ECE437/CS481

M03A: CPU SCHEDULING
MORE ABOUT SCHEDULING

CHAPTER 6.4-6.8

Xiang Sun

The University of New Mexico

Dynamics with feedback

- Feedback in CPU scheduling
  - Service/burst time of a process may vary over time.
  - Processes may be terminated and create by the system over time.
  - Feedback is used to dynamic adjust the priorities of different processes.

Dynamics with feedback

- Feedback to emphasize on **fairness**

  - For N users
    - ✓ Promise every user about 1/N of CPU time

  - Dynamically bookkeeping two variables:
    - ✓ Promised time: Tp
    - ✓ Actual time used: Tu

  - Let priority function F = Tu/Tp
    - ✓ The lower the value, the higher in priority
    - ✓ case F == 1, just right, kept promise
    - ✓ case F < 1, under provisioned, the processes can request more CPU time.
    - ✓ case F > 1, over provisioned, the processes have to slow down.

Dynamics with feedback

- Feedback to emphasize on **fairness**
  - Fair enough?
    - ✓ For those didn't get a fair share of CPU time → priority raised
    - ✓ If new I/O-intensive processes (which require low CPU time/service time) regularly enter the system, computation-intensive processes potentially get starved, increasing their turnaround time.
    - ✓ Achieving fairness is difficult.

Dynamics with feedback

- Feedback to emphasize on **aging**
  - Favor a user who spent more waiting time at Ready queue.
  - Dynamically bookkeeping two variables:
    - ✓ Total waiting time: Tw
    - ✓ Actual time used: Tu
  - Let priority function F = Tu/Tw
    - ✓ The lower the value, the higher in priority

Classification of processes

By the nature of requirement

- **Foreground processes**—the system/shell may wait for the processes to finish
  - ✓ I/O-intensive processes
- **Background processes**-- the system/shell does not have to wait for the processes to finish before it can run more processes
  - ✓ computation-intensive processes

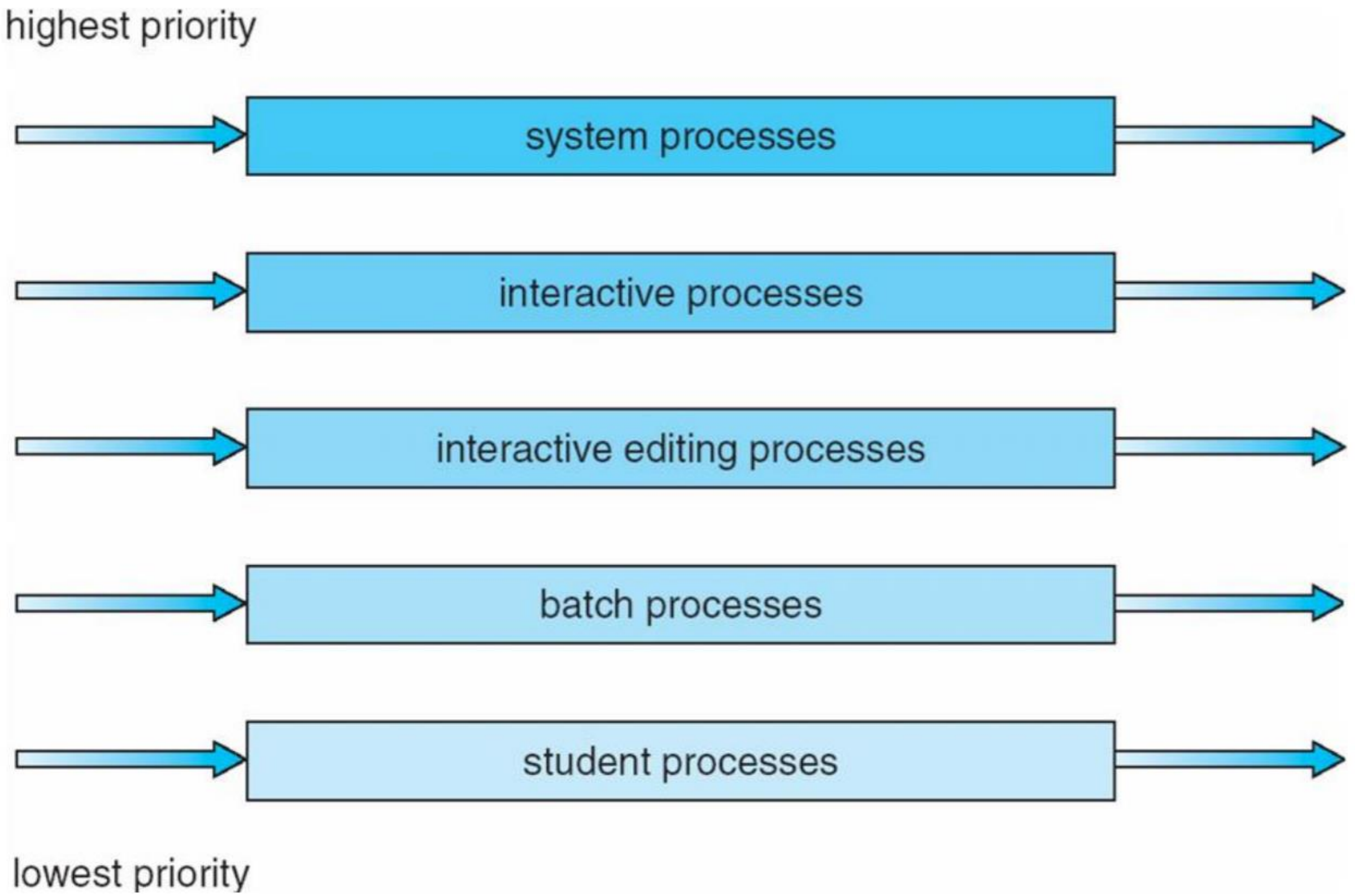
By run-time characteristics

- System process
  - ✓ Accomplishing system task: e.g., paging
  - ✓ A user process executing kernel task via system call, e.g., I/O request
- User processes

Multilevel Queues

- Ready queue is partitioned into separate queues
- Each queue has its own scheduling algorithm
  - Typical example
    - ✓ foreground queue– RR
    - ✓ background queue– FCFS
- Scheduling must be done among these 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 among its processes; e.g., 80% to foreground in RR, 20% to background in FCFS.

Multilevel Queues



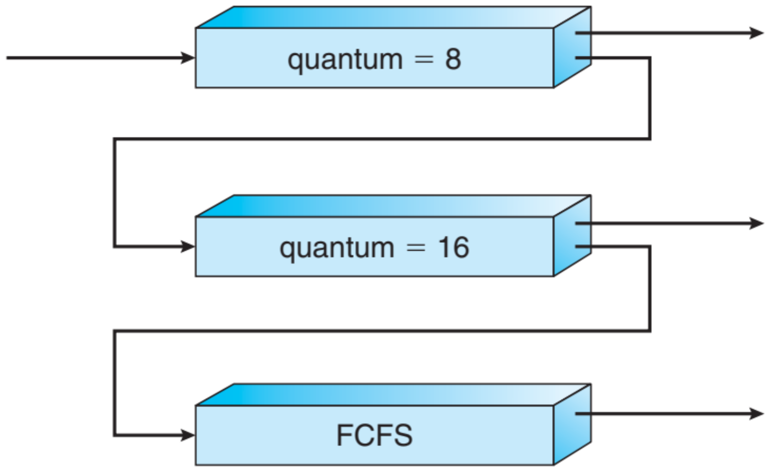
Multilevel Queues

- Multilevel queues + feedback
  - Process can move among queues based on their feedbacks (e.g., waiting time or burst time).
- Design & implementation of multilevel queues
  - # of queues and their related scheduling strategies (i.e., intra-queue scheduling).
  - Scheduling strategies among different queues (i.e., inter-queue scheduling).
  - Method used to adjust the process among queues (feedback control).
  - Initially process deployment (i.e., determine which queue a process will enter initially)

Multilevel Feedback Queues

An example of multilevel feedback queues

- There are three queues, and their intra-queue scheduling are:
  - ✓ Q0 – RR with time quantum 8 ms
  - ✓ Q1 – RR with time quantum 16 ms
  - ✓ Q2 – FCFS



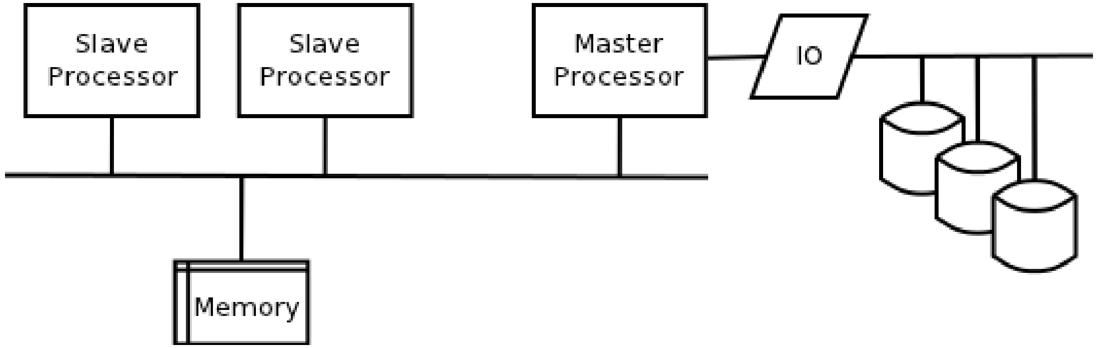
- The inter-queue scheduling is
  - ✓ Fix priority scheduling: Q0—high, Q1—medium, Q2—low.
- The feedback control are designed as follows:
  - ✓ A new process first enters queue Q0
  - ✓ If the process does not finish in 8 ms, the process is moved to Q1; otherwise, stay in Q1.
  - ✓ Once the process moves to Q1, it will receive 16 ms in the next cycle.
  - ✓ If the process still does not finish in 16 ms, it is preempted and moved to Q2.
- I/O-intensive process will normally end up on high priority queue (Q0), and computational-intensive process will normally end up on low priority (Q2).

Multiprocessor Scheduling

- What is multiprocessor scheduling
  - Given a set of runnable processes/threads, and a set of CPUs, assign processes/threads to CPUs
- Same metrics as uniprocessor scheduling
  - Fairness, efficiency, throughput, response time…
- But also new considerations
  - Relationship among processors
  - Load balancing
  - Processor affinity—keep a process running on the same CPU

Asymmetric Multiprocessor Processing (AMP)

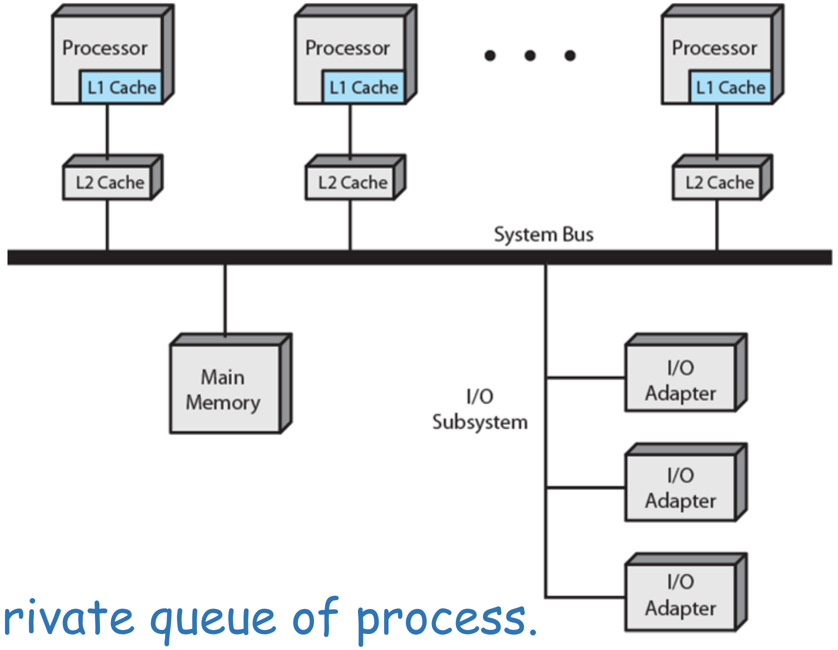
❑ Solution 1: Asymmetric multiprocessor processing (Centralized processing)



- Two types of processors (one master processors and a number of slave process). One ready queue.
- Master processor makes scheduling decision and handles I/O operations.
- Slave processors executes assigned processes. The master processor would balance the workload among slave processes
- If a master processor fails, a slave processor become the master processor. If a slave processor fails, its allocated processes are switched to other slave processors.

Symmetric Multiprocessor Processing (SMP)

- Solution 2: Symmetric multiprocessor processing (Distributed processing)
  - Each processors is self-scheduling.
  - All processes may be in a common ready queue (global ready queue) or each processor may have its own private queue for ready processes.
  - Balancing the workload among processors is necessary to maximize the performance of the system.
    - ✓ Load balancing is necessary only if each processor has its own private queue of process.
    - ✓ Two general approaches to achieve load balancing: Push migration and Pull migration.
    - ✓ Push migration: A surveillance task periodically checks the workload on each processor and moves processes from processors with high load to processors with low load if needed.
    - ✓ Pull migration: A processor's scheduler notices its queue is empty (or shorter than a threshold), and tries to fetch a process from another processor.



Processor affinity in SMP

Recall

- Processors share main memory.
- Processors have their own local cache memories.
- Recently accessed data are stored in local cache memories in order to speed up data retrieval.

Process affinity

- Try to keep the process running in the same processor.
- Benefit of process affinity: quicker to restart process on same processor since the cache may already contain needed data.
- Two types of processor affinity:
  - ✓ Soft Affinity: a scheduler has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so.
  - ✓ Hard Affinity: some systems such as Linux have a system call to specify that a process shall execute on a specific processor.

```
unsigned long mask = 7; /* processors 0, 1, and 2 */
unsigned int len = sizeof(mask);
if (sched_setaffinity(0, len, &mask) < 0) {
    perror("sched_setaffinity");
}
```

Realtime System

- In a realtime system, the correctness of the system depends on
  - The logical result of the computation, AND.
  - The time when the results are produced.
- Deadline is associated a particular task
  - hard deadline: required to complete a critical task within a guaranteed amount of time
  - soft deadline: the deadline is desirable but not mandatory; still make sense to schedule and complete the task even if it has passed its deadline.
- Two types of processes in realtime system, i.e., Periodic and aperiodic process
  - Periodic processes: arriving at fixed frequency, can be characterized by 3 parameters (C,D,T) where C = service/burst time, D = deadline, T = period (e.g. 20ms, or 50HZ). Periodic processes are called Time-driven processes, their activations are generated by timers.
  - Aperiodic processes : all processes that are not periodic, also known as event-driven, their activations may be generated by external interrupts.

Realtime Scheduling

❑ Deadline scheduling

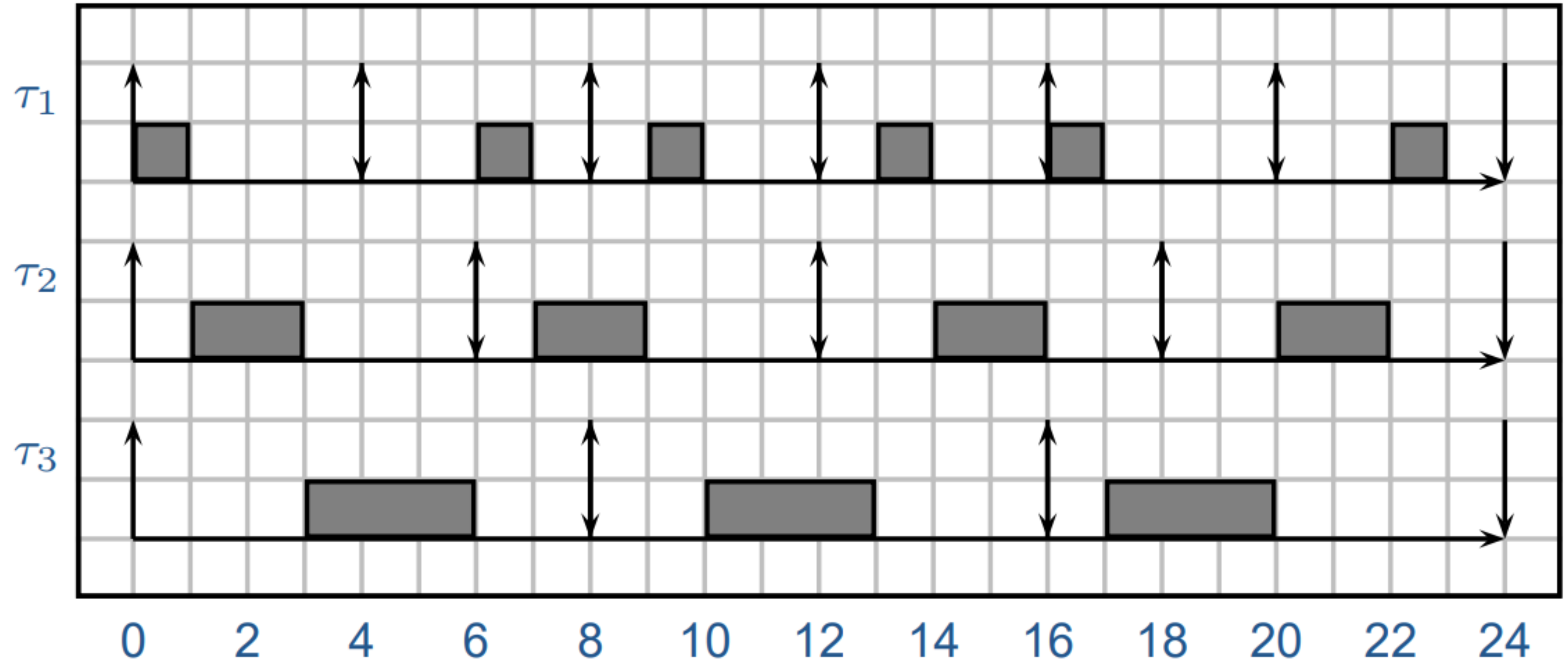
- Need preemptive strategy & priority function
- The service time of each task is known in advance.
- Complete task neither too early nor too late.
- Find a schedule for all the tasks so that each meets its deadline.
  - ✓ What if we can't schedule all task?
  - ✓ What if the run-time is not constant but has a known probability distribution?
- A popular algorithm: EDF (Earliest Deadline First)

Earliest Deadline First (EDF) Scheduling

- Scheduler selects a job with EDF
  - ✓ The highest priority job is the one with the earliest absolute deadline;
  - ✓ If two jobs have the same deadline, chose one of the two at random;
  - ✓ The priority of a job is dynamic since the job's deadline changes over time;
  - ✓ Decision mode: preemption (by default).

Earliest Deadline First (EDF) Scheduling

- Example: scheduling with EDF
  - ✓ Three processes τ_1, τ_2, τ_3 are initially in the ready queue.
  - ✓ τ_1=(1,4), which indicates that the burst time of τ_1 is 1 time unit, and τ_1 will be in the ready queue after each 4 time unit; the relative deadline of τ_1 is also 4 time unit. Accordingly, τ_2=(2,6), and τ_3=(3,8).



$$CPU\ utilization = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$$

Earliest Deadline First (EDF) Scheduling

- **Theorem**: Given a job set of periodic or sporadic jobs, with relative deadlines equal to periods, the job set is schedulable by EDF iff

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

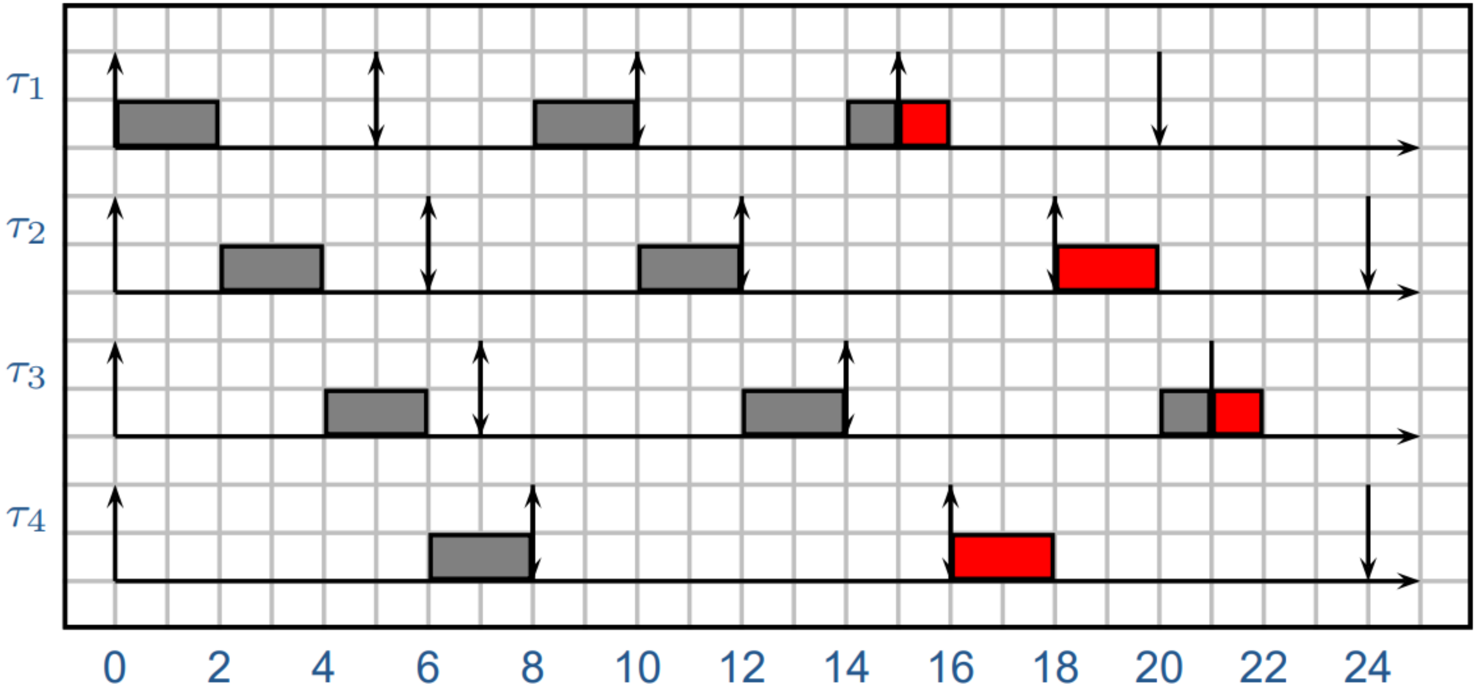
where C_i is the burst time of job i, is the T_i relative deadline of job i, N is the total number of jobs of the job set, and U is the CPU utilization.

- Lemma: EDF is an optimal algorithm, in the sense that if a job set is schedulable, then it is schedulable by EDF.
  - ✓ If U>1, no algorithm can successfully schedule the job set;
  - ✓ If U≤1, EDF can always provide a feasible schedule.

Earliest Deadline First (EDF) Scheduling

- **Domino effect** with EDF

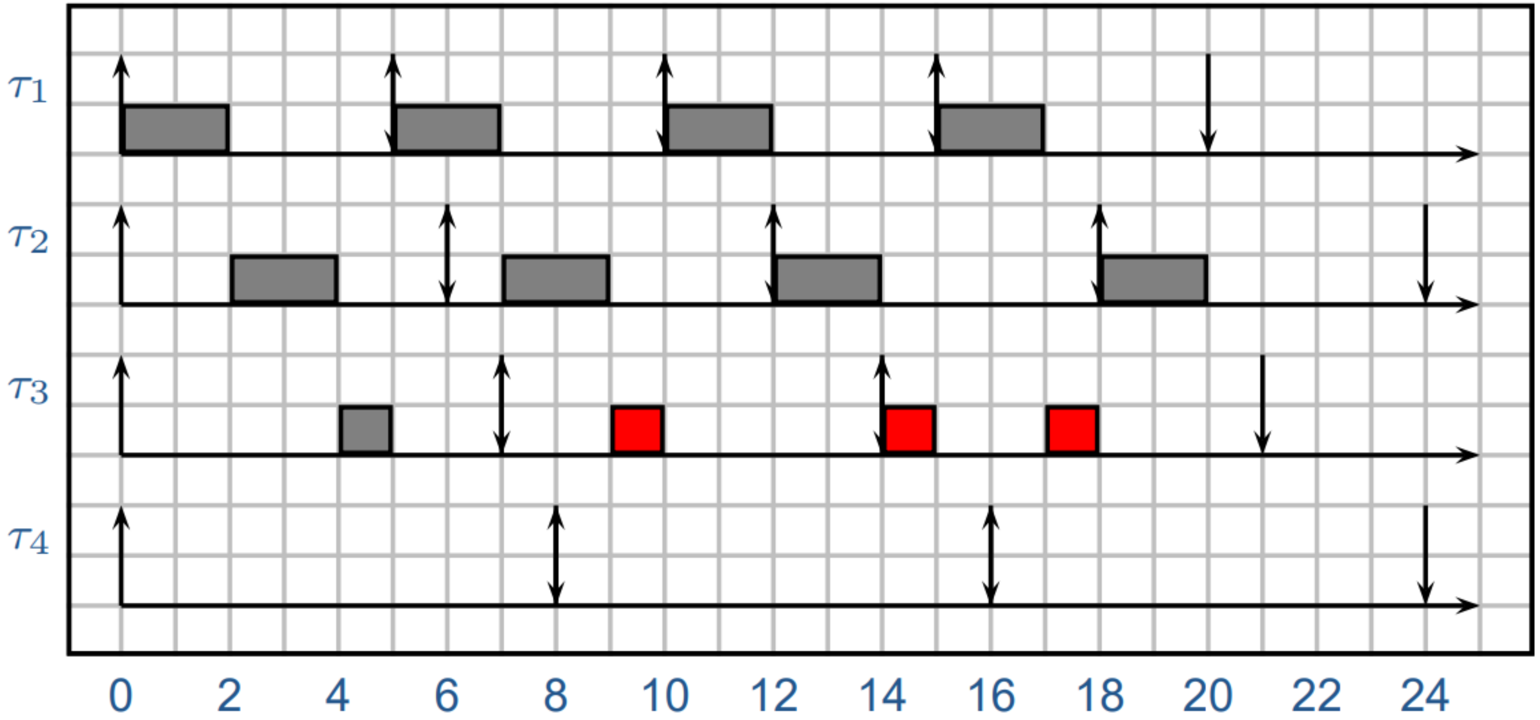
  - If U>1 (i.e., the job is NOT schedulable), we have the domino effect with EDF: it means that all tasks miss their deadlines.
  - An example of domino effect:
    - ✓ Four processes: τ_1, τ_2, τ_3, and τ_4.
    - ✓ τ_1= (2,5), τ_2= (2,6), τ_1= (2,7), τ_1= (2,8)



❑ Rate Monotonic (RM) Scheduling

- The highest priority job is the one with the earliest relative deadline/smallest period;
- If two jobs have the same deadline, chose one of the two at random;
- The priority of a job is static;
- Decision mode: preemption (by default).

  - ✓ τ_1 and τ_2 never miss their deadlines;
  - ✓ τ_3 misses a lot of deadline;
  - ✓ τ_4 is not executed!



❑ Rate Monotonic (RM) Scheduling

- RM cannot guarantee all the jobs in the job set meet their deadlines, even if the job set is schedulable (i.e., U≤1).
- *τ1=(1,4), τ2=(2,6), and τ3=(3,8).* $U = \frac{1}{4} + \frac{2}{6} + \frac{3}{8} = \frac{23}{24}$