ECE437/CS481

M03B: CPU SCHEDULING
MORE ABOUT SCHEDULING

CHAPTER 6.4-6.8

Xiang Sun

The University of New Mexico

Feedback in CPU scheduling

- Remaining time of a process may vary over time.
- Processes may be terminated and created by the system over time.
- Feedback is used to dynamic adjust the priorities of different processes.

Dynamics with feedback

- Feedback to emphasize on **service time fairness**
  - Dynamically bookkeeping two variables:
    - ✓ Required/Promised CPU time: Tp
    - ✓ Actual CPU time used: Tu
  - Let priority function F = Tp/Tu
    - ✓ The larger the value, the higher in priority
    - ✓ case F == 1, just right, kept promise.
    - ✓ case F > 1, under provisioned, the process can have more CPU time.
    - ✓ case F < 1, over provisioned, the process has to slow down.

Dynamics with feedback

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

Classification of processes

❑ By the nature of requirement

- **Foreground process**
  - ✓ Users can directly interact with a foreground process, for example, via a terminal/shell.
  - ✓ Executing a foreground process does not disable the user to execute other background processes via the same terminal until it terminates.

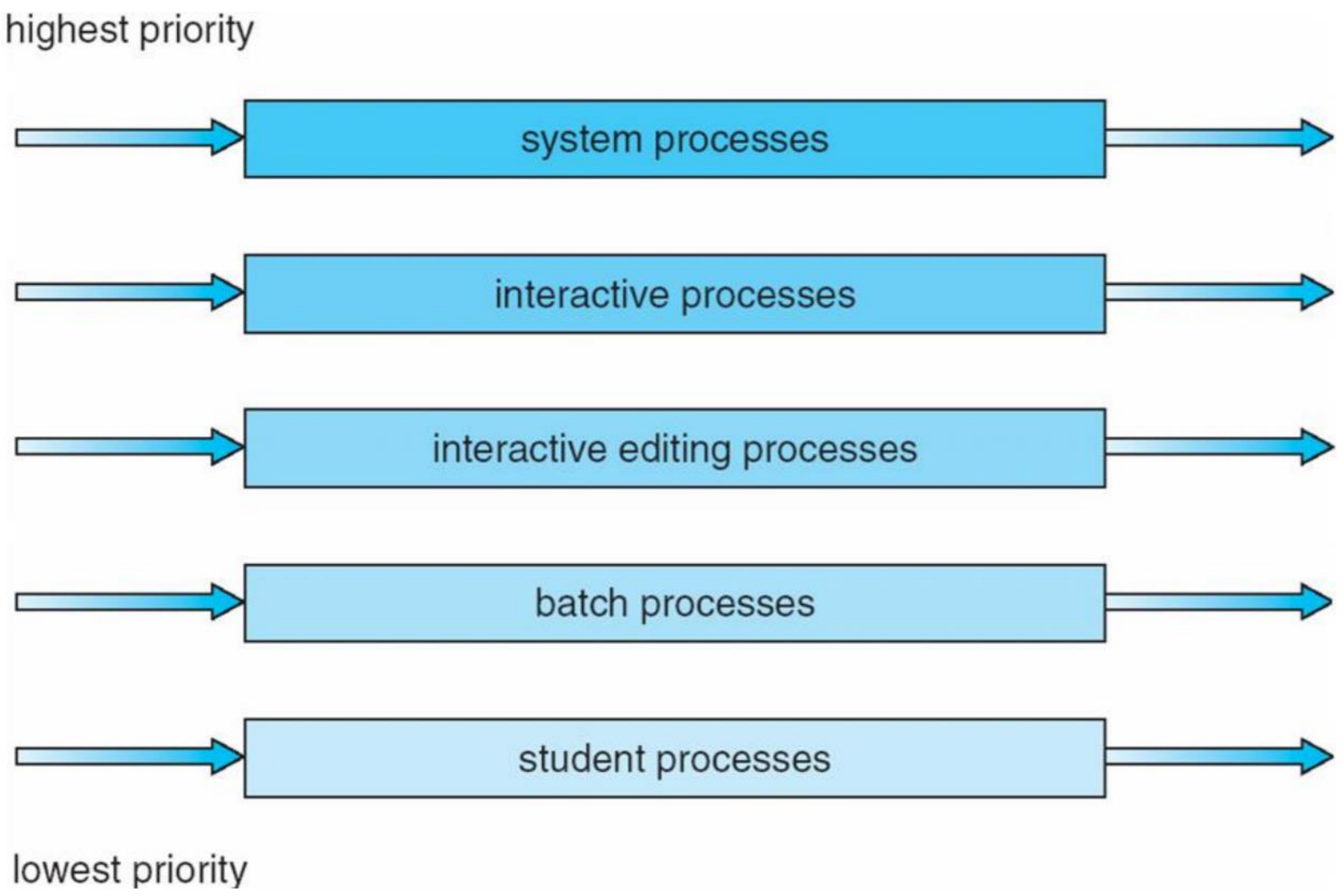
- **Background process**
  - ✓ Users cannot directly interact with a background process.
  - ✓ Executing a background process does not disable the execution of other processes.

- Example:
  - Run "yes &" create a yes background process. You cannot stop it by "ctrl+z".
  - Enter "fg" to bring yes process into foreground. Then, you can stop it by "ctrl+z".

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 scheduling- each queue gets a certain amount of CPU time, e.g., 80% to foreground processes in RR, 20% to background processes in FCFS.

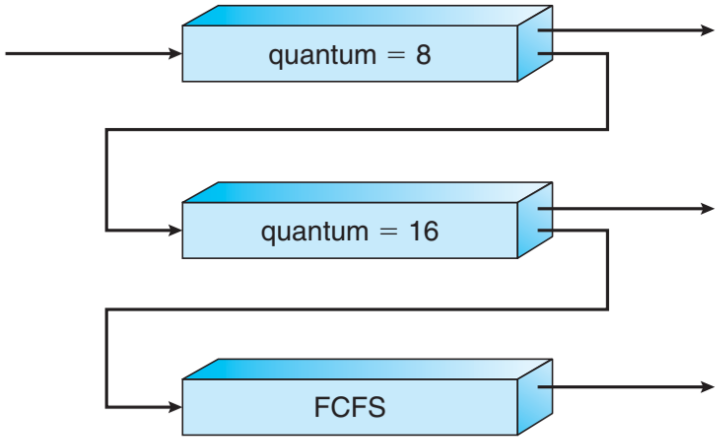
Multilevel Queues



Feedback-enabled Multilevel Queues

- Multilevel queues + feedback
  - Process can move among queues based on their feedbacks (e.g., waiting time or service time).
- In the feedback-enabled multilevel scheduling method
  - # 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 processes among queues (feedback control).
  - Initially process placement.

- An example of feedback-enabled multilevel scheduling method
  - 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 initial process placement and feedback control are designed as follows:
    - ✓ A new ready process first enters Q0
    - ✓ If a process does not stop/block in 8 ms, the process is moved to Q1; otherwise, it stays in Q0.
    - ✓ Once the process moves to Q1, it will receive 16 ms in the next cycle.
    - ✓ If the process still does not stop/block in 16 ms, it is moved to Q2.



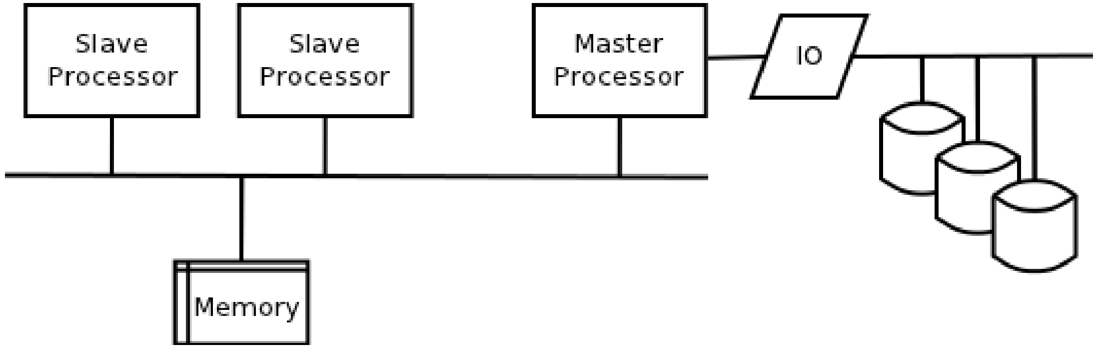
  - I/O-intensive process will normally end up on high priority queue (Q0), and computational-intensive process will normally end up on low priority (Q2).

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
  - Load balancing
  - Processor affinity—keep a process running on the same core

Asymmetric Multiprocessor Processing (AMP)

❑ Solution 1: Asymmetric multiprocessor processing (Centralized processing)



- Two types of processors (one master processor and a number of slave processors). One ready queue(s).

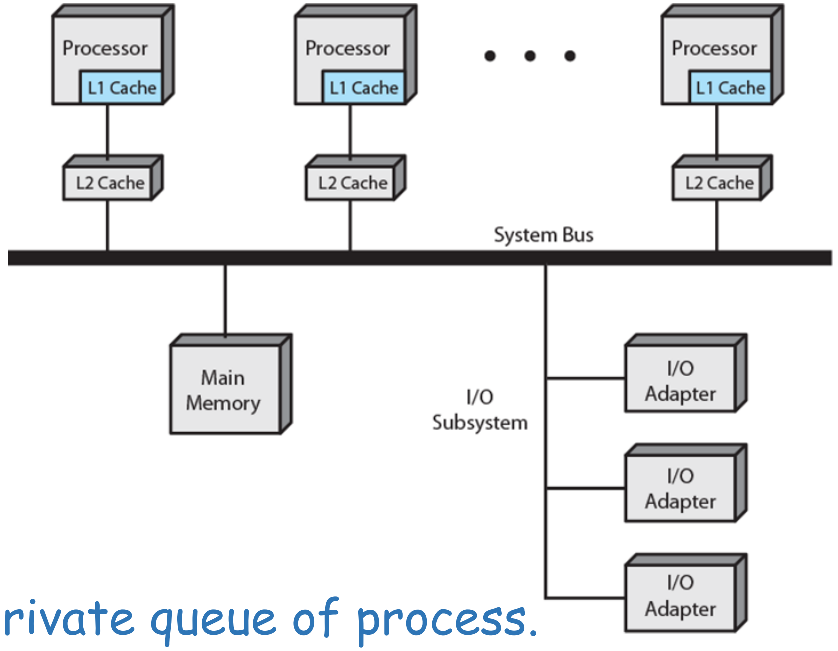
- Master processor makes scheduling decision (to balance the workload among slave processors) and handles I/O requests.

- Slave processors simply execute assigned 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 processor 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 less than a predefined threshold), and tries to fetch a process from another processor's queue.

Processor affinity in SMP

- In the multiprocessor architecture
  - 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 existing processes 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 methods to achieve 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 real-time system, the performance of the system depends on
  - The logical result of the computation.
  - 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.

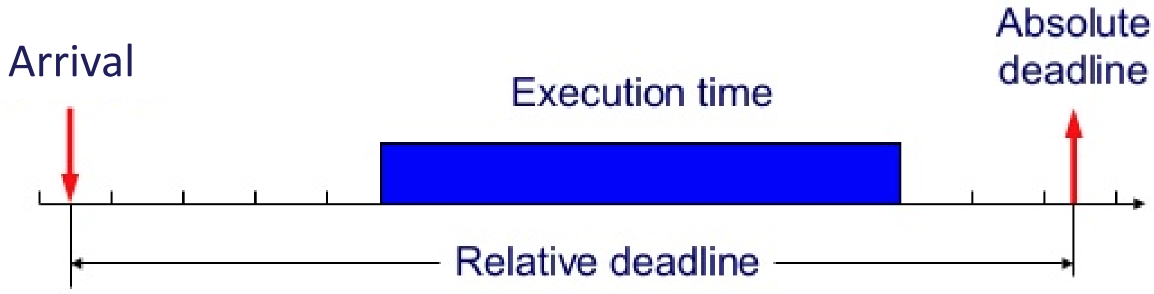
- Two types of processes in real-time system, i.e., Periodic and aperiodic process
  - Periodic processes: arriving at fixed frequency, can be characterized by 3 parameters (C,D,T) where C = service/burst time, D = relative deadline, T = period (e.g., 20ms, or 50HZ). Periodic processes are called Time-driven 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

- Realtime scheduling
  - Need preemptive strategy & priority function
  - The service time of each process is known in advance.
  - Find a schedule for all the processes so that each meets its deadline.
  - A popular algorithm: EDF (Earliest Deadline First)

Earliest Deadline First (EDF) Scheduling

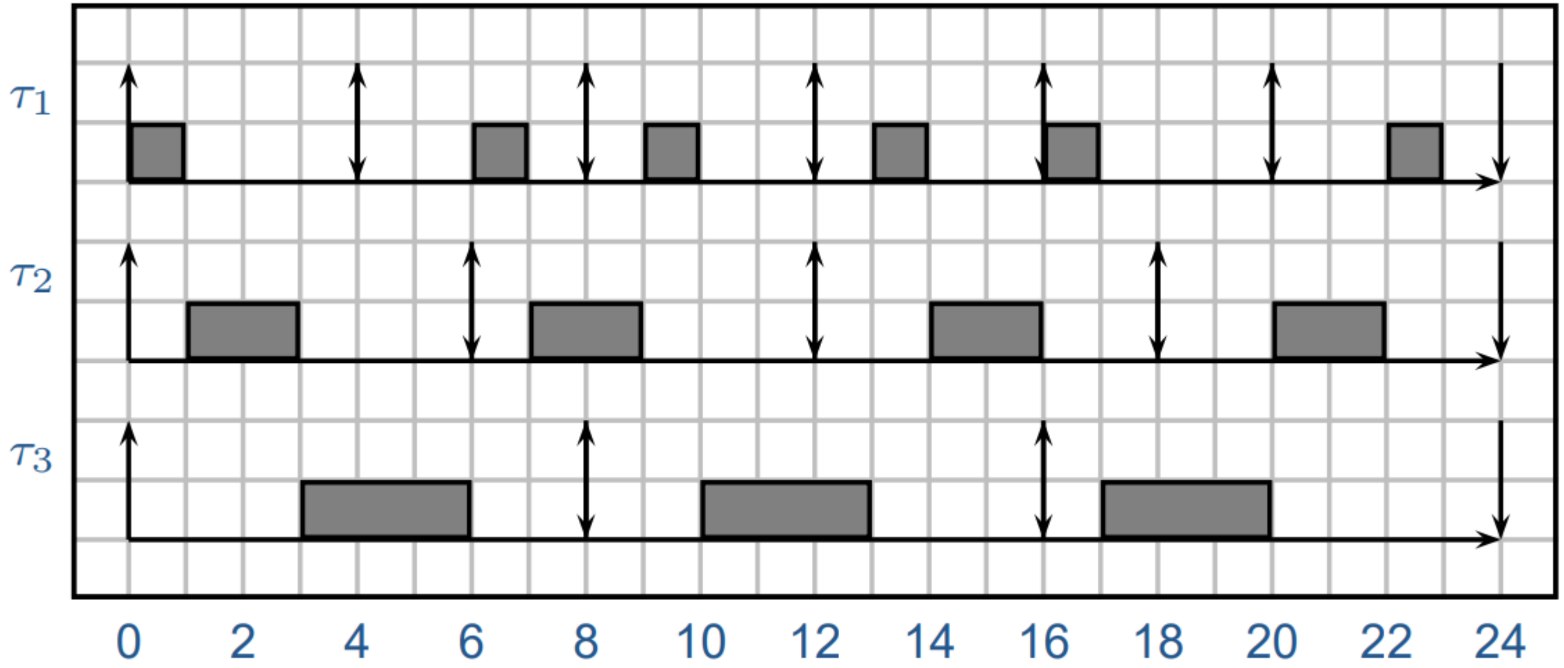
- Scheduler selects a job (e.g., a process) with EDF
  - The highest priority job is the one with the earliest absolute deadline;
    - ✓ Absolute deadline vs relative deadline



  -
  - If two jobs (one is the executed by the CPU and the other is in the ready queue) have the same absolute deadline, select the running job.
  - Decision mode: preemption.

Earliest Deadline First (EDF) Scheduling

- Example: scheduling with EDF
  - ✓ Three types of processes τ_1, τ_2, τ_3 are initially in the ready queue.
  - ✓ $\tau_1=(1,4,4)$, which indicates that the service time of τ_1 is 1 time unit, and τ_1 will be in the ready queue after each 4 units; the relative deadline of τ_1 is also 4 units. Accordingly, $\tau_2=(2,6,6)$, and $\tau_3=(3,8,8)$.



$$CPU\ utilization = \frac{23}{24}$$

Earliest Deadline First (EDF) Scheduling

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

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

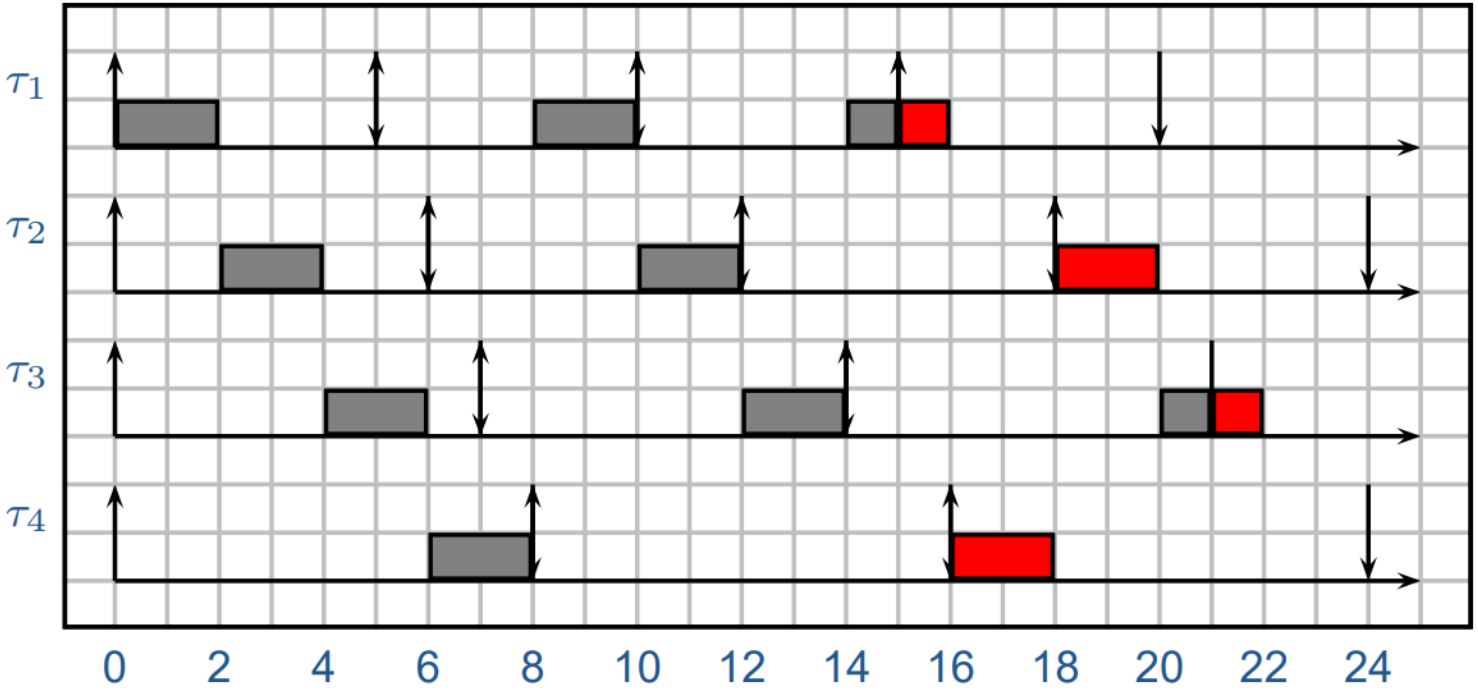
where C_i is the service time (burst time) of job i, T_i is the relative deadline of job i, N is the total number of jobs in the job set, and U is the CPU utilization.

- **Lemma**: EDF is an optimal algorithm, in the sense that if a job set is schedulable, then it is always 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

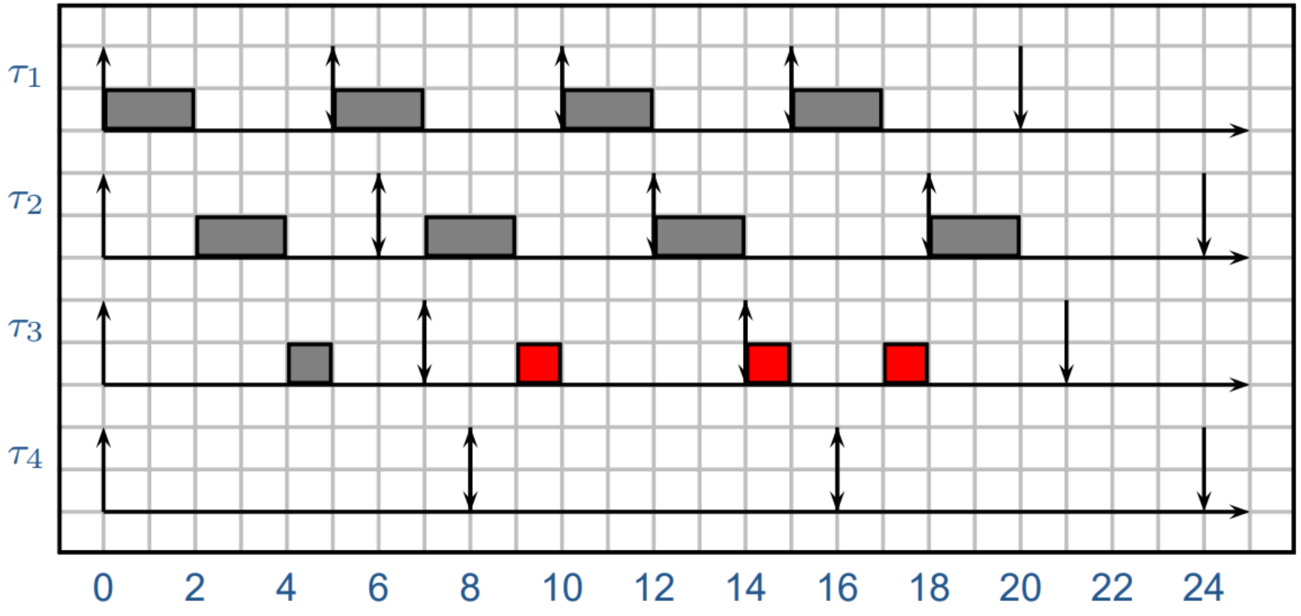
Domino effect with EDF

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



❑ Rate Monotonic (RM) Scheduling

- ➢ The highest priority job is the one with the earliest relative deadline;
- ➢ Decision mode: preemption (by default).
- ➢ Four processes: τ_1, τ_2, τ_3, and τ_4, where τ_1= (2,5,5), τ_2= (2,6,6), τ_3= (2,7,7), τ_4= (2,8,8)
  - ✓ τ_1 and τ_2 never miss their deadlines;
  - ✓ τ_3 misses many deadlines;
  - ✓ τ_4 is not executed!



❑ Rate Monotonic (RM) Scheduling

- RM cannot guarantee all the jobs meet their deadlines, even if the job set is schedulable (i.e., U≤1).
- $\tau_1=(1,4,4)$, $\tau_2=(2,6,6)$, and $\tau_3=(3,8,8)$. $U=\frac{1}{4}+\frac{2}{6}+\frac{3}{8}=\frac{23}{24}$