OS PROJECT 2
PROCESS SCHEDULING IN LINUX

Advisor: Chien-Chung Ho

TA: Zong-Zhe Yang

Shih-Hao Chin

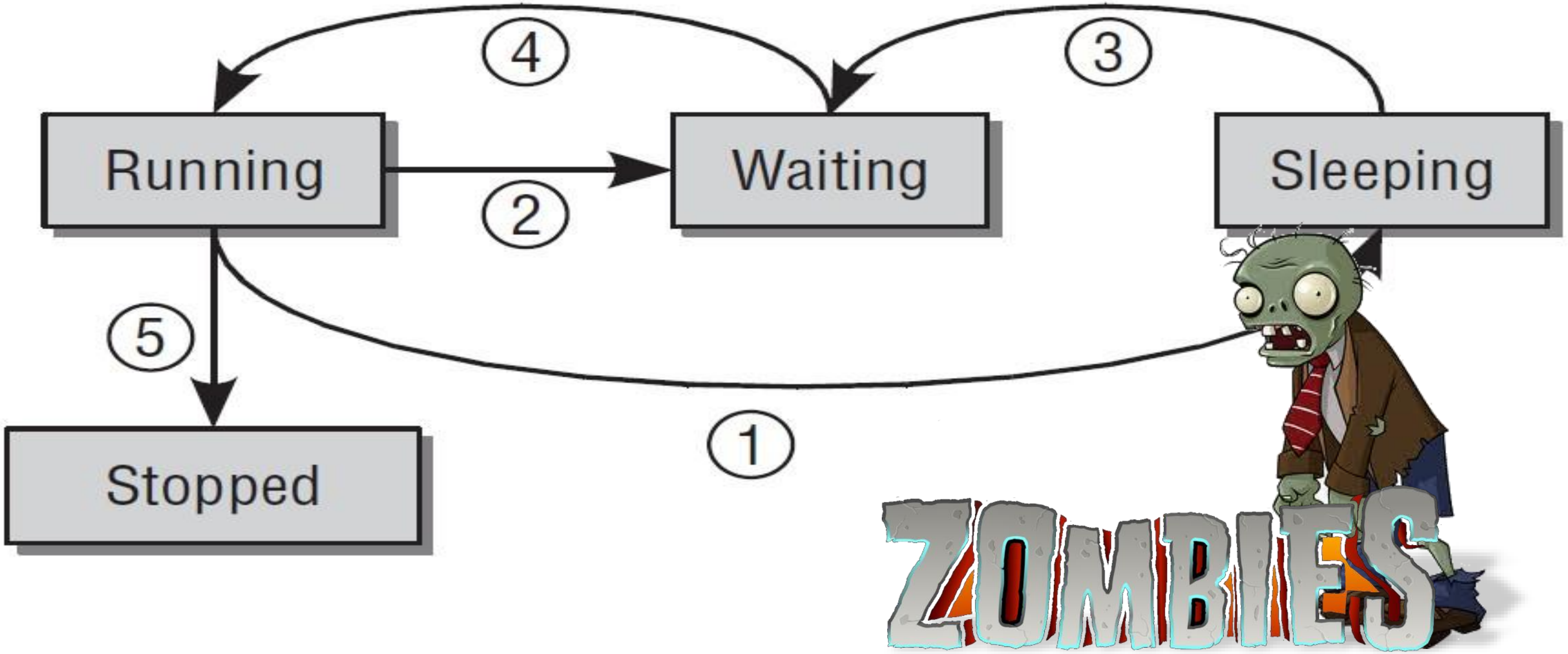
OUTLINE

- Introduction
- Project Requirements
- Submission Rules
- References

PROCESS LIFE CYCLE

- A process is *not* always ready to run.
- The scheduler must know the status of every
  - process in the system when switching between tasks.
- A process may have one of the following states:
  - **Running** — The process is executing at the moment.
  - **Waiting** — The process is able to run but is not allowed to because the CPU is allocated to another process. The scheduler can select the process at the next task switch.
  - **Sleeping** — The process is sleeping and cannot run because it is waiting for an external event. The scheduler *cannot* select the process at the next task switch.
- The system saves all processes in a process table.

TRANSITIONS BETWEEN PROCESS STATES



SCHEDULING IN LINUX (1/2)

- The schedule function is the starting point to an understanding of scheduling operations.
- It is defined in “kernel/sched/core.c” and is one of the most frequently invoked functions in the kernel code.
- Not only *priority scheduling* but also two other soft real-time policies required by the POSIX standard are implemented.
  - E.g., completely fair scheduling, real-time scheduling and scheduling of the idle task, *etc*.

SCHEDULING IN LINUX (2/2)

- The scheduler uses a series of data structures to sort and manage the processes in the system.
- Scheduling can be activated in two ways:
  - Main scheduler: Either directly if a task goes to sleep or wants to yield the CPU for other reasons,
  - Periodic scheduler: Or by a periodic mechanism that is run with constant frequency to check from time to time if switching tasks is necessary

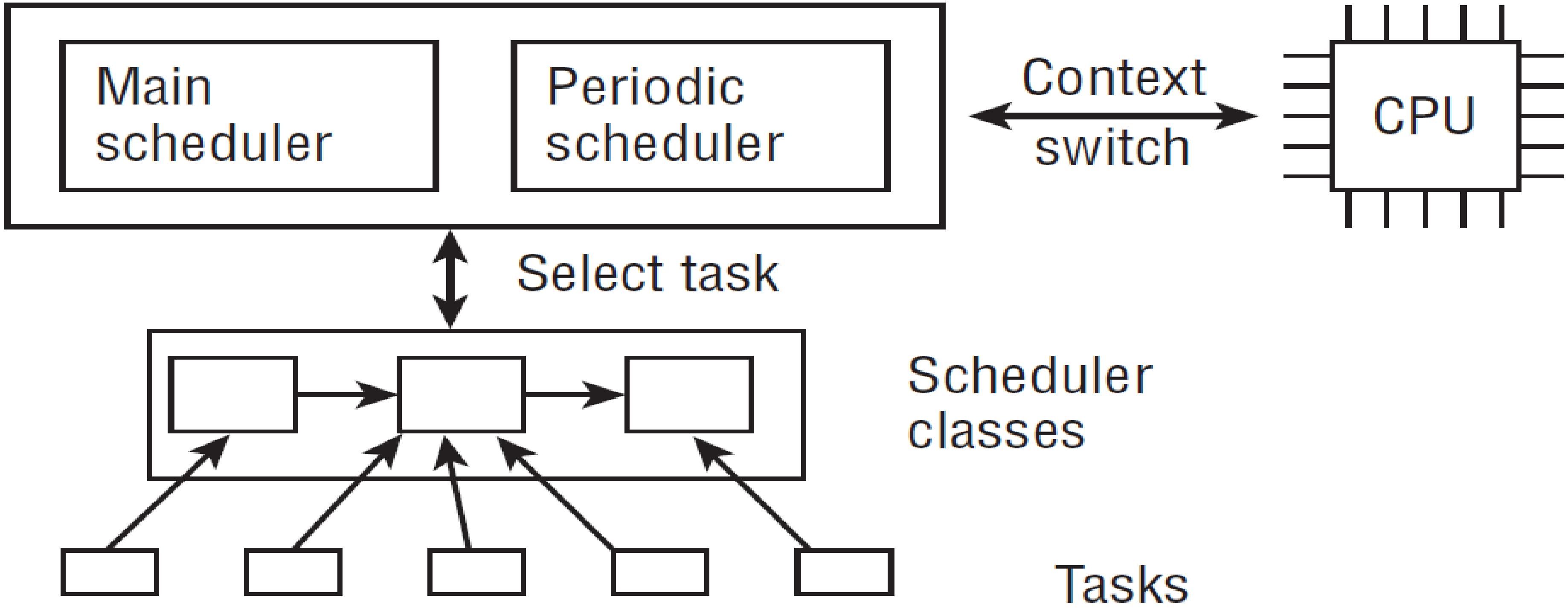
Generic scheduler = Main + Periodic schedulers

OVERVIEW OF THE COMPONENTS OF THE SCHEDULING SUBSYSTEM

Generic Scheduler

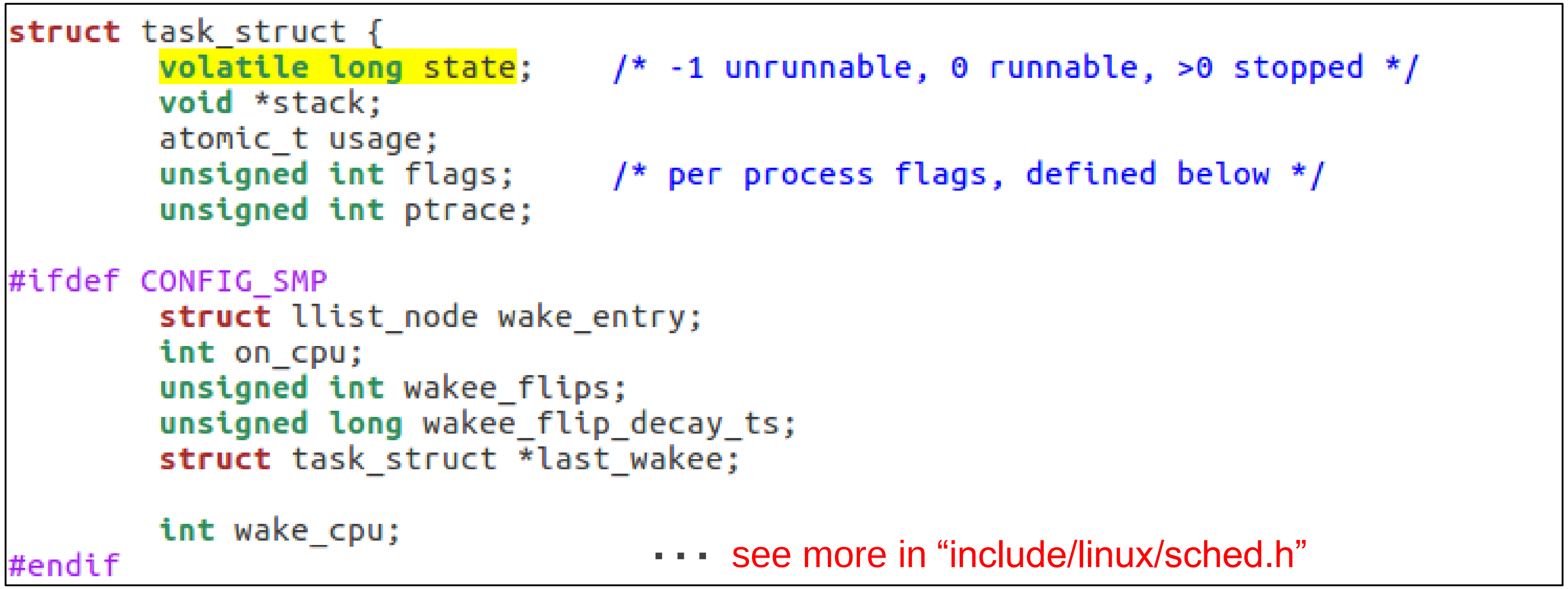
Scheduler Classes

Task Task Task



EXAMPLE:
PROCESS REPRESENTATION IN LINUX

- In Linux, all concerned with processes and programs are built around a data structure: **task_struct**.



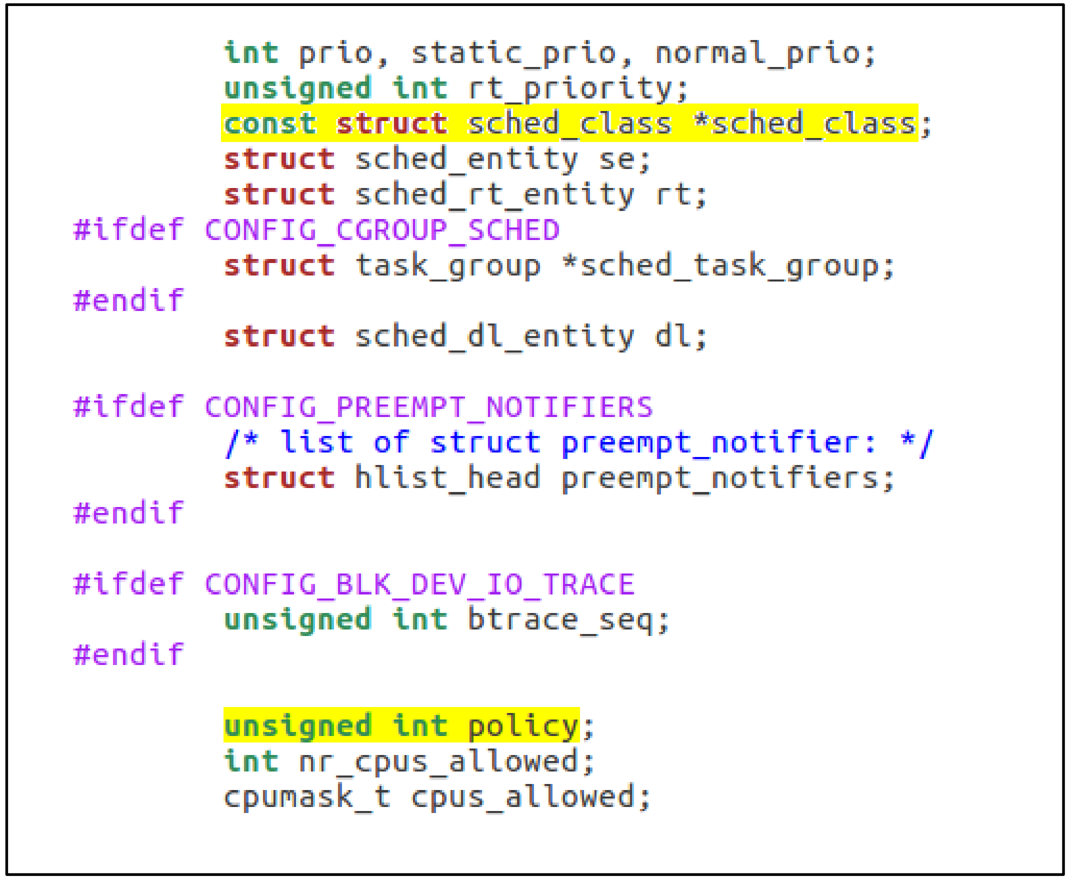
```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        unsigned int wakee_flips;
        unsigned long wakee_flip_decay_ts;
        struct task_struct *last_wakee;

        int wake_cpu;
#endif
```

… see more in “include/linux/sched.h”

HOW TO DESIGNATE A SCHEDULER FOR TASKS?



```
        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
#ifdef CONFIG_CGROUP_SCHED
        struct task_group *sched_task_group;
#endif
        struct sched_dl_entity dl;

#ifdef CONFIG_PREEMPT_NOTIFIERS
        /* list of struct preempt_notifier: */
        struct hlist_head preempt_notifiers;
#endif

#ifdef CONFIG_BLK_DEV_IO_TRACE
        unsigned int btrace_seq;
#endif

        unsigned int policy;
        int nr_cpus_allowed;
        cpumask_t cpus_allowed;
```

OUTLINE

- Introduction
- Project Requirements
- Submission Rules
- References

LINUX SCHEDULING POLICIES

- Linux Scheduling Polices
  - Normal Scheduling policies (Non-real-time)
    - SCHED_OTHER, SCHED_BATCH, SCHED_IDLE.
  - Real-Time policies
    - SCHED_FIFO, SCHED_RR.
- The default scheduling policy is non-real-time.
- In this part, using Linux real-time scheduling policy (FIFO) to schedule threads in a process.

PROJECT REQUIREMENTS

1) Write a C program (sched_test.c) to create threads
2) Need busy-waiting to ensure that process exist long
3) Run the program by default time sharing schedule policy and show the result. (20%)
   - Ex: sudo ./sched_test 5
4) Run the program by real time scheduling policy (RR,FIFO) and show the result.(40%)
   - Ex: sudo ./sched_test 5 SCHED_FIFO
5) The parameters must be able to select the number of threads and the policy

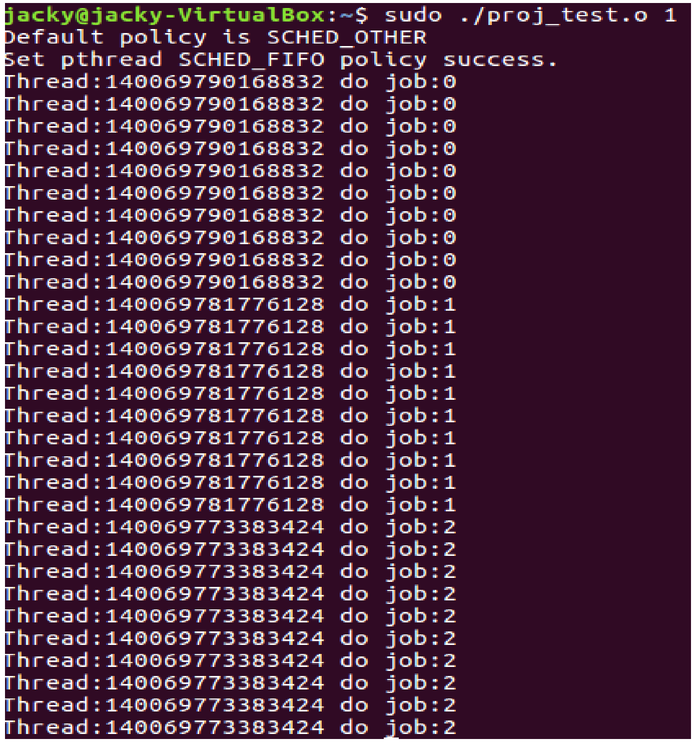
HINT

- Set CPU affinity
- sched_setscheduler
- Set the priority of real time process sched_param *param
- The permission to run real time process

PROJECT REQUIREMENTS

6) Install kernelShark + trace-cmd to analyze process (ex:FIFO)

https://hackmd.io/@ULxjDFy0QLKSAQnD5DMKtQ/HJfBkEYnH



```
jacky@jacky-VirtualBox:~$ sudo ./proj_test.o 1
Default policy is SCHED_OTHER
Set pthread SCHED_FIFO policy success.
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069790168832 do job:0
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069781776128 do job:1
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
Thread:140069773383424 do job:2
```

PROJECT REQUIREMENTS

- Report(.PDF) (60%)
  - What do you learn from this project
  - Is there any problem, state it Your implementation details
  - The test results (contains the resulting diagram generated by kernel shark)
  - At most 6 pages

OUTLINE

- Introduction
- Project Requirements
- Submission Rules
- References

SUBMISSION RULES

- Be packed as one file named “OSPJ2_Team##.tar”
  - OSPJ2_Team## (directory)
    - report.pdf
    - sched_test.c
    - sched_test.o

SUBMISSION RULES

- Project deadline: 2019/12/10 (23:59)
  - Delayed submissions yield 5 point deduction per day
- Upload your team project to the FTP site.
  - **FTP server: 140.123.105.185**
  - **user ID: OS2019 ; pass: 2019OS**
- If you've submitted files for many times, TAs will only check the up-to-date version.
- If the up-to-date file misses the deadline, it will be regarded as the delayed submission.
- **DO NOT COPY THE HOMEWORK**

CONTACT TAS

- If you have any problem about the projects or this course, you can contact TAs by the following ways.
- Facebook: CCU OS2019 Fall Group 👍
  - https://www.facebook.com/groups/1319131934920970/
  - E-mail: Shih-Hao Chin: ae8681239@gmail.com
    Zong-Zhe Yang : daankfust123@gmail.com

OUTLINE

- Introduction
- Project Requirements
- Submission Rules
- References

REFERENCES

IMPORTANT

- Reference Book
  - Professional Linux® Kernel Architecture, Wolfgang Mauerer, Wiley Publishing, Inc.
- Process Scheduling
  - *http://www.cs.rutgers.edu/~pxk/416/notes/07- scheduling.html*