RTS Project 2
Scheduling in Linux

Advisor: Prof. Tei-Wei Kuo

TA: Wen Sheng Lim
d08922028@csie.ntu.edu.tw

Slides credit: Han-Yi Lin

Goal

- Run a video player and a CPU-intensive background application, and schedule both for responsiveness and throughput
- Modify EDF to SJF (Shortest Job First) Scheduling Algorithm
- Achieve by assigning proper CPU utilization via scheduling parameters – runtime, period and deadline

Outline

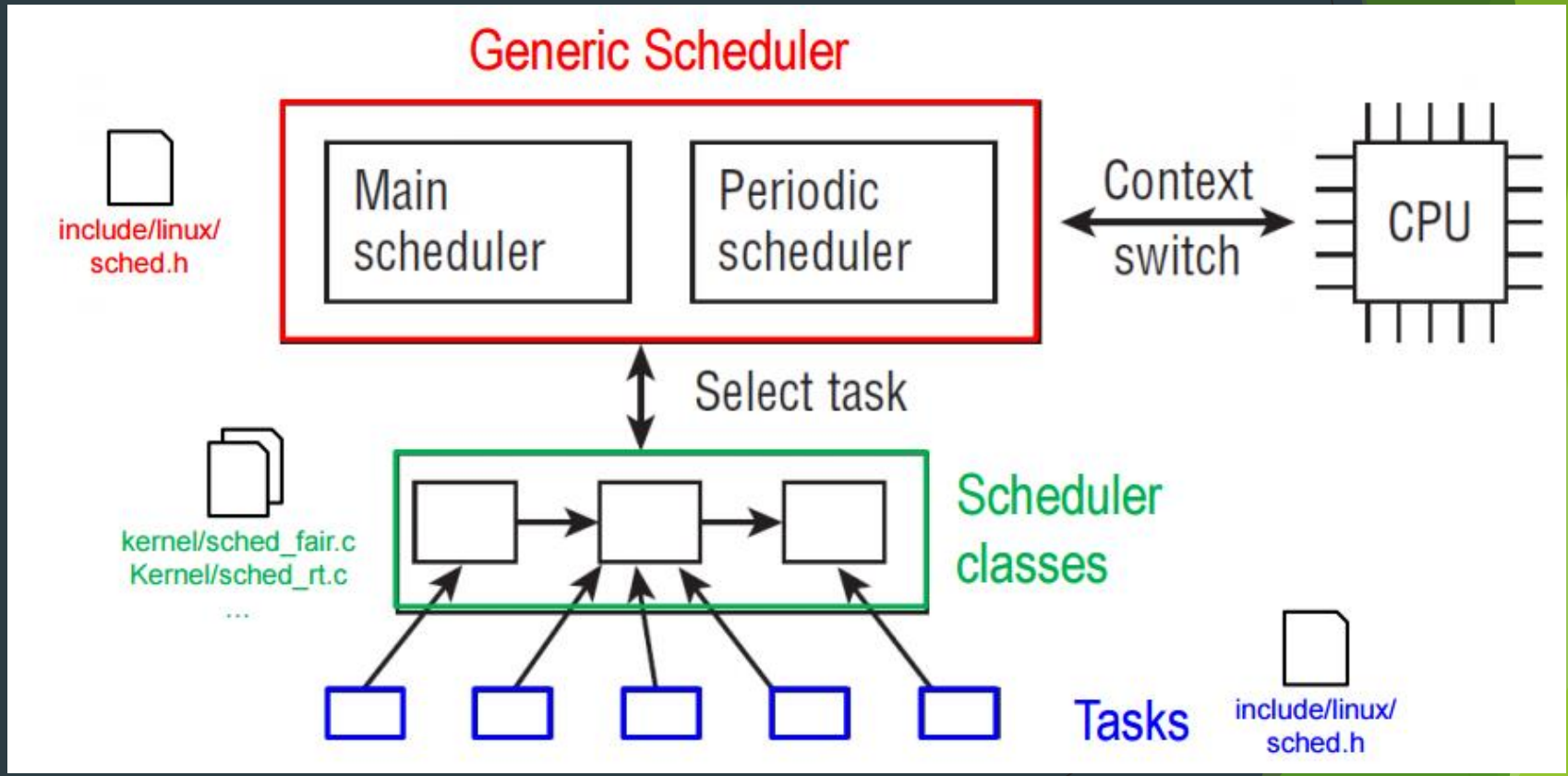
- Scheduling in Linux
- Project Requirements
- Submission Rules
- References

Overview of the Scheduling Subsystem in Linux

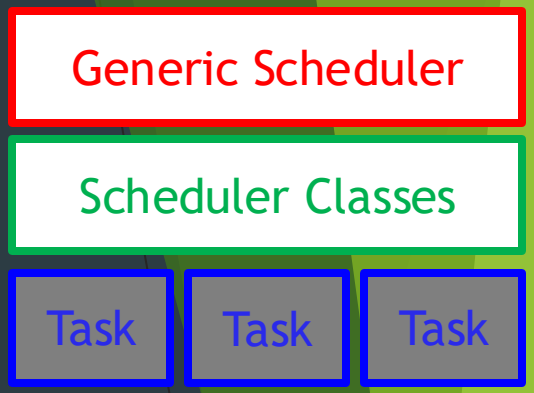
Generic Scheduler

Scheduler Classes

Task | Task | Task

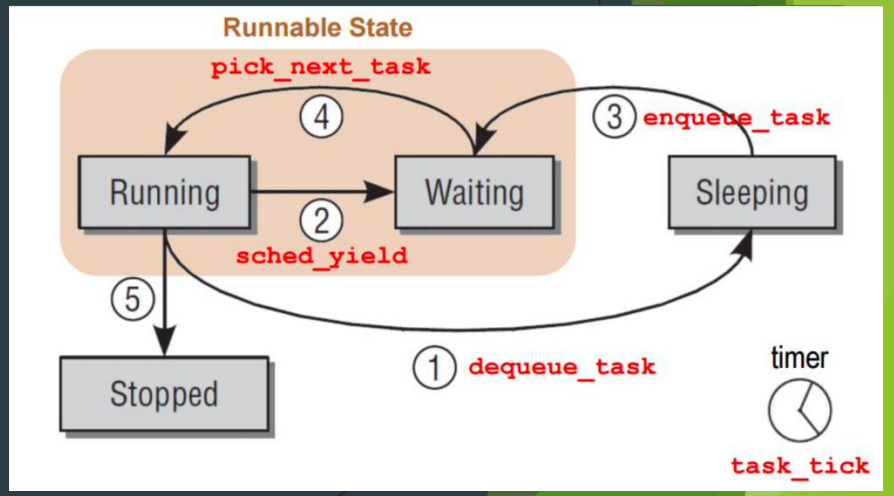


Scheduler Classes

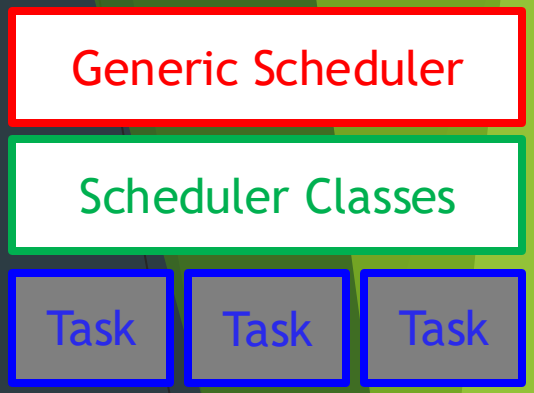


- An instance of struct sched_class defines function pointers for various operations of each scheduling class.

```
DEFINE_SCHED_CLASS(dl) = {

        .enqueue_task           = enqueue_task_dl,
        .dequeue_task           = dequeue_task_dl,
        .yield_task             = yield_task_dl,

        .check_preempt_curr     = check_preempt_curr_dl,

        .pick_next_task                 = pick_next_task_dl,
        .put_prev_task          = put_prev_task_dl,
        .set_next_task          = set_next_task_dl,
```

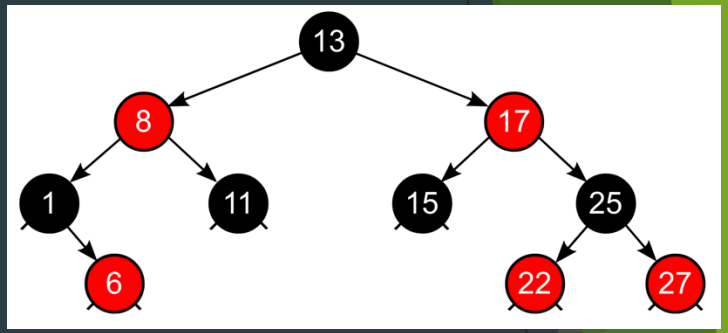


Scheduler Classes



- enqueue_task: adds a new process to the run queue. This happens when a process changes from a sleeping into a runnable state.
- dequeue_task: provides the inverse operation: It takes a process off a run queue. Naturally, this happens when a process switches from a runnable into an un-runnable state, or when the kernel decides to take it off the run queue for other reasons.
- pick_next_task: selects the next task that is supposed to run. These operations are not equivalent to putting tasks on and off the run queue like enqueue_task and dequeue_task. Instead, they are responsible to give the CPU to a task, respectively, take it away.

The Linux Deadline Scheduler



- Earliest Deadline First (EDF)
  - Each task is tracked as a node of a red-black tree
  - Nodes are ordered by dynamic scheduling deadline
  - In enqueue_task, the new node is inserted to a proper place in the tree
  - In pick_next_task, the node with earliest deadline is picked

```
static void __enqueue_dl_entity(struct sched_dl_entity *dl_se)
{
        struct dl_rq *dl_rq = dl_rq_of_se(dl_se);

        BUG_ON(!RB_EMPTY_NODE(&dl_se->rb_node));

        rb_add_cached(&dl_se->rb_node, &dl_rq->root, __dl_less);

        inc_dl_tasks(dl_se, dl_rq);
}
```

Outline

- Scheduling in Linux
- Project Requirements
- Submission Rules
- References

Implement Shortest Job First (SJF) scheduler

- Replace the EDF part in the deadline scheduler with SJF
- Keep CBS for assigning dynamic scheduling deadline and remaining runtime
- Use “remaining runtime” parameter as the CPU burst time for SJF (i.e., in sched_dl_entity)

```
583 struct sched_dl_entity {
584         struct rb_node                  rb_node;
585
586         /*
587          * Original scheduling parameters. Copied here from sched_attr
588          * during sched_setattr(), they will remain the same until
589          * the next sched_setattr().
590          */
591         u64                             dl_runtime;     /* Maximum runtime for each instance   */
592         u64                             dl_deadline;    /* Relative deadline of each instance  */
593         u64                             dl_period;      /* Separation of two instances (period) */
594         u64                             dl_bw;          /* dl_runtime / dl_period              */
595         u64                             dl_density;     /* dl_runtime / dl_deadline            */
596
597         /*
598          * Actual scheduling parameters. Initialized with the values above,
599          * they are continuously updated during task execution. Note that
600          * the remaining runtime could be < 0 in case we are in overrun.
601          */
602         s64                             runtime;        /* Remaining runtime for this instance        */
603         u64                             deadline;       /* Absolute deadline for this instance        */
604         unsigned int                    flags;          /* Specifying the scheduler behaviour */
```

Benchmark

- Use only 1 CPU for the VM to avoid complexity of multi-core scheduling
- $ sudo apt-get install p7zip-full sysstat
- Play a video along with a CPU-intensive application (compression)
  - $ 7z b 100 –md16
  - Use a high resolution (>=720p and >=24fps) video for experiment
- Use deadline (and SJF) scheduler for both the video player and the compression application
- Adjust runtime/period/deadline parameters for:
  - Make the video player meet deadline (no dropped frames)
  - Keep high CPU utilization (i.e., 7z) for the system

Benchmark

- Measure average CPU utilization for every 5-second window
  - Recommended idle < 25%
  - $ mpstat 5

21時30分55秒	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
21時31分00秒	all	85.11	0.00	1.28	0.43	0.00	0.00	0.00	0.00	0.00	13.19
21時31分05秒	all	84.21	0.00	1.47	0.00	0.00	0.00	0.00	0.00	0.00	14.32
21時31分10秒	all	84.81	0.00	1.05	0.42	0.00	0.00	0.00	0.00	0.00	13.71
21時31分15秒	all	84.00	0.00	1.26	0.21	0.00	0.00	0.00	0.00	0.00	14.53
21時31分20秒	all	83.58	0.00	1.28	0.85	0.00	0.00	0.00	0.00	0.00	14.29
21時31分25秒	all	83.33	0.00	1.27	0.00	0.00	0.00	0.00	0.00	0.00	15.40
21時31分30秒	all	84.03	0.00	1.26	0.00	0.00	0.00	0.00	0.00	0.00	14.71
21時31分35秒	all	82.87	0.00	1.71	0.00	0.00	0.00	0.00	0.00	0.00	15.42
21時31分40秒	all	83.47	0.00	1.46	0.00	0.00	0.00	0.00	0.00	0.00	15.06
21時31分45秒	all	84.01	0.00	1.07	0.64	0.00	0.00	0.00	0.00	0.00	14.29

Project Requirements

- Adjust scheduling parameters manually with `chrt` (100%)
  - Describe how you find proper parameters, like criteria for parameter selection
  - Do this on both EDF and SJF variants of the deadline scheduler
  - Observations – ex: what happens with different scheduling parameters? differences between EDF and SJF?
  - Experiment results: the used parameters and the CPU utilization

Outline

- Scheduling in Linux
- Project Requirements
- Submission Rules
- References

Project 2 Requirements

- A PDF report within 4 pages
- modified kernel source files
- Please show your names and student IDs in your report
- Concretely describe work done by each member
- Be packed as one file named "RTS_PJ2_Team##_vN.zip"
  - RTS_PJ2_report.pdf
  - linux-5.15.71/kernel/sched/deadline.c
  - other files if needed

Submission Rules

- Project deadline: 2025/06/06 (Friday) 23:59
  - Delayed penalty: -20/Day
- Upload the zip file via NTU COOL
- DO NOT COPY THE HOMEWORK
  - Discussions among teams are encouraged, as long as properly credited

References

- Professional Linux® Kernel Architecture, Wolfgang Mauerer, Wiley Publishing, Inc.
- Deadline Task Scheduling https://www.kernel.org/doc/html/latest/scheduler/sched-deadline.html
- L. Abeni , G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. Proc. of RTSS, 1998. http://retis.sssup.it/~giorgio/paps/1998/rtss98-cbs.pdf
- Red-black Tree https://en.wikipedia.org/wiki/Red%E2%80%93black_tree