OS 2021

Homework 3: thread scheduler

(Due date 12/21 00:00:00)

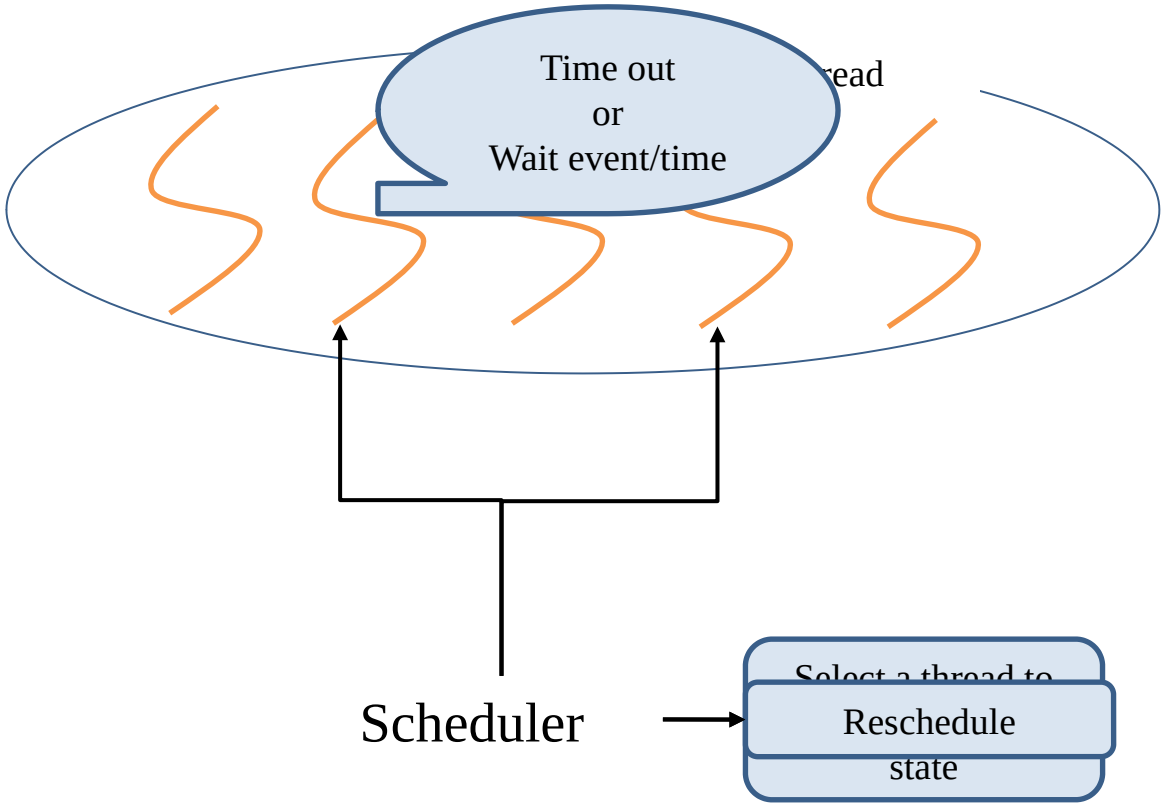
GitHub Classroom

TA: Tsen-En, Chou 周岑恩

Email: p76091234@gs.ncku.edu.tw

Overview

This homework will require students to create multiple threads and schedule these threads according to the homework requirements.



Overview

1. Scheduler
   - The scheduler will select a thread to enter the running state according to the scheduling algorithm.

2. Threads
   - Each thread has its **priority**, **name**, **entry function**, and **cancellation mode**.

Objectives

- Understand how to implement a user-level thread scheduler
- Understand how to implement signal handler
- Understand how to realize asy./deferred thread cancellation

Requirements (1/4)

Create threads

- Initial user threads are specified in a static thread configuration file.
- The name of the thread configuration file must be **init_threads.json.**
- You should implement a function that can parse init_threads.json and generate the corresponding thread to the ready queue.
- The JSON format
  - name
    - A descriptive name for the thread. It is included purely as a debugging aid.
  - entry function
    - One of the five functions: Function1 to Function5
    - Function implementations are provided by the TAs and included in function_library.c. All function implementations can not be changed.
  - priority
    - Three priority levels, H, M, L.
    - You can decide the priority of the thread yourself.
  - cancel mode
    - Can be 1 or 0.
    - If setting **1** means that the thread is **deferred cancellation** type.
    - If setting **0** means that the thread is **asynchronous cancellation** type.

Requirements (1/4)

A 2-thread example of the thread configuration file

{} init_threads.json > [ ] Threads > {} 1 > entry function

```
{
    "Threads":[
        {
            "name" : "f4",
            "entry function" : "Function4",
            "priority": "M",
            "cancel mode": "0"
        },
        {
            "name" : "f5",
            "entry function" : "Function5",
            "priority": "M",
            "cancel mode": "0"
        }
    ]
}
```

Requirements (2/4)

Scheduler

- Use ucontext and related APIs to implement context switch
  - See the References slide (*slide 27*)

- Implement **Multilevel Feedback Queue**
  - More detailed requirements will be *described in slide 13.*
  - There should be a timer that sends a signal (SIGALRM) every 10ms.
    - Calculate all thread-related time.
    - Check if any threads need to switch state.
    - Decide whether you need to wake up the scheduler.

    e.g.,

    1. The thread currently in the running state has 30ms left in time quantum(TQ). There is no need to switch thread states, also no need to wake up the scheduler.

    2. The waiting time for a thread in the waiting queue has expired and needs to be switched to the READY state. Wake up the scheduler according to the scheduling algorithm.

Requirements (2/4)

- An additional thread must be created before the system starts scheduling.
  - name: *reclaimer*
  - entry function: ResourceReclaim(Defined in function_library.c)
  - priority: L
  - cancel mode: 1

  Reclaimer can not be included in init_threads.json.
  Reclaimer will not enter the terminated state.

Requirements (3/4)

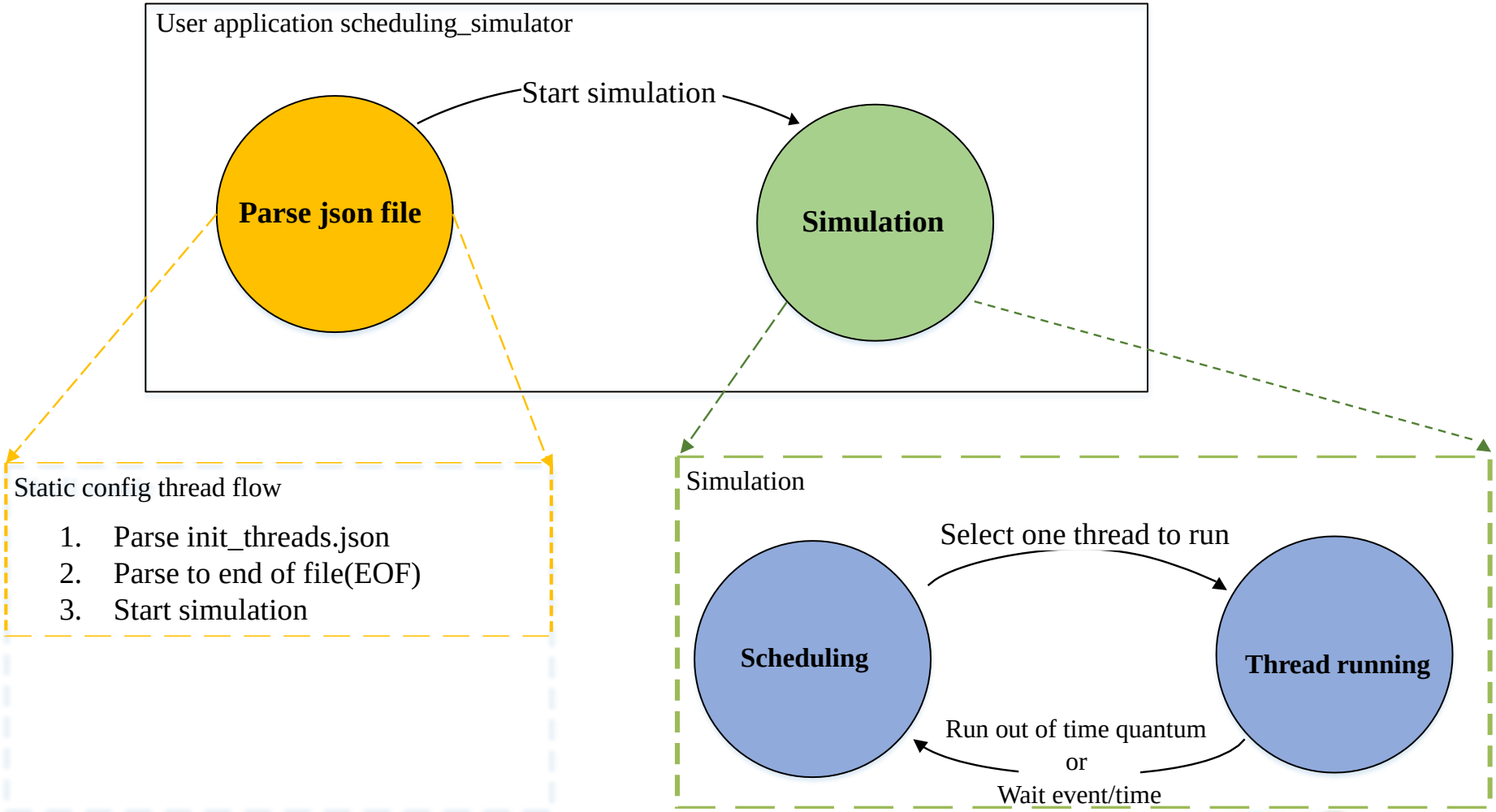
- Report
  - Use **ctrl + z** to report threads information on the terminal.
  - Base priority
    - Thread priority configuration in init_threads.json.
  - Current priority
    - The priority of the thread after the change according to the scheduling algorithm.
  - Queueing time
    - The total time the thread **stays in the ready state** during all the simulation period.
  - Waiting time
    - The total time the thread **stays in the waiting state** during all the simulation period.

TID	Name	Thread state (in slide 12)	Base Priority	Current Priority	Queueing time	Waiting time
1	P_Hello	WAITING	L	M	120	20
2	CalcAvg	RUNNING	H	M	0	0
3	IdleTask	READY	L	L	120	0

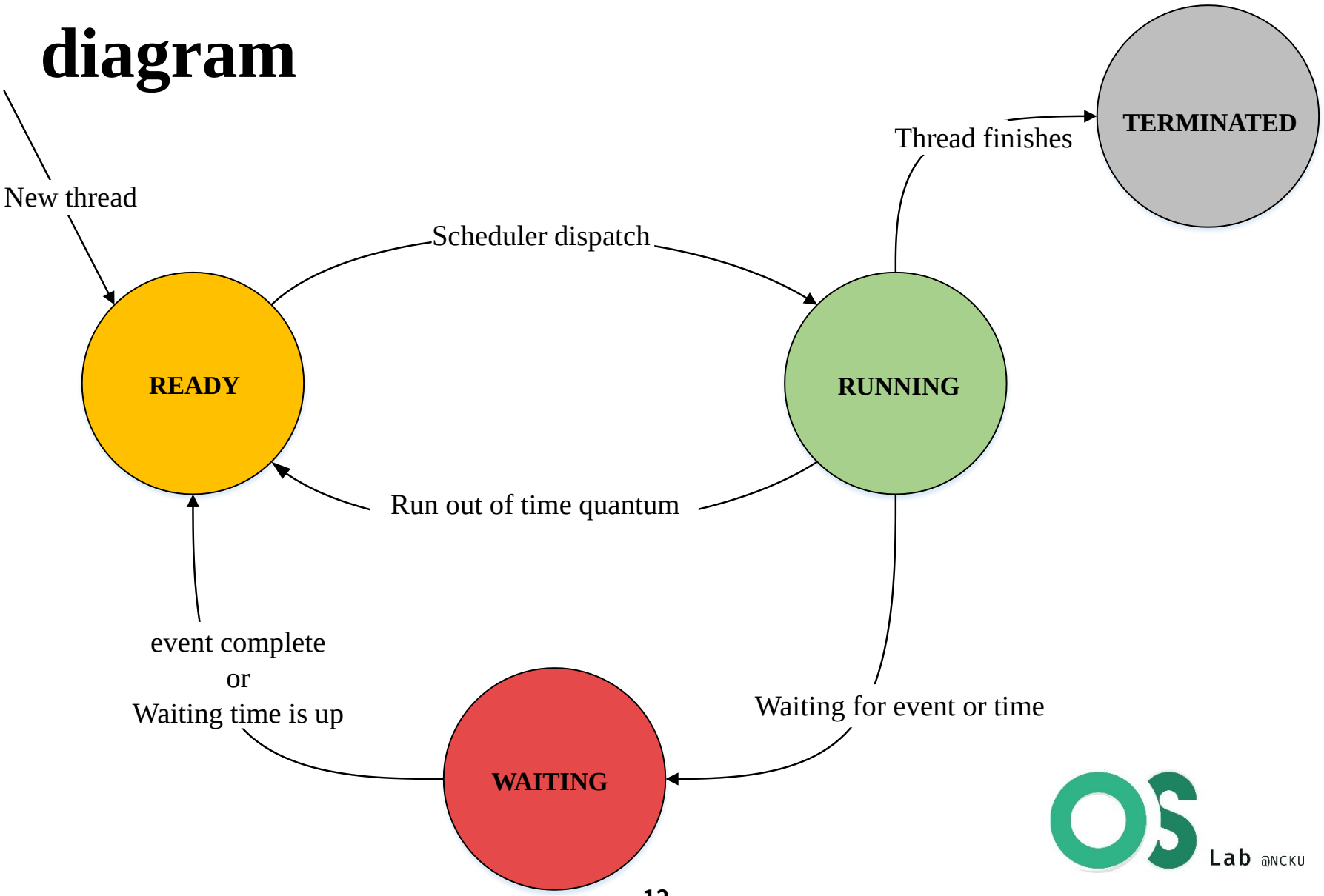
Requirements (4/4)

- Implement APIs that can be used by threads(*described in slide 16*)
  - int OS2021_ThreadCreate(char **name*, char * *p_function*, char **priority*, int *cancel_mode*);
  - int OS2021_ThreadCancel(char **name*);
  - void OS2021_ThreadWaitEvent(int *event_id*);
  - void OS2021_ThreadSetEvent(int *event_id*);
  - void OS2021_ThreadWaitTime(int *10msec*);
  - void OS2021_DeallocateThreadResource();
  - void OS2021_TestCancel();

Architecture

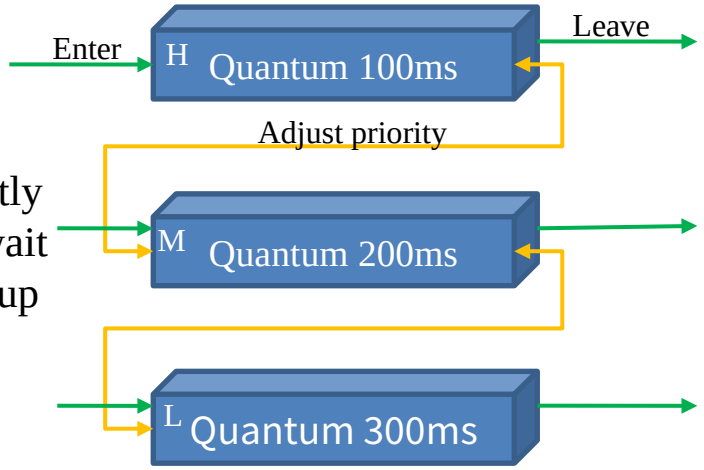


Thread state diagram



Multilevel Feedback Queue

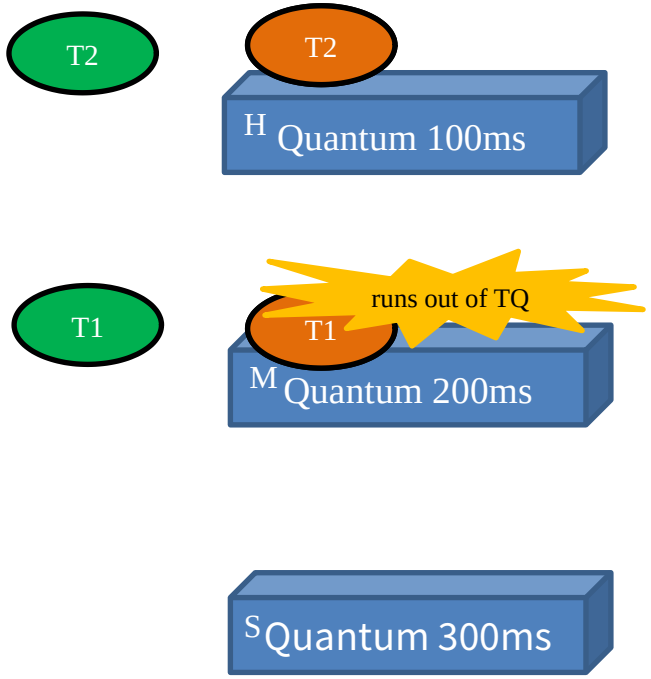
- 3-level Non-preemptive Priority Feedback Queue
  - H – Round Robin(RR) with Time Quantum(TQ) 100ms
  - M – RR with TQ 200ms
  - L – RR with TQ 300ms
- Scheduling
  - Non-preemptive
    - If a thread with a priority higher than the currently running thread enters the ready queue, it must wait for the running thread to run out of TQ or give up the CPU before switching to a higher thread.
  - Adjust priority
    - If the thread gives up the CPU without running out of its TQ (Wait time or event), it increases its priority by one level.
    - If the thread runs out of its TQ, it decrease its priority by one level.
  - If the priority of the thread is changed, please print the information on the terminal.
    - e.g., The priority of thread xxx is changed form M to H.



Multilevel Feedback Queue

- Example1(Non-preemptive)
  1. Thread1(T1) initial priority is M. Enter M queue.
  2. T1 is running.
  3. Thread2(T2) initial priority is H. Enter H queue.
  4. T1 runs out of TQ in the M queue(200ms).
  5. T2 is running.

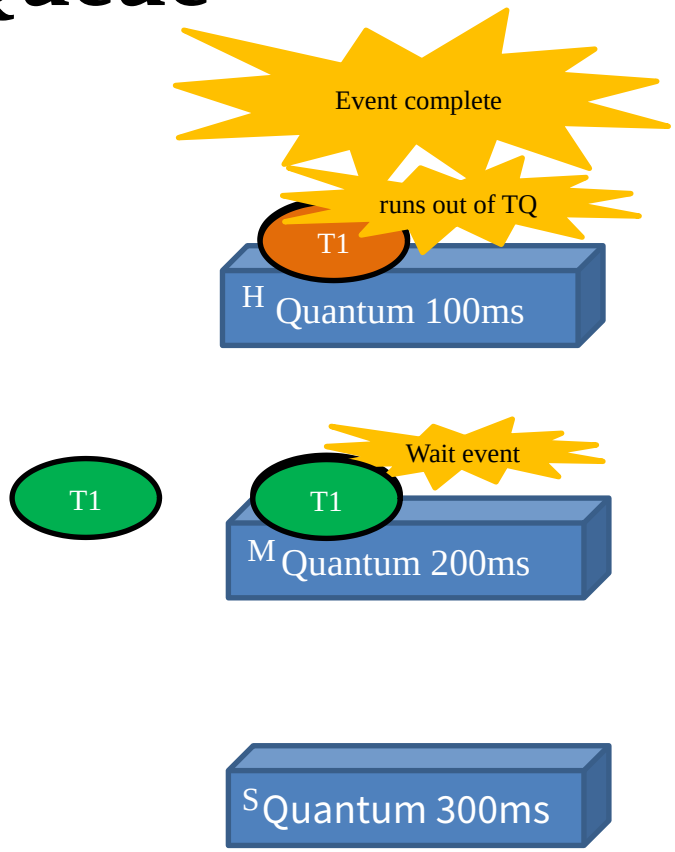
  p.s. Green: ready, Orange: running



Multilevel Feedback Queue

- Example2(Adjust priority)
  1. The initial priority of Thread1(T1) is M. Enter the M queue.
  2. T1 calls OS2021_ThreadWaitEvent() to make it enter the waiting state before running out of its TQ.
  3. The event is completed and is awakened by another thread.
  4. T1 enters H queue.
  5. T1 runs out of TQ in the H queue(100ms), and then enters the M queue when it returns to the ready state.

  p.s. Green: ready, Orange: running



API Description(1/3)

- int OS2021_ThreadCreate(char **name*, char **p_function*, char **priority*, int *cancel_mode*);
  - Create a thread named *name* and set its priority to *priority.*
  - If function_libary.c doesn't have a function named *p_function*, API **returns -1**.
  - The cancellation type of the thread is marked according to the *cancel_mode*.
  - **Return (Thread ID)TID** of the created thread.

- void OS2021_ThreadCancel(char **name*);
  - Cancel the thread named *name* according to the cancel mode of the thread.
  - If the cancel mode is 0, change the thread status to **TERMINATED**, and then the resource memory block will be reclaimed by the reclaimer.
  - If the cancel mode is 1, please inform the thread named *name* that another thread wants to cancel it.
    - Then *name* thread will be terminated when it enters the cancellation point

API Description(2/3)

A total of 8 types of events, use numbers 0~7 to number the events.

- void OS2021_threadWaitEvent(int *event_id*);
  - The running thread changes its state to **WAITING** and enters the event waiting queue corresponding to the *event_id*.
  - Reschedule if needed.
  - The thread calling this API must print the event it wants to wait for on the terminal.
    - e.g., xxx wants to wait for event 2.

- void OS2021_threadSetEvent(int *event_id*);
  - If there is currently a thread waiting for an event, the API will move the state of the thread from **WAITING** to **READY** and remove the thread from the event waiting queue corresponding to the *event_id*.
  - If no threads are waiting for the event, nothing is done.
  - If a thread is awakened, the wake-up information must be printed on the terminal.
    - e.g., xxx1 changes the status of xxx2 to **READY**.

API Description(2/3)

- Notice
  - If multiple threads are waiting for the same event, the thread with the highest priority will be awakened first.

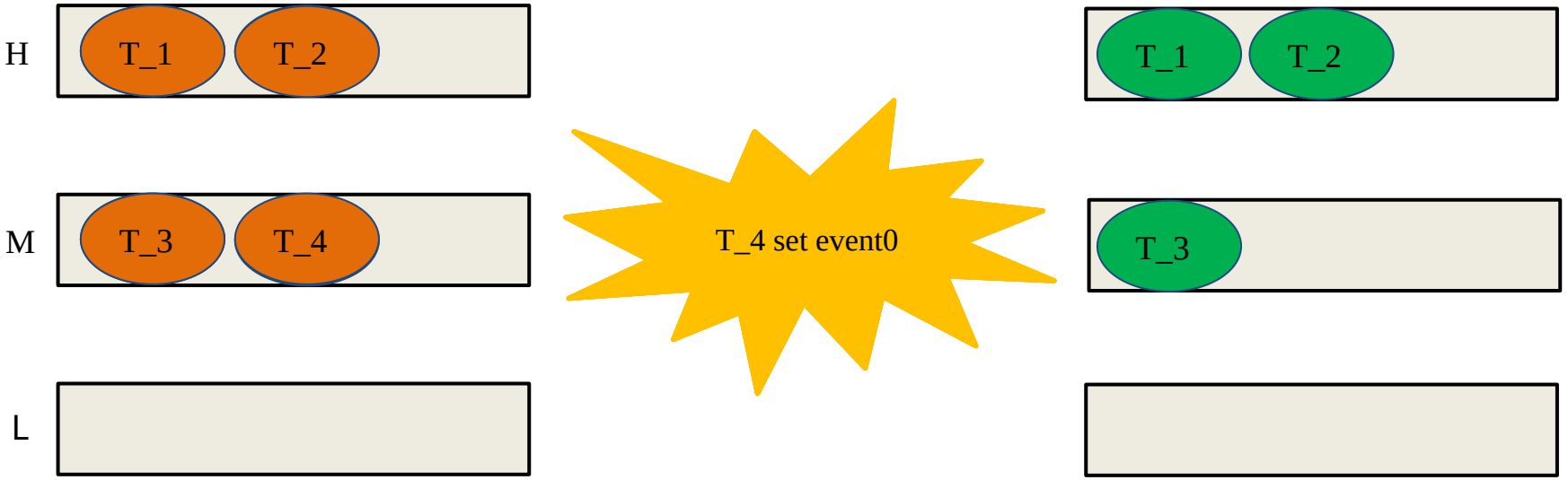
  - If the priority is the same, it will be served according to the first-come, first-served (FCFS) algorithm.

Event example

Ready queue

Orange: running

Event0 Waiting queue



API Description(3/3)

- void OS2021_ThreadWaitTime(int *10msec*);
  - The running task change its state to **WAITING.**
  - Change the state of the suspended task to **READY** after *10msec* * 10ms.
  - Reschedule (schedule next thread to run).

- void OS2021_DeallocateThreadResource();
  - This API will be used by ResourceReclaim().
  - Deallocate the memory of the thread that has been terminated.

- void OS2021_TestCancel();
  - Allows a thread to be cancelled in a safe point(cancellation point).

Hw3 Example1

{} init_threads.json > ...

```json
{
    "Threads":[
        {
            "name" : "f1",
            "entry function" : "Function1",
            "priority": "M",
            "cancel mode": "1"
        },
        {
            "name" : "f3",
            "entry function" : "Function3",
            "priority": "M",
            "cancel mode": "0"
        }
    ]
}
```

Hw3 Example1

```c
void Function1(void)
{
    int i,j;
    while (1)
    {
        i = OS2021_ThreadCreate("random_1","Function2","L",1);
        ((i>0) ? fprintf(stdout,"Created random_1 successfully\n"):
        fprintf(stdout,"Failed to create random_1\n"));
        fflush(stdout);

        j = OS2021_ThreadCreate("random_2","Function2","L",1);
        ((j>0) ? fprintf(stdout,"Created random_2 successfully\n"):
        fprintf(stdout,"Failed to create random_2\n"));
        fflush(stdout);

        OS2021_ThreadWaitEvent(3);

        ((i>0) ? OS2021_ThreadCancel("random_1"): "");
        ((j>0) ? OS2021_ThreadCancel("random_2"): "");
        while(1);
    }
}
```

```c
void Function2(void)
{
    int the_num;

    int min = 65400;
    int max = 65410;

    while (1)
    {
        srand(time(NULL));
        the_num = rand() % (max - min + 1) + min;
        if(the_num == 65409)
        {
            fprintf(stdout,"I found 65409.\n");
            fflush(stdout);
            OS2021_ThreadSetEvent(3);
            min = 0;
            max = 0;
        }
        OS2021_TestCancel();
    }
}
```

```c
void Function3(void)
{
    while(1)
    {
        OS2021_ThreadWaitEvent(3);
        fprintf(stdout,"I fell in love with the operating system.\n");
        fflush(stdout);
    }
}
```

Hw3 Example1

```
f1 wants to wait for event 3
The priority of thread f1 is changed from M to H
f3 wants to wait for event 3
The priority of thread f3 is changed from M to H
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               20      0       *
*       4       random_1        READY   L               L               1110    0       *
*       5       random_2        READY   L               L               1110    0       *
*       1       f1              WAITING M               H               0       1110    *
*       2       f3              WAITING M               H               10      1100    *
*********************************************************************************
I found 65409.
random_1 changes the status of f1 to READY
The priority of thread f1 is changed from H to M
The priority of thread f1 is changed from M to L
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               15020   0       *
*       4       random_1        READY   L               L               16390   0       *
*       1       f1              READY   M               L               4910    14480   *
*       5       random_2        READY   L               L               16390   0       *
*       2       f3              WAITING M               H               10      22380   *
*********************************************************************************
The memeory space by random_1 has been released.
The memeory space by random_2 has been released.
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               18040   0       *
*       1       f1              READY   M               L               8380    14480   *
*       2       f3              WAITING M               H               10      28850   *
*********************************************************************************
```

Hw3 Example1

```
f1 wants to wait for event 3
The priority of thread f1 is changed from M to H
f3 wants to wait for event 3
The priority of thread f3 is changed from M to H
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       4       random_1        RUNNING L               L               320     0       *
*       5       random_2        READY   L               L               360     0       *
*       3       reclaimer       READY   L               L               60      0       *
*       1       f1              WAITING M               H               0       360     *
*       2       f3              WAITING M               H               10      350     *
*********************************************************************************
I found 65409.
random_2 changes the status of f1 to READY
The priority of thread f1 is changed from H to M
The priority of thread f1 is changed from M to L
The memeory space by random_2 has been released.
The memeory space by random_1 has been released.
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       1       f1              RUNNING M               L               52820   37520   *
*       3       reclaimer       READY   L               L               77740   0       *
*       2       f3              WAITING M               H               10      142530  *
*********************************************************************************
```

Hw3 Example1

```
f1 wants to wait for event 3
The priority of thread f1 is changed from M to H
f3 wants to wait for event 3
The priority of thread f3 is changed from M to H
I found 65409.
random_1 changes the status of f1 to READY
The priority of thread f1 is changed from H to M
The priority of thread f1 is changed from M to L
I found 65409.
random_2 changes the status of f3 to READY
I fell in love with the operating system.
f3 wants to wait for event 3
The memeory space by random_1 has been released.
The memeory space by random_2 has been released.
^Z
*****************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               3350    0       *
*       1       f1              READY   M               L               3200    320     *
*       2       f3              WAITING M               H               20      6200    *
*****************************************************************************
```

Hw3 Example2

{} init_threads.json > [ ] Threads > {} 1 > abc entry function

```
1   {
2       "Threads":[
3           {
4               "name" : "f4",
5               "entry function" : "Function4",
6               "priority": "M",
7               "cancel mode": "0"
8           },
9           {
10              "name" : "f5",
11              "entry function" : "Function5",
12              "priority": "M",
13              "cancel mode": "0"
14          }
15      ]
16  }
```

OS Lab @NCKU

Hw3 Example2

```c
void Function4(void)
{
    while(1)
    {
        OS2021_ThreadSetEvent(6);
        OS2021_ThreadWaitTime(1234);
        fprintf(stdout,"I found 65409.\n");
        fflush(stdout);
        OS2021_ThreadSetEvent(6);
        while(1);
    }
}
```

```c
void Function5(void)
{
    while(1)
    {
        OS2021_ThreadWaitEvent(6);
        fprintf(stdout,"I fell in love with the operating system.\n");
        fflush(stdout);
        OS2021_ThreadWaitTime(86400000);
    }
}
```

OS Lab @NCKU

Hw3 Example2

```
The priority of thread f4 is changed from M to H
f5 wants to wait for event 6
The priority of thread f5 is changed from M to H
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               20      0       *
*       1       f4              WAITING M               H               0       530     *
*       2       f5              WAITING M               H               10      520     *
*********************************************************************************
I found 65409.
f4 changes the status of f5 to READY
The priority of thread f4 is changed from H to M
I fell in love with the operating system.
The priority of thread f4 is changed from M to L
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               6030    0       *
*       1       f4              READY   M               L               6680    12340   *
*       2       f5              WAITING M               H               1010    24010   *
*********************************************************************************
^Z
*********************************************************************************
*       TID     Name            State   B_Priority      C_Priority      Q_Time  W_Time  *
*       3       reclaimer       RUNNING L               L               9030    0       *
*       1       f4              READY   M               L               9880    12340   *
*       2       f5              WAITING M               H               1010    30210   *
*********************************************************************************
^C
```

References

1. ucontext
   - The Open Group Library
   - IBM® IBM Knowledge Center
     - getcontext()
     - setcontext()
     - makecontext()
     - swapcontext()

2. signal handler
   - Gitbook
   - Linux manual page

3. timer
   - Linux manual page
   - IBM® IBM Knowledge Center

4. .json
   - JSON Introduction