPU UTILIZATION OBTAINED WITH MULTIPROGRAMMING
- CPU–I/O BURST CYCLE – PROCESS EXECUTION CONSISTS OF A CYCLE OF CPU EXECUTION AND I/O WAIT
- CPU BURST FOLLOWED BY I/O BURST
- CPU BURST DISTRIBUTION IS OF MAIN CONCERN



HISTOGRAM OF CPU-BURST TIMES



CPU SCHEDULER

- SHORT-TERM SCHEDULER SELECTS FROM AMONG THE PROCESSES IN READY QUEUE, AND ALLOCATES THE CPU TO ONE OF THEM
  - QUEUE MAY BE ORDERED IN VARIOUS WAYS
- CPU SCHEDULING DECISIONS MAY TAKE PLACE WHEN A PROCESS:
  1. SWITCHES FROM RUNNING TO WAITING STATE
  2. SWITCHES FROM RUNNING TO READY STATE
  3. SWITCHES FROM WAITING TO READY
  4. TERMINATES
- SCHEDULING UNDER 1 AND 4 IS NONPREEMPTIVE
- ALL OTHER SCHEDULING IS PREEMPTIVE
  - CONSIDER ACCESS TO SHARED DATA
  - CONSIDER PREEMPTION WHILE IN KERNEL MODE
  - CONSIDER INTERRUPTS OCCURRING DURING CRUCIAL OS ACTIVITIES

SCHEDULING CRITERIA

- **CPU UTILIZATION** – KEEP THE CPU AS BUSY AS POSSIBLE
- **THROUGHPUT** – # OF PROCESSES THAT COMPLETE THEIR EXECUTION PER TIME UNIT
- **TURNAROUND TIME** – AMOUNT OF TIME TO EXECUTE A PARTICULAR PROCESS
- **WAITING TIME** – AMOUNT OF TIME A PROCESS HAS BEEN WAITING IN THE READY QUEUE
- **RESPONSE TIME** – AMOUNT OF TIME IT TAKES FROM WHEN A REQUEST WAS SUBMITTED UNTIL THE FIRST RESPONSE IS PRODUCED, NOT OUTPUT (FOR TIME-SHARING ENVIRONMENT)

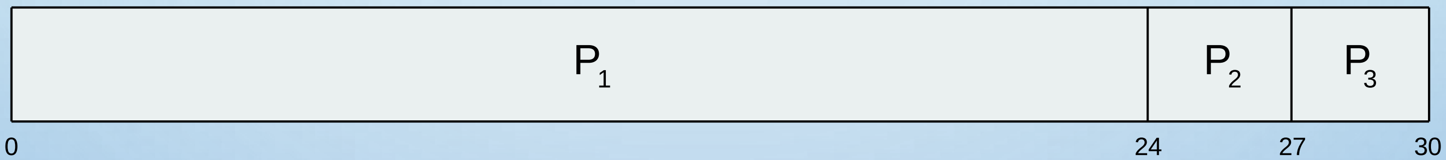
SCHEDULING ALGORITHM OPTIMIZATION CRITERIA

- MAX CPU UTILIZATION
- MAX THROUGHPUT
- MIN TURNAROUND TIME
- MIN WAITING TIME
- MIN RESPONSE TIME

FIRST- COME, FIRST-SERVED (FCFS) SCHEDULING

PROCESS	BURST TIME
P_1	24
P_2	3
P_3	3

- SUPPOSE THAT THE PROCESSES ARRIVE IN THE ORDER: P_1 , P_2 , P_3
  THE GANTT CHART FOR THE SCHEDULE IS:



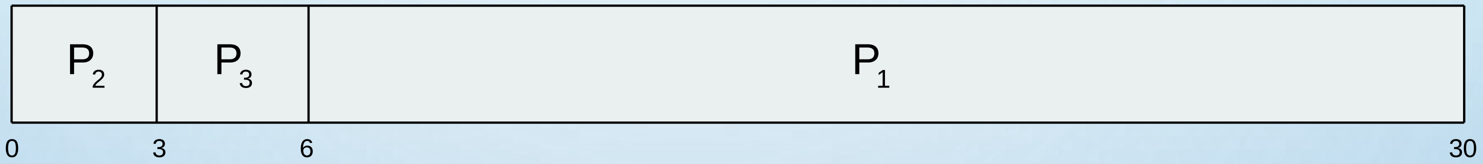
- WAITING TIME FOR $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- AVERAGE WAITING TIME: $(0 + 24 + 27)/3 = 17$

FCFS SCHEDULING (CONT.)

SUPPOSE THAT THE PROCESSES ARRIVE IN THE ORDER:

$$P_2\ ,\ P_3\ ,\ P_1$$

- THE GANTT CHART FOR THE SCHEDULE IS:



- WAITING TIME FOR $P_1 = 6; P_2 = 0; P_3 = 3$
- AVERAGE WAITING TIME: $(6 + 0 + 3)/3 = 3$
- MUCH BETTER THAN PREVIOUS CASE
- CONVOY EFFECT - SHORT PROCESS BEHIND LONG PROCESS
  - CONSIDER ONE CPU-BOUND AND MANY I/O-BOUND PROCESSES

SHORTEST-JOB-FIRST (SJF) SCHEDULING

- ASSOCIATE WITH EACH PROCESS THE LENGTH OF ITS NEXT CPU BURST
  - USE THESE LENGTHS TO SCHEDULE THE PROCESS WITH THE SHORTEST TIME
- SJF IS OPTIMAL – GIVES MINIMUM AVERAGE WAITING TIME FOR A GIVEN SET OF PROCESSES
  - THE DIFFICULTY IS KNOWING THE LENGTH OF THE NEXT CPU REQUEST
  - COULD ASK THE USER

EXAMPLE OF SJF

PROCESS	ARRIVAL TIME	BURST TIME
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- SJF SCHEDULING CHART

P_4	P_1	P_3	P_2

0 3 9 16 24

- AVERAGE WAITING TIME = (3 + 16 + 9 + 0) / 4 = 7

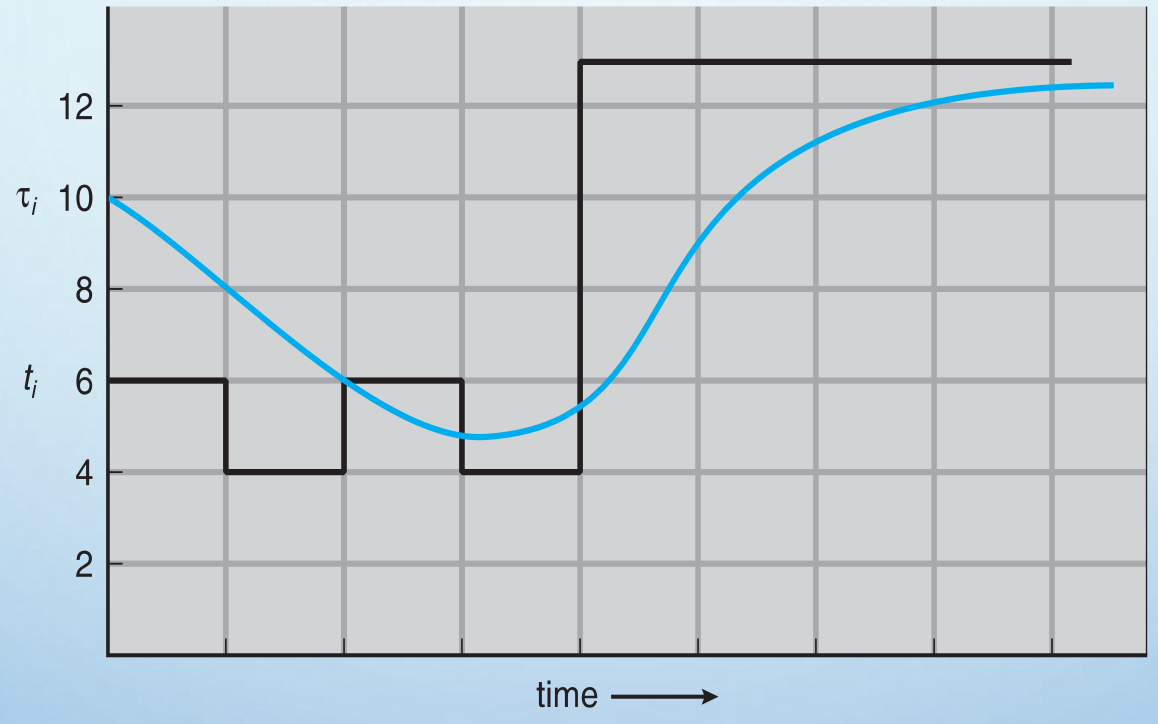
DETERMINING LENGTH OF NEXT CPU BURST

- CAN ONLY ESTIMATE THE LENGTH – SHOULD BE SIMILAR TO THE PREVIOUS ONE
  - THEN PICK PROCESS WITH SHORTEST PREDICTED NEXT CPU BURST

- CAN BE DONE BY USING THE LENGTH OF PREVIOUS CPU BURSTS, USING EXPONENTIAL AVERAGING
  1. $t_n =$ actual length of n^{th} CPU burst
  2. $\tau_{n+1} =$ predicted value for the next CPU burst
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define : $\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n.$

- COMMONLY, Α SET TO ½
- PREEMPTIVE VERSION CALLED SHORTEST-REMAINING-TIME-FIRST

PREDICTION OF THE LENGTH OF THE NEXT CPU BURST



CPU burst (t_i)		6	4	6	4	13	13	13 …
"guess" (τ_i)	10	8	6	6	5	9	11	12 …

EXAMPLES OF EXPONENTIAL AVERAGING

- $\alpha = 0$
  - $\tau_{N+1} = \tau_N$
  - RECENT HISTORY DOES NOT COUNT
- $\alpha = 1$
  - $\tau_{N+1} = \alpha \; T_N$
  - ONLY THE ACTUAL LAST CPU BURST COUNTS
- IF WE EXPAND THE FORMULA, WE GET:

$$\begin{aligned}\tau_{N+1} &= \alpha\, T_N + (1 - \alpha)\alpha\, T_{N-1} + \ldots \\ &+ (1 - \alpha)^J \alpha\, T_{N-J} + \ldots \\ &+ (1 - \alpha)^{N+1} \tau_0\end{aligned}$$

- SINCE BOTH α AND $(1 - \alpha)$ ARE LESS THAN OR EQUAL TO 1, EACH SUCCESSIVE TERM HAS LESS WEIGHT THAN ITS PREDECESSOR

EXAMPLE OF SHORTEST-REMAINING-TIME-FIRST

- NOW WE ADD THE CONCEPTS OF VARYING ARRIVAL TIMES AND PREEMPTION TO THE ANALYSIS

PROCESS	*ARRIVAL* TIME	BURST TIME
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *PREEMPTIVE* SJF GANTT CHART

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

- AVERAGE WAITING TIME = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 MSEC

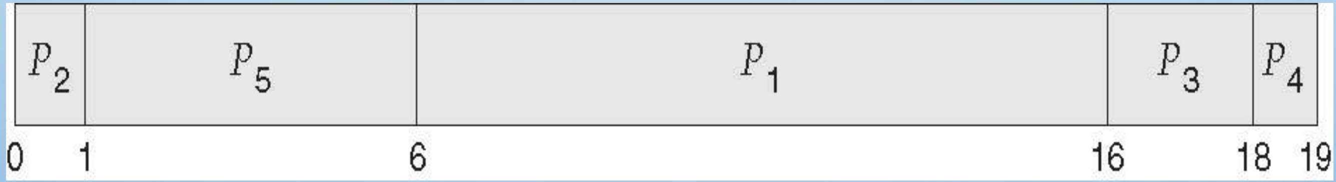
PRIORITY SCHEDULING

- A PRIORITY NUMBER (INTEGER) IS ASSOCIATED WITH EACH PROCESS
- THE CPU IS ALLOCATED TO THE PROCESS WITH THE HIGHEST PRIORITY (SMALLEST INTEGER ≡ HIGHEST PRIORITY)
  - PREEMPTIVE
  - NONPREEMPTIVE
- SJF IS PRIORITY SCHEDULING WHERE PRIORITY IS THE INVERSE OF PREDICTED NEXT CPU BURST TIME
- PROBLEM ≡ STARVATION – LOW PRIORITY PROCESSES MAY NEVER EXECUTE
- SOLUTION ≡ AGING – AS TIME PROGRESSES INCREASE THE PRIORITY OF THE PROCESS

EXAMPLE OF PRIORITY SCHEDULING

PROCESS	BURST TIME	PRIORITY
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- PRIORITY SCHEDULING GANTT CHART



- AVERAGE WAITING TIME = 8.2 MSEC

ROUND ROBIN (RR)

- EACH PROCESS GETS A SMALL UNIT OF CPU TIME (TIME QUANTUM *Q*), USUALLY 10-100 MILLISECONDS. AFTER THIS TIME HAS ELAPSED, THE PROCESS IS PREEMPTED AND ADDED TO THE END OF THE READY QUEUE.
- IF THERE ARE N PROCESSES IN THE READY QUEUE AND THE TIME QUANTUM IS Q, THEN EACH PROCESS GETS $1/N$ OF THE CPU TIME IN CHUNKS OF AT MOST Q TIME UNITS AT ONCE. NO PROCESS WAITS MORE THAN $(N-1)Q$ TIME UNITS.
- TIMER INTERRUPTS EVERY QUANTUM TO SCHEDULE NEXT PROCESS
- PERFORMANCE
  - Q LARGE $\Rightarrow$ FIFO
  - Q SMALL $\Rightarrow$ Q MUST BE LARGE WITH RESPECT TO CONTEXT SWITCH, OTHERWISE OVERHEAD IS TOO HIGH

EXAMPLE OF RR WITH TIME QUANTUM = 4

PROCESS	BURST TIME
P_1	24
P_2	3
P_3	3

- THE GANTT CHART IS:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

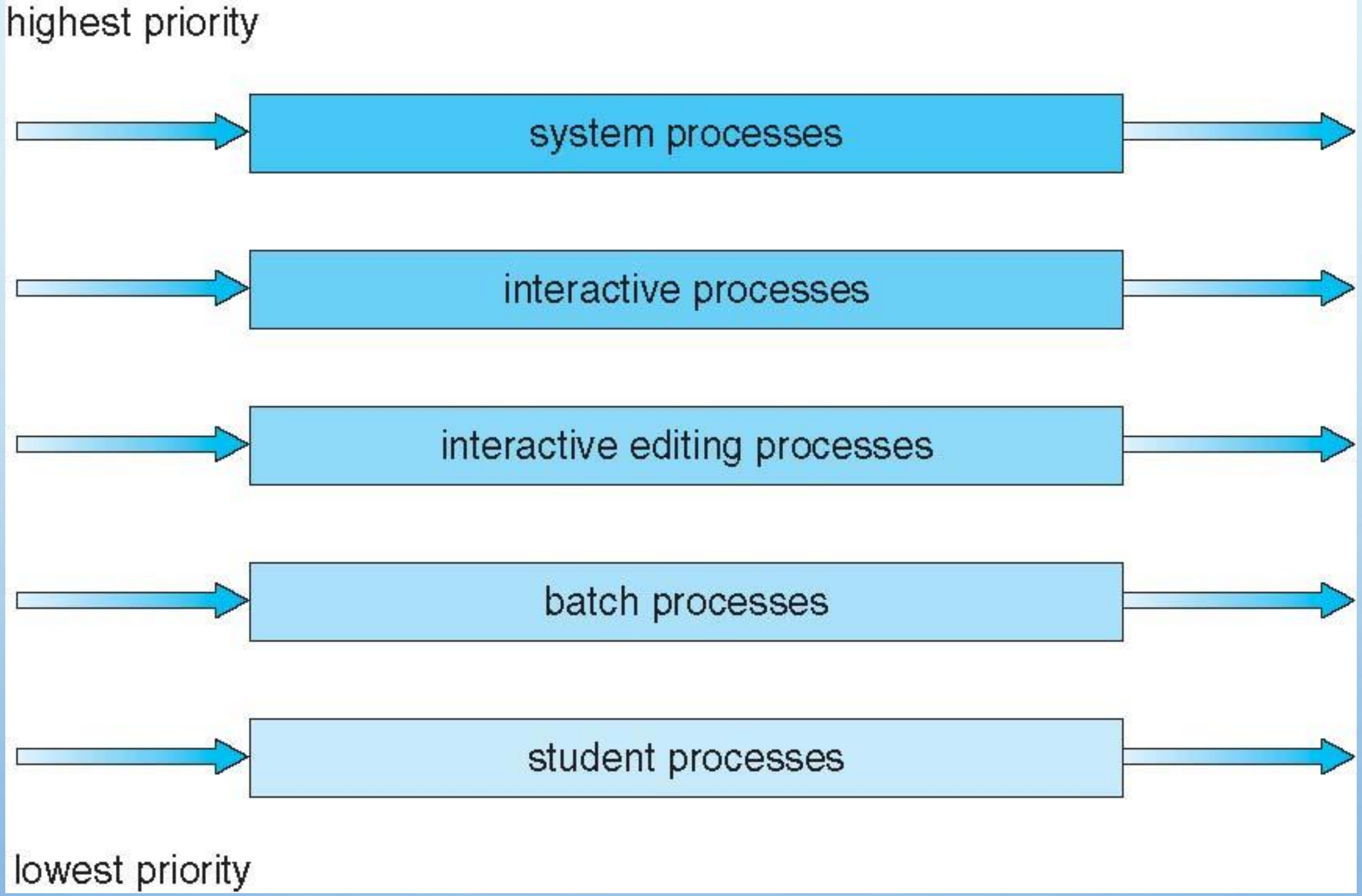
0 4 7 10 14 18 22 26 30

- TYPICALLY, HIGHER AVERAGE TURNAROUND THAN SJF, BUT BETTER ***RESPONSE***
- Q SHOULD BE LARGE COMPARED TO CONTEXT SWITCH TIME
- Q USUALLY 10MS TO 100MS, CONTEXT SWITCH < 10 USEC

MULTILEVEL QUEUE

- READY QUEUE IS PARTITIONED INTO SEPARATE QUEUES, EG:
  - FOREGROUND (INTERACTIVE)
  - BACKGROUND (BATCH)
- PROCESS PERMANENTLY IN A GIVEN QUEUE
- EACH QUEUE HAS ITS OWN SCHEDULING ALGORITHM:
  - FOREGROUND – RR
  - BACKGROUND – FCFS
- SCHEDULING MUST BE DONE BETWEEN THE QUEUES:
  - FIXED PRIORITY SCHEDULING; (I.E., SERVE ALL FROM FOREGROUND THEN FROM BACKGROUND). POSSIBILITY OF STARVATION.
  - TIME SLICE – EACH QUEUE GETS A CERTAIN AMOUNT OF CPU TIME WHICH IT CAN SCHEDULE AMONGST ITS PROCESSES; I.E., 80% TO FOREGROUND IN RR
  - 20% TO BACKGROUND IN FCFS

MULTILEVEL QUEUE SCHEDULING



MULTILEVEL FEEDBACK QUEUE

- A PROCESS CAN MOVE BETWEEN THE VARIOUS QUEUES; AGING CAN BE IMPLEMENTED THIS WAY
- MULTILEVEL-FEEDBACK-QUEUE SCHEDULER DEFINED BY THE FOLLOWING PARAMETERS:
  - NUMBER OF QUEUES
  - SCHEDULING ALGORITHMS FOR EACH QUEUE
  - METHOD USED TO DETERMINE WHEN TO UPGRADE A PROCESS
  - METHOD USED TO DETERMINE WHEN TO DEMOTE A PROCESS
  - METHOD USED TO DETERMINE WHICH QUEUE A PROCESS WILL ENTER WHEN THAT PROCESS NEEDS SERVICE

THREAD SCHEDULING

- DISTINCTION BETWEEN USER-LEVEL AND KERNEL-LEVEL THREADS
- WHEN THREADS SUPPORTED, THREADS SCHEDULED, NOT PROCESSES
- MANY-TO-ONE AND MANY-TO-MANY MODELS, THREAD LIBRARY SCHEDULES USER-LEVEL THREADS TO RUN ON LWP
  - KNOWN AS PROCESS-CONTENTION SCOPE (PCS) SINCE SCHEDULING COMPETITION IS WITHIN THE PROCESS
  - TYPICALLY DONE VIA PRIORITY SET BY PROGRAMMER
- KERNEL THREAD SCHEDULED ONTO AVAILABLE CPU IS SYSTEM-CONTENTION SCOPE (SCS) – COMPETITION AMONG ALL THREADS IN SYSTEM

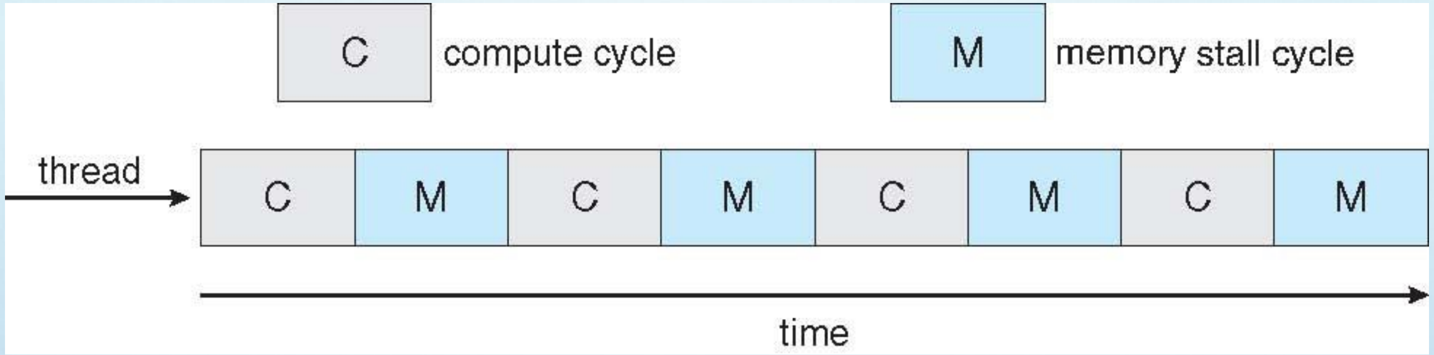
MULTIPLE-PROCESSOR SCHEDULING – LOAD BALANCING

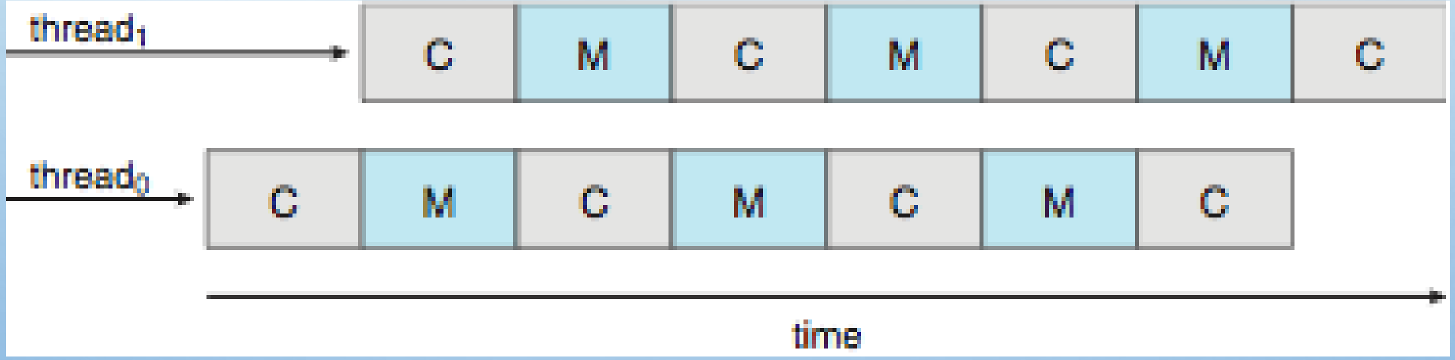
- IF SMP, NEED TO KEEP ALL CPUS LOADED FOR EFFICIENCY
- LOAD BALANCING ATTEMPTS TO KEEP WORKLOAD EVENLY DISTRIBUTED
- PUSH MIGRATION – PERIODIC TASK CHECKS LOAD ON EACH PROCESSOR, AND IF FOUND PUSHES TASK FROM OVERLOADED CPU TO OTHER CPUS
- PULL MIGRATION – IDLE PROCESSORS PULLS WAITING TASK FROM BUSY PROCESSOR

MULTICORE PROCESSORS

- RECENT TREND TO PLACE MULTIPLE PROCESSOR CORES ON SAME PHYSICAL CHIP
- FASTER AND CONSUMES LESS POWER
- MULTIPLE THREADS PER CORE ALSO GROWING
  - TAKES ADVANTAGE OF MEMORY STALL TO MAKE PROGRESS ON ANOTHER THREAD WHILE MEMORY RETRIEVE HAPPENS

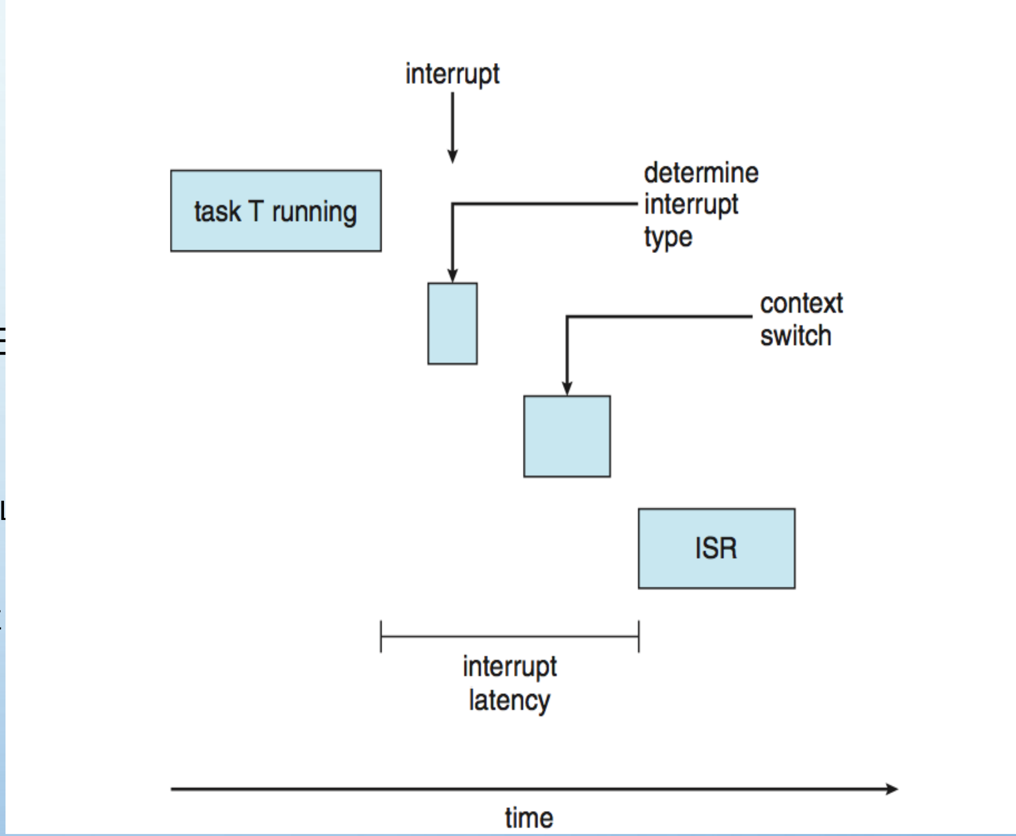
MULTITHREADED MULTICORE SYSTEM





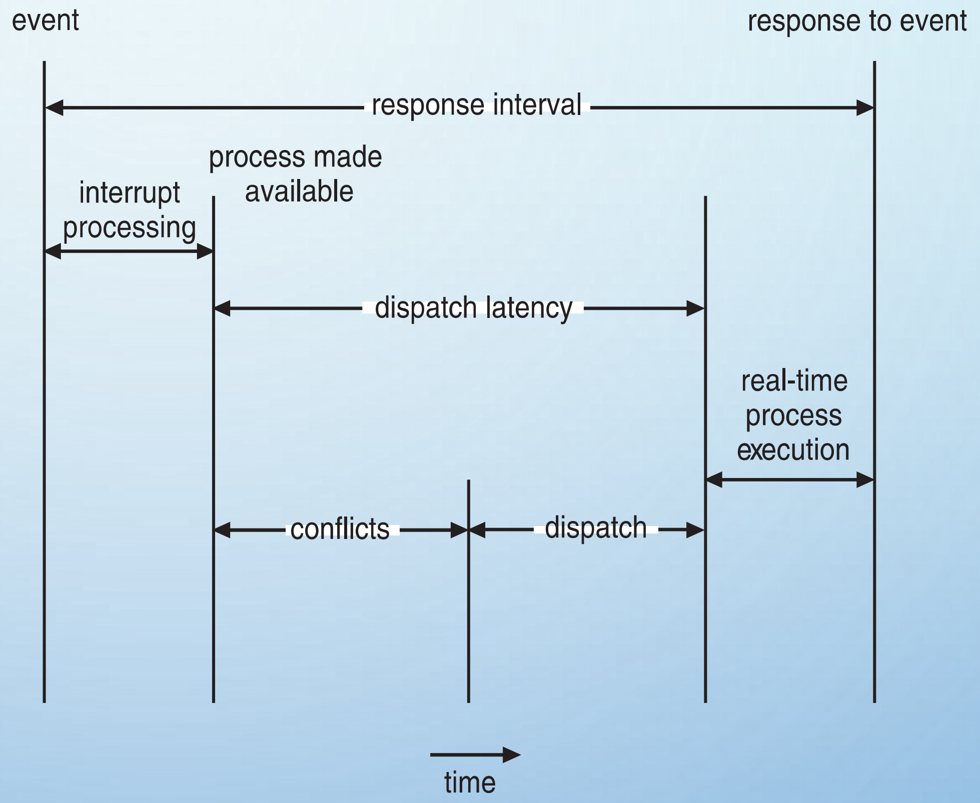
REAL-TIME CPU SCHEDULING

- CAN PRESENT OBVIOUS CHALLENGES
- SOFT REAL-TIME SYSTEMS – NO GUARANTEE AS TO WHEN CRITICAL REAL-TIME PROCESS WILL BE SCHEDULED
- HARD REAL-TIME SYSTEMS – TASK MUST BE SERVICED BY ITS DEADLINE
- TWO TYPES OF LATENCIES AFFECT PERFORMANCE
  1. INTERRUPT LATENCY – TIME FROM ARRIVAL OF INTERRUPT TO START OF ROUTINE THAT SERVICES INTERRUPT
  2. DISPATCH LATENCY – TIME FOR SCHEDULE TO TAKE CURRENT PROCESS OFF CPU AND SWITCH TO ANOTHER



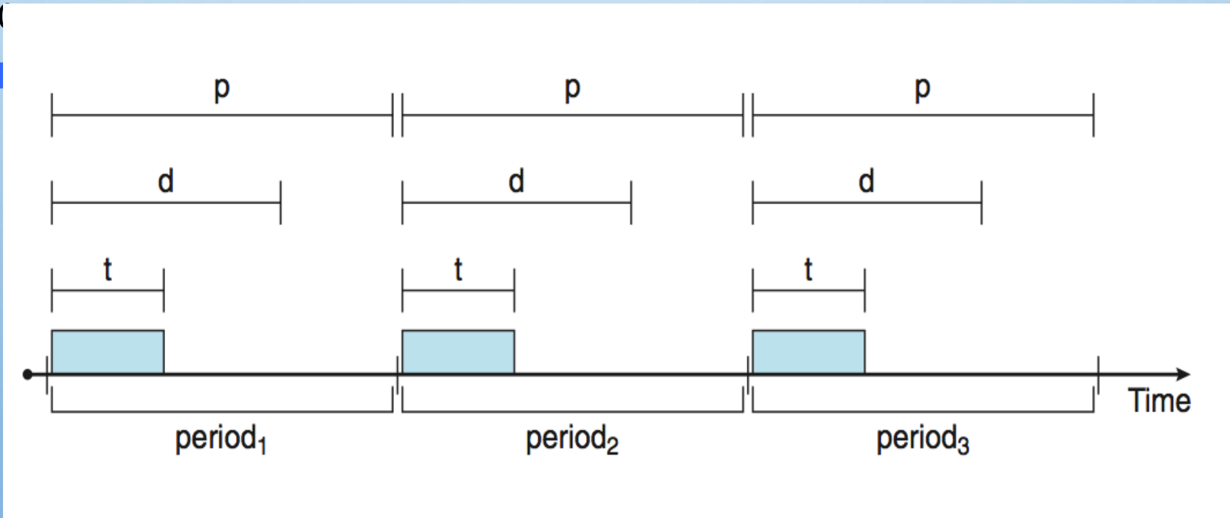
REAL-TIME CPU SCHEDULING (CONT.)

- CONFLICT PHASE OF DISPATCH LATENCY:
    1. PREEMPTION OF ANY PROCESS RUNNING IN KERNEL MODE
    2. RELEASE BY LOW-PRIORITY PROCESS OF RESOURCES NEEDED BY HIGH-PRIORITY PROCESSES



PRIORITY-BASED SCHEDULING

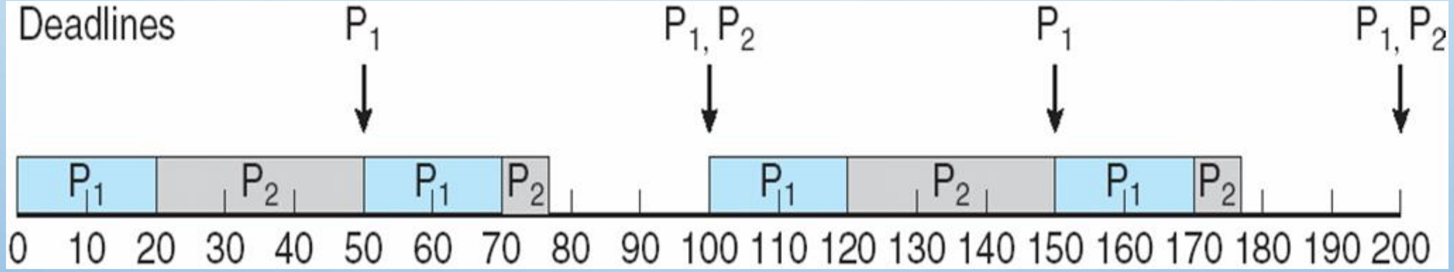
- FOR REAL-TIME SCHEDULING, SCHEDULER MUST SUPPORT PREEMPTIVE, PRIORITY-BASED SCHEDULING
  - BUT ONLY GUARANTEES SOFT REAL-TIME
- FOR HARD REAL-TIME MUST ALSO PROVIDE ABILITY TO MEET DEADLINES
- PROCESSES HAVE NEW CHARACTERISTICS: PERIODIC ONES REQUIRE CPU AT CONSTANT INTERVALS
  - HAS PROCESSING TIME *T*, DEADLINE *D*, PERIOD *P*



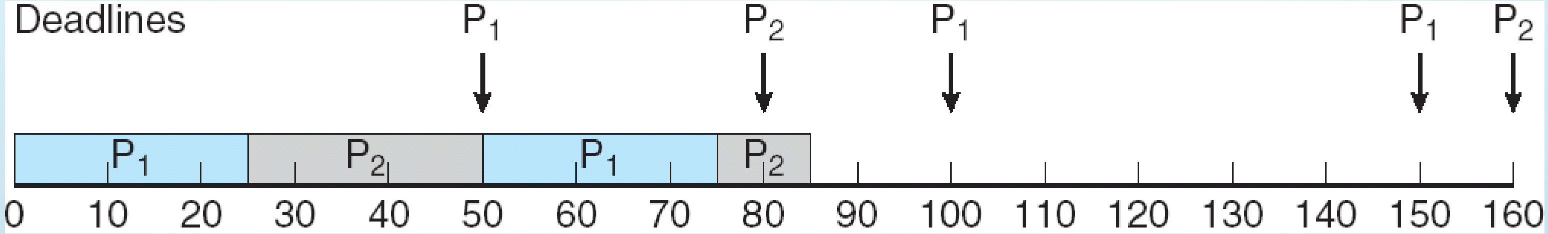
VIRTUALIZATION AND SCHEDULING

- VIRTUALIZATION SOFTWARE SCHEDULES MULTIPLE GUESTS ONTO CPU(S)
- EACH GUEST DOING ITS OWN SCHEDULING
  - NOT KNOWING IT DOESN'T OWN THE CPUS
  - CAN RESULT IN POOR RESPONSE TIME
  - CAN EFFECT TIME-OF-DAY CLOCKS IN GUESTS
- CAN UNDO GOOD SCHEDULING ALGORITHM EFFORTS OF GUESTS

RATE MONTONIC SCHEDULING

- A PRIORITY IS ASSIGNED BASED ON THE INVERSE OF ITS PERIOD
- SHORTER PERIODS = HIGHER PRIORITY;
- LONGER PERIODS = LOWER PRIORITY
- 

MISSED DEADLINES WITH RATE MONOTONIC SCHEDULING

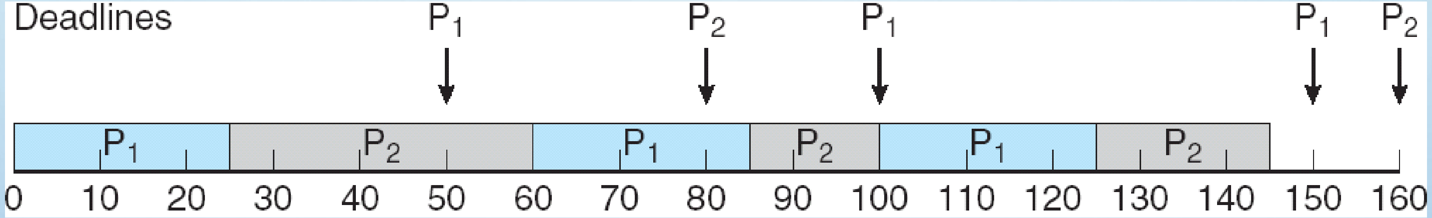


EARLIEST DEADLINE FIRST SCHEDULING (EDF)

- PRIORITIES ARE ASSIGNED ACCORDING TO DEADLINES:

  THE EARLIER THE DEADLINE, THE HIGHER THE PRIORITY;

  THE LATER THE DEADLINE, THE LOWER THE PRIORITY



PROPORTIONAL SHARE SCHEDULING

- T SHARES ARE ALLOCATED AMONG ALL PROCESSES IN THE SYSTEM
- AN APPLICATION RECEIVES N SHARES WHERE $N < T$
- THIS ENSURES EACH APPLICATION WILL RECEIVE N / T OF THE TOTAL PROCESSOR TIME

OPERATING SYSTEM EXAMPLES

- LINUX SCHEDULING
- WINDOWS SCHEDULING
- SOLARIS SCHEDULING