中国科学技术大学
University of Science and Technology of China

011174.01: Operating System
操作系统原理与设计

Chapter 4 之 real-time CPU scheduling

陈香兰(xlanchen@ustc.edu.cn)
高能效智能计算实验室，CS，USTC @ 合肥
嵌入式系统实验室，CS，USTC @ 苏州

温馨提示：

为了您和他人的工作学习，

请在课堂上**关机或静音**。

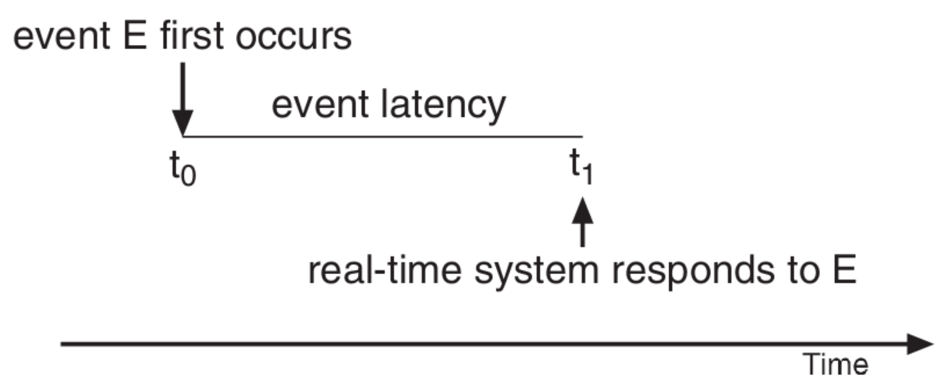
不要在课堂上**接打电话**。

Outline

- Some special issues for real-time CPU scheduling
  - Performance: How to minimize the latencies
    - Event-driven & event latency
    - RTOS
      - Priority Scheduling with preemption
      - Short interrupt latency & short dispatch latency
  - Schedulability
    - Process modeling
    - Admission-control
    - Rate-Monotonic Scheduling
    - Earliest-Deadline-First Scheduling
    - Proportional Share Scheduling
  - Example: POSIX Real-Time Scheduling

Minimizing Latency

- **Soft real-time computing**: requires that critical processes receive priority over less fortunate ones
- **Hard real-time systems**: requires to complete a critical task within a guaranteed amount of time
- **Event-driven** VS. interrupt-driven
  - The system is typically waiting for an event in real time to occur.
  - Events may arise either in software or in hardware
- **Event latency**: the amount of time from an event occurring to when it is serviced



Minimizing Latency

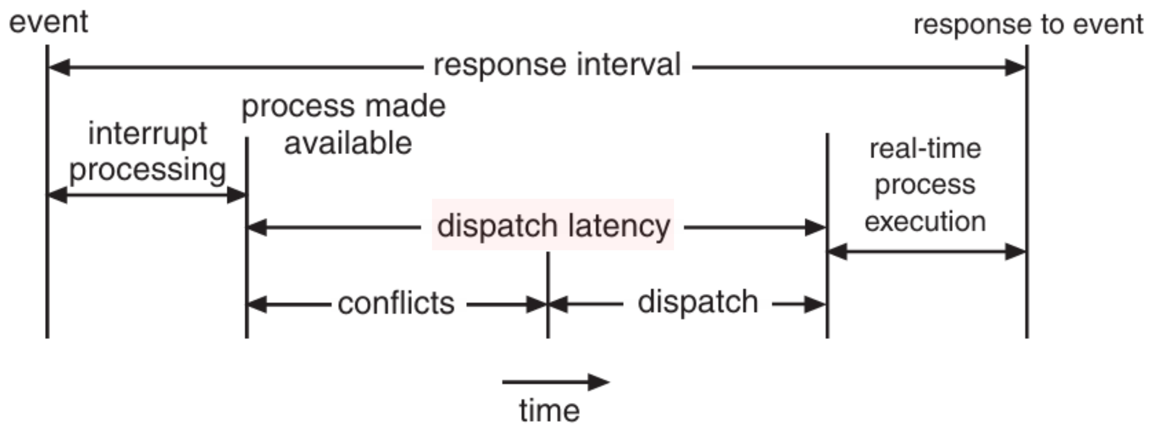
- Two types of latencies affect the performance of RTS
  - **Interrupt latency**: Time from the arrival of an interrupt at the CPU to the start of the ISR, should be minimized; For Hard RTS: should be bounded

    0. Interrupt may be disabled, the time should be very short and bounded. When interrupt is enabled:
    1. Complete the current instruction
    2. Determine the interrupt type
    3. Save the state of the current process
    4. Start to execute the ISR

  - **Dispatch latency**

Minimizing Latency

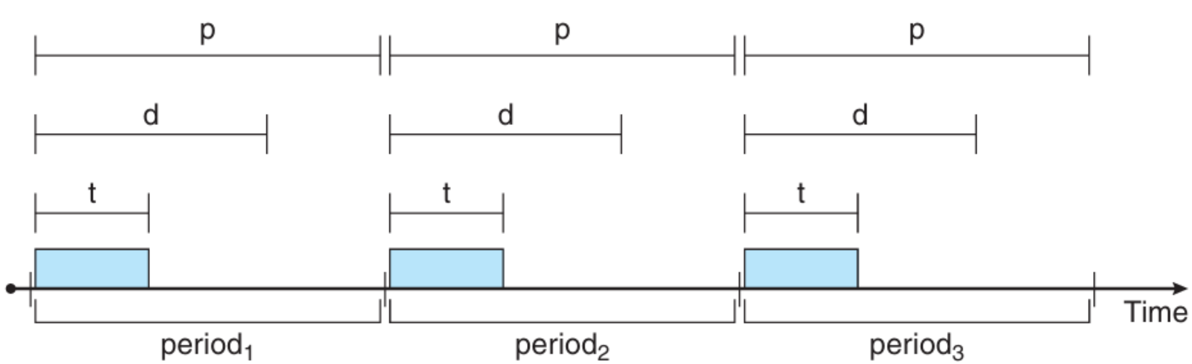
- **Dispatch latency:** The amount of time required for the scheduling dispatcher to stop one process and start another



- Approaches for short dispatch latency
  - Preemption
    - preemption point (抢占点) in system calls with long period
    - preemptible kernel
  - priority inversion
    - priority-inheritance protocol
    - priority-ceiling protocol

Priority-Based Scheduling

- The scheduler for a RTOS must support a **priority based algorithm with preemption**.
  - Real-time processes: highest priority levels
    - Example: Windows [16-32]
  - Only guarantees soft real-time functionality
  - For hard real-time with deadline requirements, need additional scheduling feature.
- Usually, the processes are periodic tasks



Mostly, $t \le d \le p$

Rate(速率) = 1/p

- Schedulability: admission-control algorithm
  - Admit(=guarantee)? Or, reject?

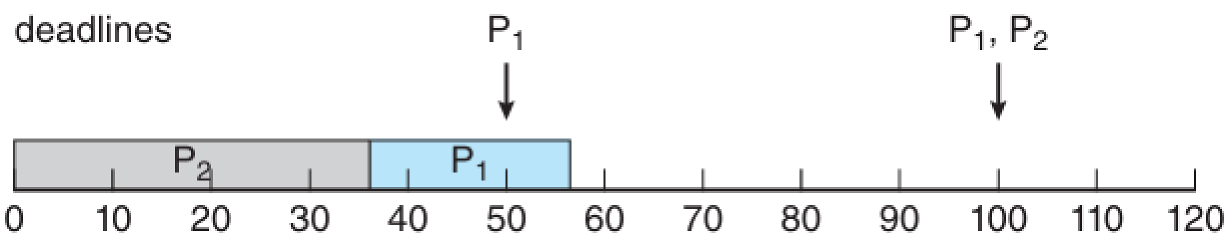
RM scheduling

University of Science and Technology of China

- Example1: P1: $t_1 = 20, p_1 = d_1 = 50$; P2: $t_2 = 35, p_2 = d_2 = 100$

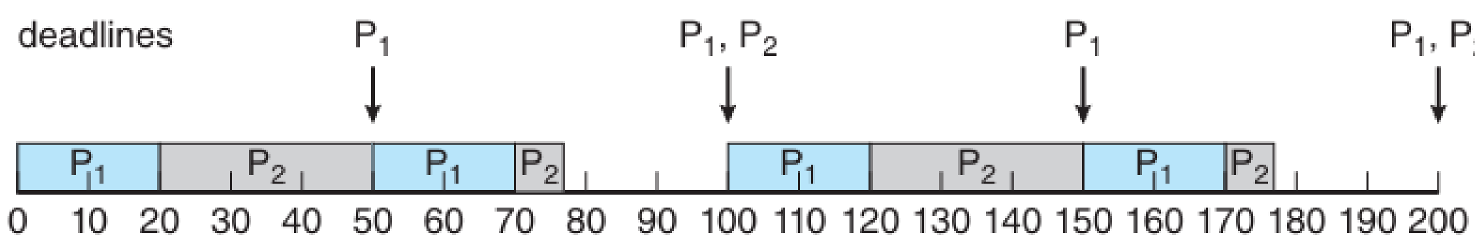
CPU utilization
$$u_1 = {}^{t_1}/_{p_1} = {}^{20}/_{50} = 0.4$$
$$u_2 = {}^{t_2}/_{p_2} = {}^{35}/_{100} = 0.35$$
$$u_{total1} = u_1 + u_2 = 0.75$$

Scheduling1



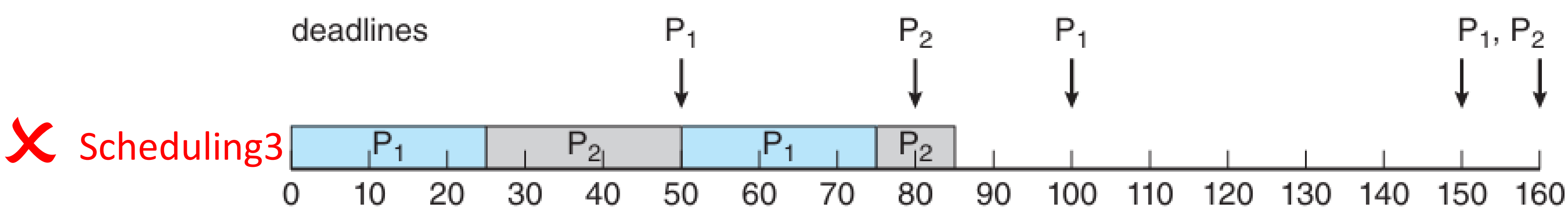
- Rate-Monotonic (RM) Scheduling
  - using a static priority policy with preemption
  - **Priority**: the shorter the period, the higher the priority

Scheduling2



RM scheduling

- Rate-monotonic scheduling is considered **optimal** in that
  - If a set of processes cannot be scheduled by this algorithm, it cannot be scheduled by any other algorithm that assigns **static** priorities.
- Example2: $P1: t_1 = 25, p_1 = d_1 = 50; P2: t_2 = 35, p_2 = d_2 = 80$



CPU utilization $u_{total2} = u_1 + u_2 = 25/50 + 35/80 = 0.94$

RM scheduling

- Limitation:
  - CPU utilization is bounded, and it is not always possible to maximize CPU resources fully.
- The worst-case CPU utilization for scheduling N processes is

$$u_{\text{wc, total}} = N\left(2^{1/N} - 1\right)$$

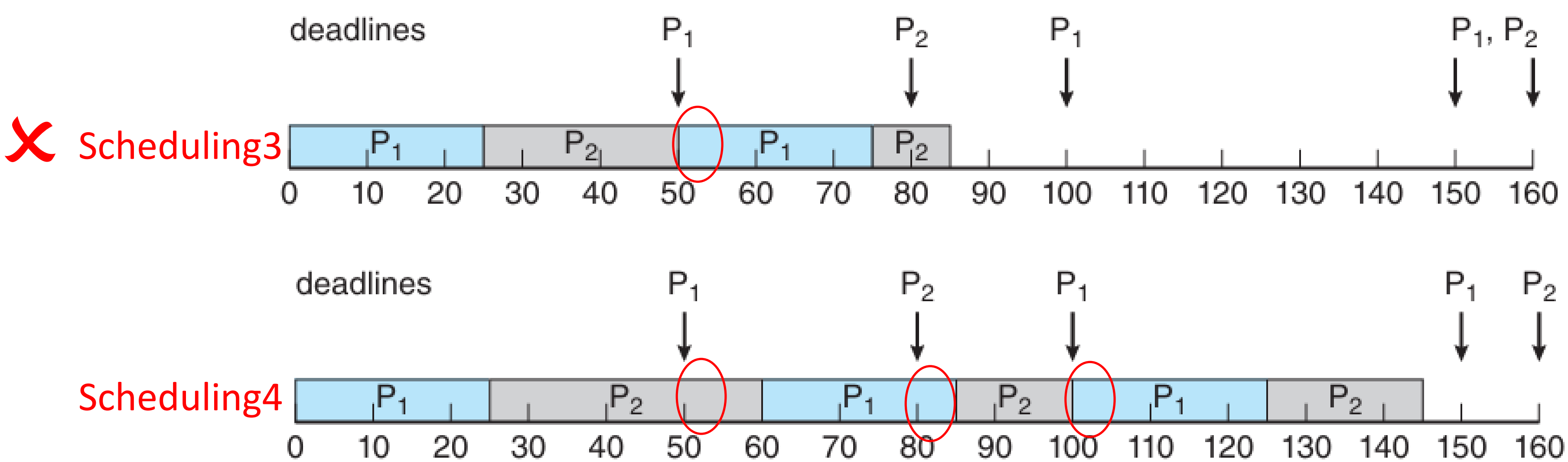
$$u_{total1} = 0.75$$
$$u_{total2} = 0.94$$

  - $N = 1$: $u_{\text{wc, total}} = 100\%$; $N = 2$: $u_{\text{wc, total}} \approx 83\%$
  - $N \to \infty$: $u_{\text{wc, total}} \approx 69\%$

EDF scheduling

中国科学技术大学
University of Science and Technology of China

- Earliest-Deadline-First (EDF) Scheduling
  - Using dynamically priority according to deadline
    - May have to adjust the priorities when a new process became runnable to reflect its deadline
  - Priority: the earlier the deadline, the higher the priority
- Recall example2: $P1: t_1 = 25, p_1 = d_1 = 50;\ P2: t_2 = 35, p_2 = d_2 = 80$



EDF scheduling

- EDF VS. RM
  - Periodic?
  - Constant t?
  - Only deadline!
- The EDF scheduling is theoretically optimal
  - theoretically, guarantee & $u_{total} = 100\%$
  - But, practically, however, it is impossible due to the cost of context switching between processes and interrupt handling.

Proportional Share Scheduling

- Proportional share scheduler:
  - Total processor time: T shares
  - If an app can receive N shares, it will have N/T of the total processor time.
  - Example: T = 100, N1=50, N2=15, N3=20
- **Admission-control policy:**
  - To guarantee an app receives its allocated share
  - If P4 requested 30 shares, it will be denied.
    - 100-(50+15+20)=15 < 30

POSIX Real-Time Scheduling

- The POSIX standard also provides extensions for real-time computing— POSIX.1b.
- POSIX defines two scheduling classes for real-time threads: (with equal priority)
  - SCHED_FIFO
  - SCHED_RR
- Related POSIX API:
  - pthread_attr_getschedpolicy(pthread attr t *attr, int *policy)
  - pthread_attr_setschedpolicy(pthread attr t *attr, int policy)

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

int main(int argc, char *argv[]) {
  int i, policy;
  pthread_t tid[NUM THREADS];
  pthread_attr_t attr;

  /* get the default attributes */
  pthread_attr_init(&attr);

  /* get the current scheduling policy */
  if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
     fprintf(stderr, "Unable to get policy.\n");
  else {
     if (policy == SCHED_OTHER)       printf("SCHED_OTHER\n");
     else if (policy == SCHED_RR)     printf("SCHED_RR\n");
     else if (policy == SCHED_FIFO)   printf("SCHED_FIFO\n");
  }
```

```
 /* set the scheduling policy- FIFO, RR, or OTHER */
 if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
    fprintf(stderr, "Unable to set policy.\n");

 /* create the threads */
 for (i = 0; i < NUM_THREADS; i++)
    pthread_create(&tid[i], &attr, runner, NULL);

 /* now join on each thread */
 for (i = 0; i < NUM_THREADS; i++)
    pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param) {
 /* do some work ... */
 pthread_exit(0);
}
```

小结

- Real-time CPU scheduling
  - Performance: How to minimize the latencies
    - Event-driven & event latency
    - RTOS
      - Priority Scheduling with preemption
      - Short interrupt latency & short dispatch latency
  - Schedulability
    - Process modeling
    - Admission-control
    - Rate-Monotonic Scheduling
    - Earliest-Deadline-First Scheduling
    - Proportional Share Scheduling
  - Example: POSIX Real-Time Scheduling

Q & A