OS 2025 MP3 Scheduling

TA : R13922031 李仁偉, R13946005 程涵寧

Overview

Part I - None Real-time scheduling

1. Highest response ratio next
2. Priority-based Round Robin

Part II - Real-time scheduling

3. Deadline-Monotonic Scheduling
4. Earliest Deadline First with Constant Bandwidth Server

Parameter

```
int thrdstop_context_id;
int ID;                  // Unique ID
int is_real_time;        // 1 = RT, 0 = non-RT
int processing_time;     // Execution time per cycle
int deadline;            // = period for RT
int period;              // Cycle interval
int n;                   // RT cycles count
int remaining_time;      // Remaining time in current cycle
int current_deadline;    // Current deadline
int priority;            // For priority-based schedulers
int arrival_time;        // Release time of first cycle
```

Highest response ratio next

- Important SJF and SRTF disadvantage
  ➔ Possibility of starvation for longer processes

- HRRN choose next process with the greatest ratio:

$$\text{Response Ratio} = \frac{\text{Waiting Time} + \text{Burst Time}}{\text{Burst Time}}$$

- This favors shorter jobs that have waited longer without forcibly interrupting currently running threads.

Highest response ratio next

Process	Burst Time	Arrival Time
P1	10	0
P2	6	3
P3	4	3
P4	3	8
P5	5	13

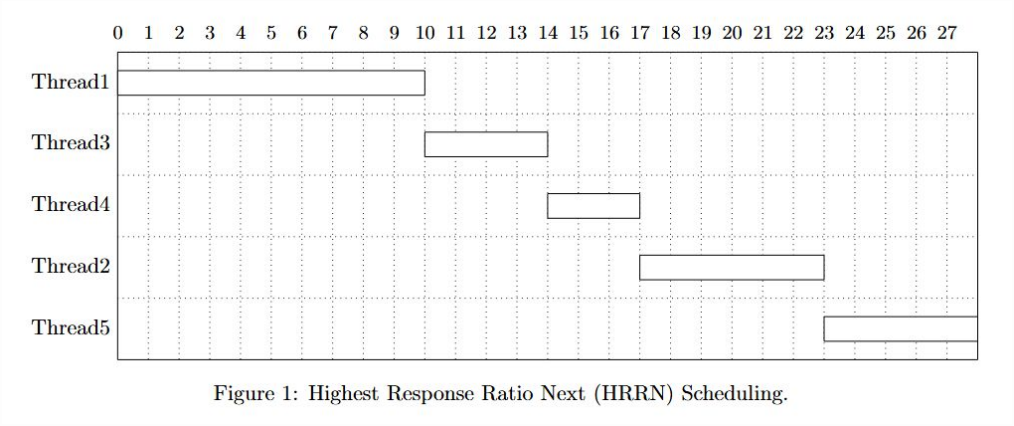

Figure 1: Highest Response Ratio Next (HRRN) Scheduling.

Priority-based Round Robin

- The scheduler always selects the **highest-priority** group first. Once all threads of the highest priority are finished, the next lower-priority group is scheduled.

- If multiple threads share the **same priority**, they are scheduled in **round-robin** order, breaking ties by thread ID.

Priority-based Round Robin

Process	Burst Time	Priority
P1	4	3
P2	5	2
P3	6	2
P4	7	1
P5	3	3

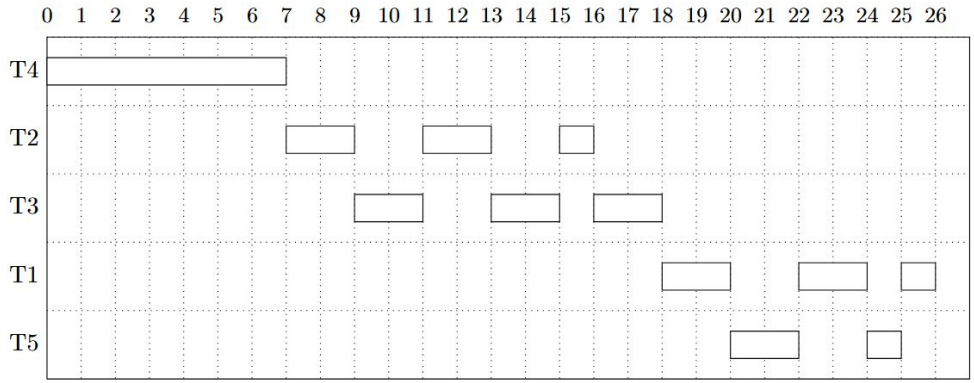

Figure 2: Priority-based Round Robin Scheduling with 5 threads.

Deadline Monotonic

- DM is a fixed priority based algorithm.Task with shortest deadline is assigned highest priority.

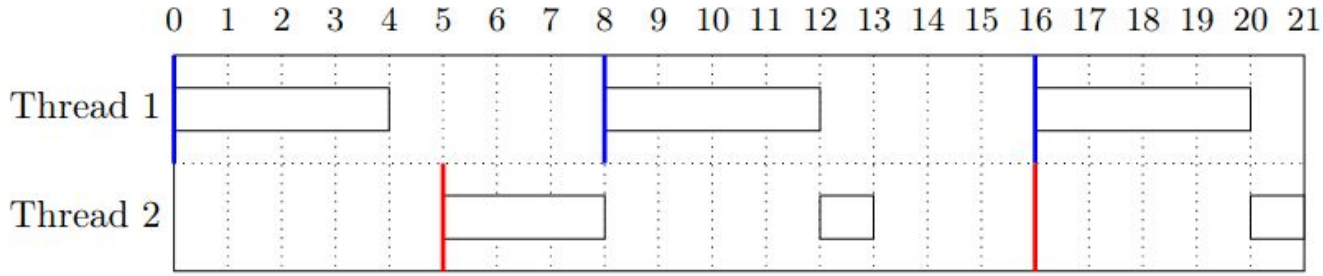

Figure 6: Deadline Monotonic Scheduling with thread 1 ($t = 4$, $p = 8$) and thread 2 ($t = 4$, $p = 11$). Thread 1 arrives at tick 0 while thread 2 arrives at tick 5. Blue and red vertical lines represent the start of each cycle. Due to deadline ($Thread$ 1) $<$ deadline ($Thread$ 2),So $Thread$ 1 has higher priority. At tick 8, $thread$ 2 still has 1 tick remaining , but $Thread$ 1 has higher priority so $Thread$ 1 gets CPU to execute.

Earliest Deadline First with Constant Bandwidth Server

- **Goal:** Introduce real-time scheduling using EDF with CBS
- **Context:** Used in SCHED_DEADLINE (Linux kernel 3.14+)
- **Key Concepts:**
  - EDF (Earliest Deadline First)
  - CBS (Constant Bandwidth Server)

Earliest Deadline First with Constant Bandwidth Server

- **Problem in plain EDF:** A task may overrun and affect others

- **CBS Solution:** Introduces bandwidth isolation
  - Limits CPU usage per task
  - Ensures tasks are throttled when exceeding their budget

- Prevents misbehaving tasks from breaking deadlines of others

Constant Bandwidth Server parameter

```
struct {
    int budget;
    int remaining_budget;
    int is_hard_rt;
    int is_throttled;
    int throttled_arrived_time;
    int throttle_new_deadline;
} cbs;
```

Constant Bandwidth Server Rules

- EDF based (preemptive)
- Check $\frac{\text{Remaining CBS Budget Time}}{\text{Time Until CBS Deadline}} > \frac{\text{CBS Budget Time}}{\text{Period}}$ or

  current_deadline < current_time when a **soft task** is about to run.

current_deadline = current time + period
remaining budget = budget

Earliest Deadline First with Constant Bandwidth Server

Task	Type	Arrival Time	Burst Time	Period	Deadline	CBS Budget
H1	Hard	0	15	20	20	-
S1	Soft	5	10	15	15	10

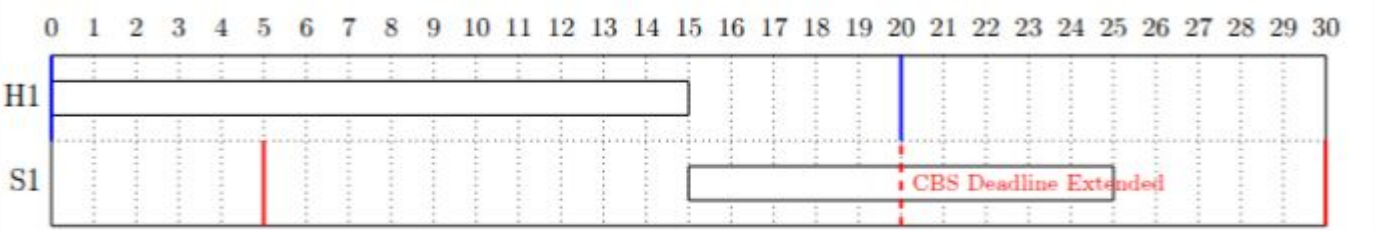

Figure 3: EDF with CBS Scheduling Timeline.

Throttled

- Remaining_budget <= 0 && remaining_time > 0
- Check is_throttled = 1 and current_deadline = current_time

current_deadline = current_deadline + period
remaining budget = budget

Environment Setup

- Download the MP3.zip from NTU COOL, unzip it and enter it

  $ unzip MP3.zip
  $ cd mp3

- Pull the Docker image from Docker Hub

  $ docker pull ntuos/mp3

- In the mp3 directory, use docker run to enter the container

  $ docker run -it -v $(pwd)/xv6:/home/xv6/ -w /home/xv6/ ntuos/mp3

Submission

- **Due Date: May 05 (MON), 23:59:00**
- Run "`make STUDENT_ID=<your_id> zip`" with LOWERCASE in the xv6 container, and submit the zip file to NTU COOL.
- For example, if your id is b12345678

```
$ make STUDENT_ID=b12345678 zip
```

- We will only accept your **`thread_sched.c`** in zip file. If you modify any other files and the code fails to compile, you will receive **zero points.**

Grading

- There are public test cases and private test cases.
  - Part 1 public test cases: 20%, 10% per algorithm.
  - Part 1 private test cases: 30%, 15% per algorithm.
  - Part 2 public test cases: 20%, 10% per algorithm.
  - Part 2 private test cases: 30%, 15% per algorithm.
- Only the latest submission is judged, even it's over the deadline.
- Compilation error leads to 0 points.
- Erroneous folder structure leads to 0 points.
- Late submission incurs 20 point penalty per day.

TA hours and Rules

- Email: ntuos@googlegroups.com
- TA hours: WED. 10:00~11:00, THU. 10:00~11:00 @CSIE R604

- We will **NOT** answer any questions about **coding** or **debugging** but if you have any question about the specification (eg. Wrong test case) or the algorithms, feel free to coming to the TA hours

- Please using website, AI, discussion on cool to solved your problems first, and if you can, sending e-mail first before coming to TA hours.

Reference for CBS　(only for reference)

- Deadline Task Scheduling
- Linux Deadline調度