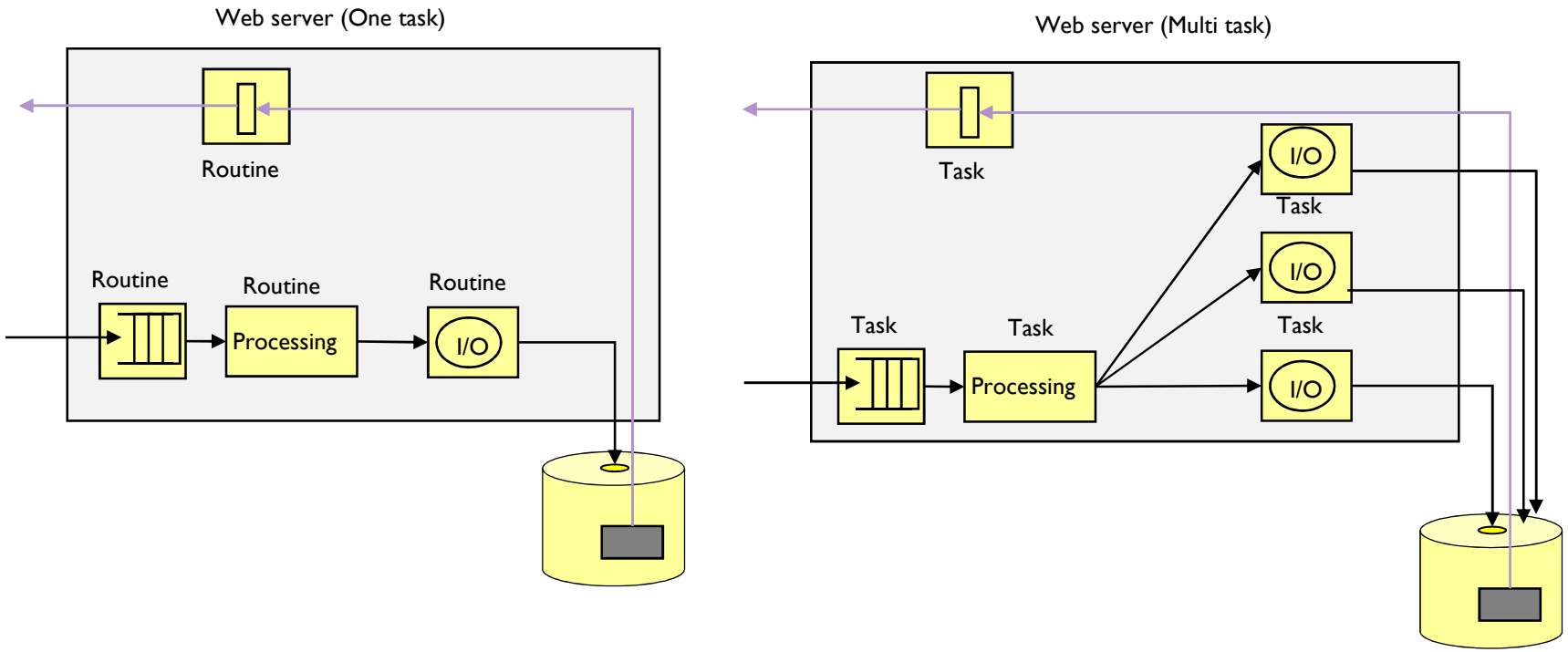
KWANGWOON UNIVERSITY

Tasks

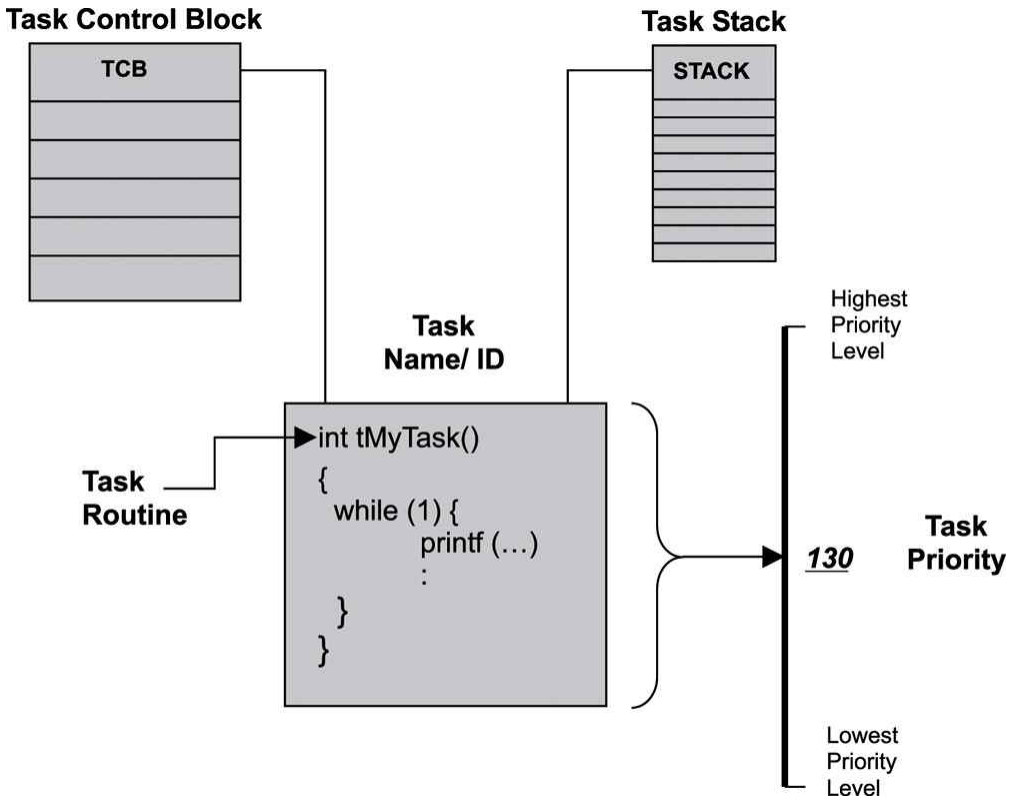
Concurrency design

- **An application is decomposed into small, schedulable, and sequential program units.**
  - Tasks
- **Allows system multitasking to meet performance and timing requirements.**



Defining a task

- **Task**
  - An independent thread of execution that can compete with other concurrent tasks.



Defining a task

- **System tasks**
  - When the kernel first starts,
    - it creates some system tasks, and
    - allocates the appropriate priorities for them.

Defining a task

- **Examples of system tasks**
  - Initialization or startup task
    - Initializes the system, and creates and starts system tasks.
  - Idle task
    - Uses up processor idle cycles when no other tasks.
    - Why idle task?
      - Processor executes the instruction to which PC points.
      - PC must still point to valid instructions even when no tasks.
    - User routine for power saving can be used instead.
  - Logging task
    - Logs system messages
  - Exception-handling task
    - Handles exception
  - Debug agent task
    - Allows debugging with a host debugger.

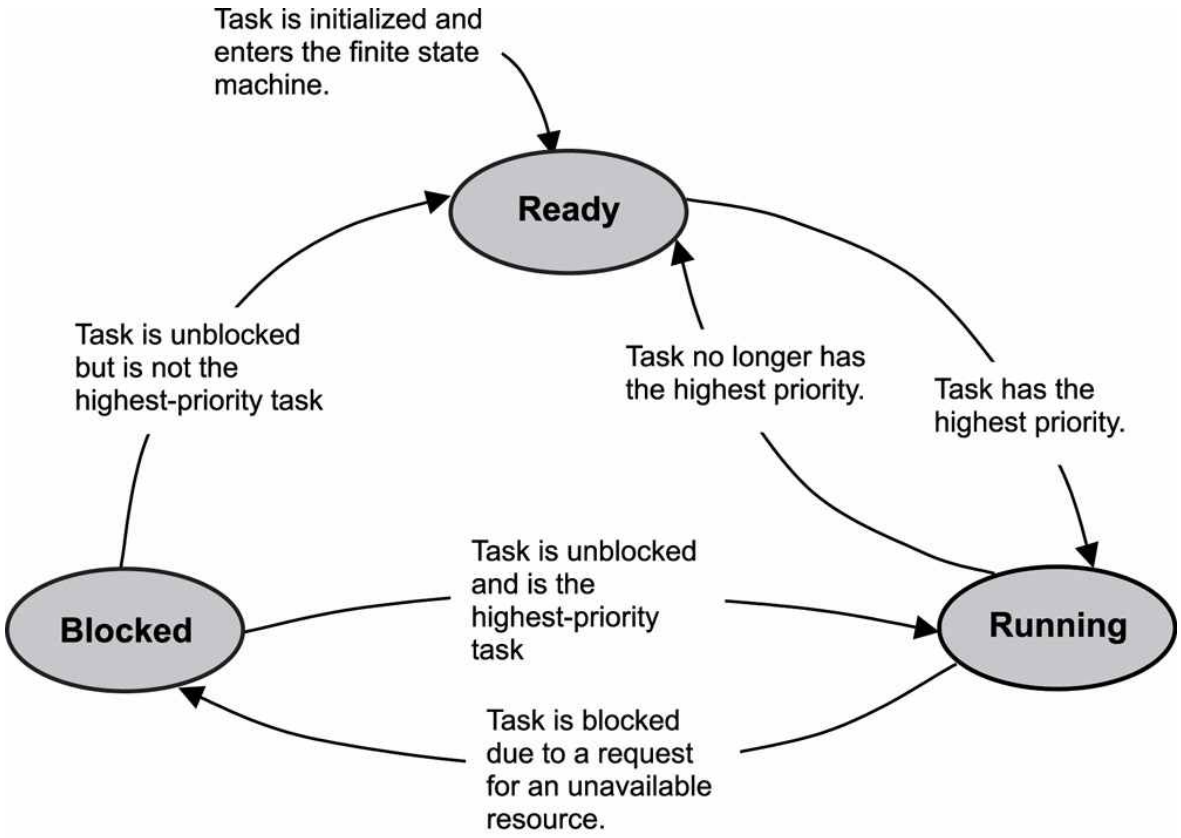
Task states and scheduling

- **Each task exists in one of the following states.**
  - Ready
    - The task is ready to run, but cannot run because a higher priority task is executing.
  - Running
    - The task is the highest priority task and is running.
  - Blocked
    - The task has requested a resource that is not available, has to wait until the requested resource is available.

- **Other RTOSs may have more different states.**
  - VxWorks has suspended, pended, and delayed states.

Task states and scheduling

- **A typical finite state machine for task execution states**



KWANGWOON UNIVERSITY

Typical task operations

- **Kernel provides**
  - Task management services
    - Create and maintain the TCB and task stacks.
  - APIs
    - Create and delete tasks
    - Control task scheduling
    - Obtain task information

Typical task operations

- **Task creation**
  - Two types of task creation
    - One phase execution
    - Two phase execution

- **One phase execution**
  - A newly created task is started as soon as applications call a system call for the task creation.

- **Two phase execution**
  - Phase 1: a new task is created, but suspended. It is first created and put into a suspended state.
  - Phase 2: the task is moved to the ready state when it is started.

Typical task operations

- **Task deletion**
  - During the deletion process,
    - kernel terminates the tasks, and
    - frees memory by deleting the task's TCB and stack.
  - In many RTOS kernels, they allow a task to delete any other task.
    - It could be hazardous.
  - If a task is deleted incorrectly?
    - The task might not release resources.
  - Premature deletion can result in memory leak or resource leaks.
    - Many kernels provide task-deletion locks.
      - Not deleted during a critical section.

Task scheduling

- **Task scheduling**
  - Kernel scheduling done by kernel automatically.
  - Manual scheduling done by user through calling special APIs.

operation	description
Suspend	suspends a task.
Resume	resumes a task.
Delay	delays a task.
Restart	restarts a task.
Get priority	gets the current task's priority.
Set priority	sets a task's priority.
Preemption lock	locks out higher priority tasks from preempting the current task.
Preemption unlock	unlocks a preemption lock.

Task scheduling

- **Suspend and resume tasks**
  - To debug
  - To suspend a high-priority task so that lower priority tasks can execute.

- **Delay a task**
  - allow manual scheduling or wait for an external condition.
  - relinquish the CPU and allow another task to execute.
  - After the delay expires, the task is returned to the task-ready list.

Task scheduling

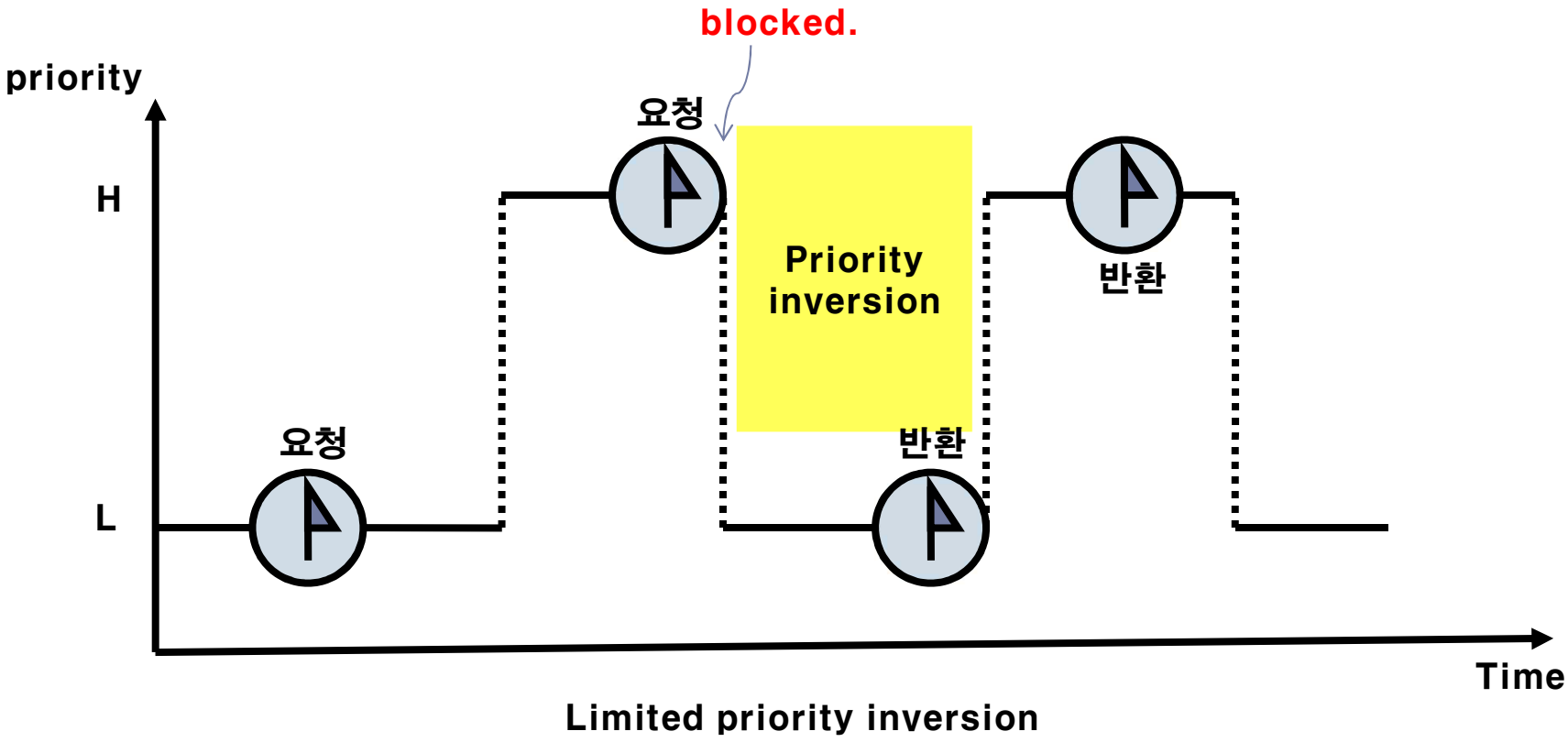
- **Restart a task**
  - Begins the task as if it had not been previously executing.
  - The internal state is lost when a task is restarted.
    - Compare with "resume"
- **Get/set priority**
  - Control task scheduling manually during execution.
  - Helpful during a priority inversion.
- **Preemption lock/unlock**
  - Can be useful if a task execute a critical section
  - The task must not be preempted by other tasks.

Priority inversion

- **Priority inversion**
  - A low priority task executes while a higher priority task waits on it
    - due to resource contentions.

Priority inversion

- Example 1

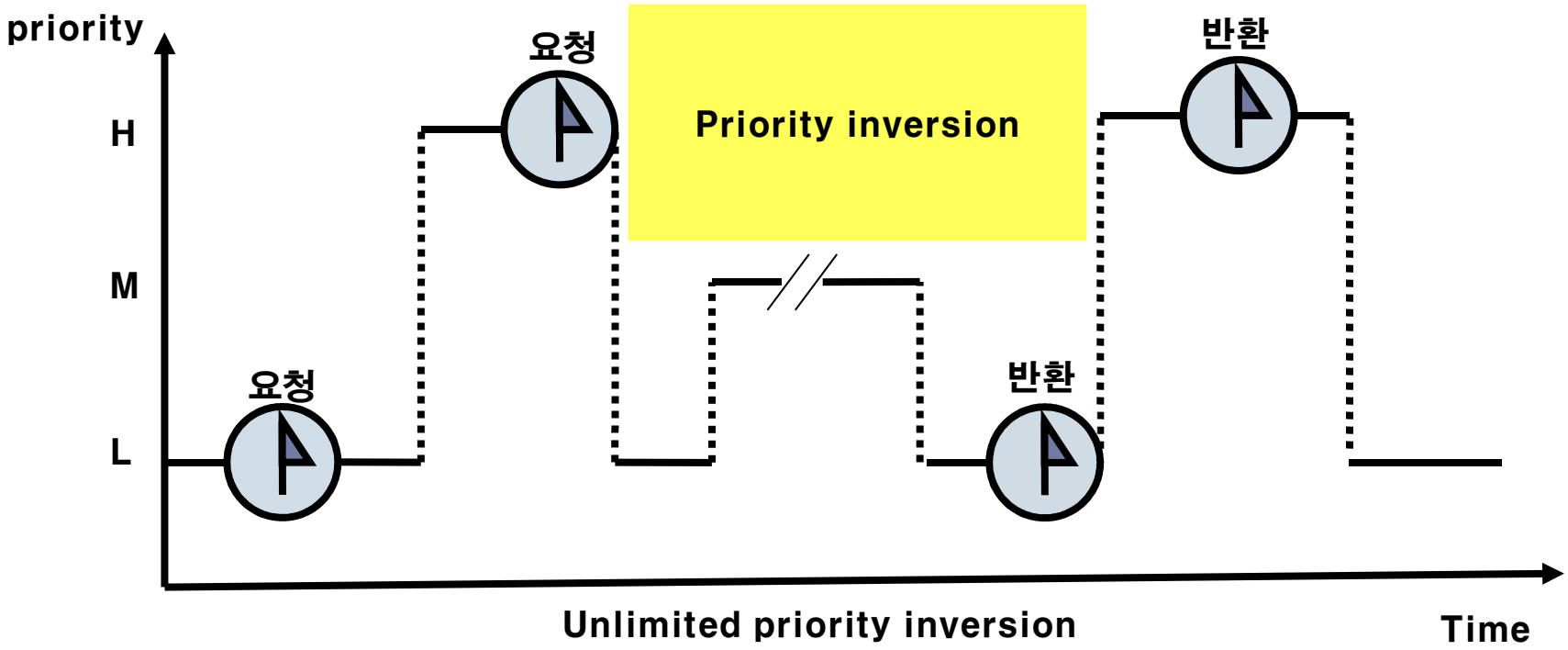


Limited priority inversion

Due to resource contention, priority inversion is often occurred.

Priority inversion

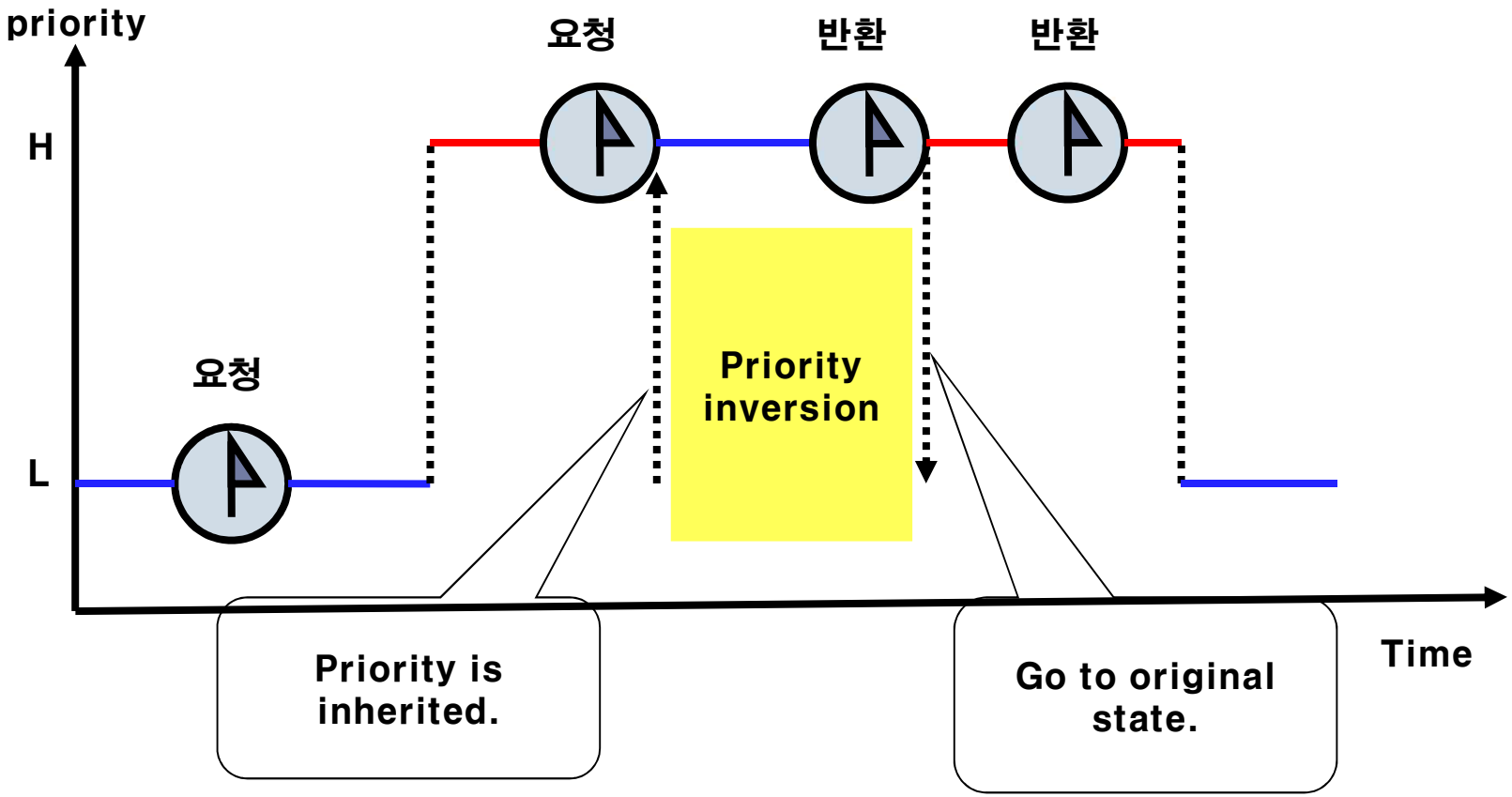
- Example 2



High task can be infinitely postponed.

Priority inversion

- Priority Inheritance Protocol



KWANGWOON UNIVERSITY

Obtaining task information

- **Task information operations**
  - Allow developers to access information within their applications.

Operation	Description
Get ID	Get the current task's ID
Get TCB	Get the current task's TCB

KWANGWOON UNIVERSITY

Typical task structure

- **Tasks are structured in one of two ways**
  - Run-to-completion
  - Endless loop

Typical task structure

- **Run-to-completion task**
  - useful for initialization and startup.
  - They typically run once, when the system first powers on.
  - E.g.

```
RunToCompletion Task() {
        Initialize application
        Create 'endless loop tasks'
        Create kernel objects
        Delete or suspend this task
}
```

Typical task structure

- **Endless-loop tasks**
  - Run many times while the system is powered on.
  - One or more blocking calls within the body of the loop.
  - E.g.

```
EndlessLoop Task() {
        Initialization code
        Loop forever {
                Body of loop
                Make one or more blocking calls
        }
}
```

Synchronization, communication, and concurrency

KWANGWOON UNIVERSITY

- **Inter-task primitives**
  - Kernel objects that facilitate synchronization and communication between two or more tasks.
  - Example of inter-task primitives
    - Semaphores
    - Message queues
    - Signals
    - Pipes
    - Other types of objects

Case study: uCOS-II

Start with uCoS-II

A simple code example

```
#include “includes.h”

OS_STK     MyTaskStack[1024];                                    // (1)

void MyTask(void *pdata);

int main(void)
{
    OSInit();                                                    // (2)
    OSTaskCreate(MyTask, NULL, &MyTaskStack[1023], 10);          // (3)
    OSStart();                                                   // (4)
     return 0;
}

void MyTask(void *pdata)                                         // (5)
{
    printf("hello world₩n");                                     // (6)

    while (1);                                                   // (7)
}
```

Typical uC/OS-II code

- Typical uC/OS-II code

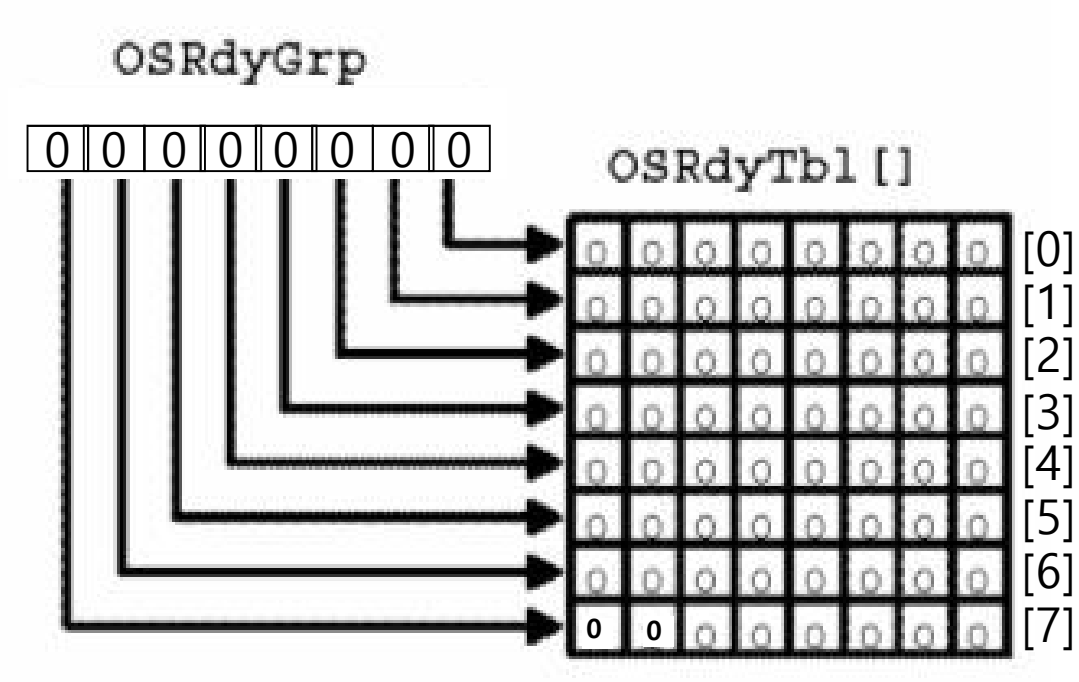
```
void main (void)
{
    OSInit();          /* Initialize uC/OS-II    */
    .
    .
    /* Create at least 1 task using OSTaskCreate(); */
    .
    .
    OSStart();         /* Start multitasking! OSStart() will not return */
}
```

uC/OS-II 초기화

- **반드시 OSInit() 함수를 호출하여 uC/OS-II를 초기화해야 함.**
  - OSInit() - OS_CORE.C/Kernel

OSInit()

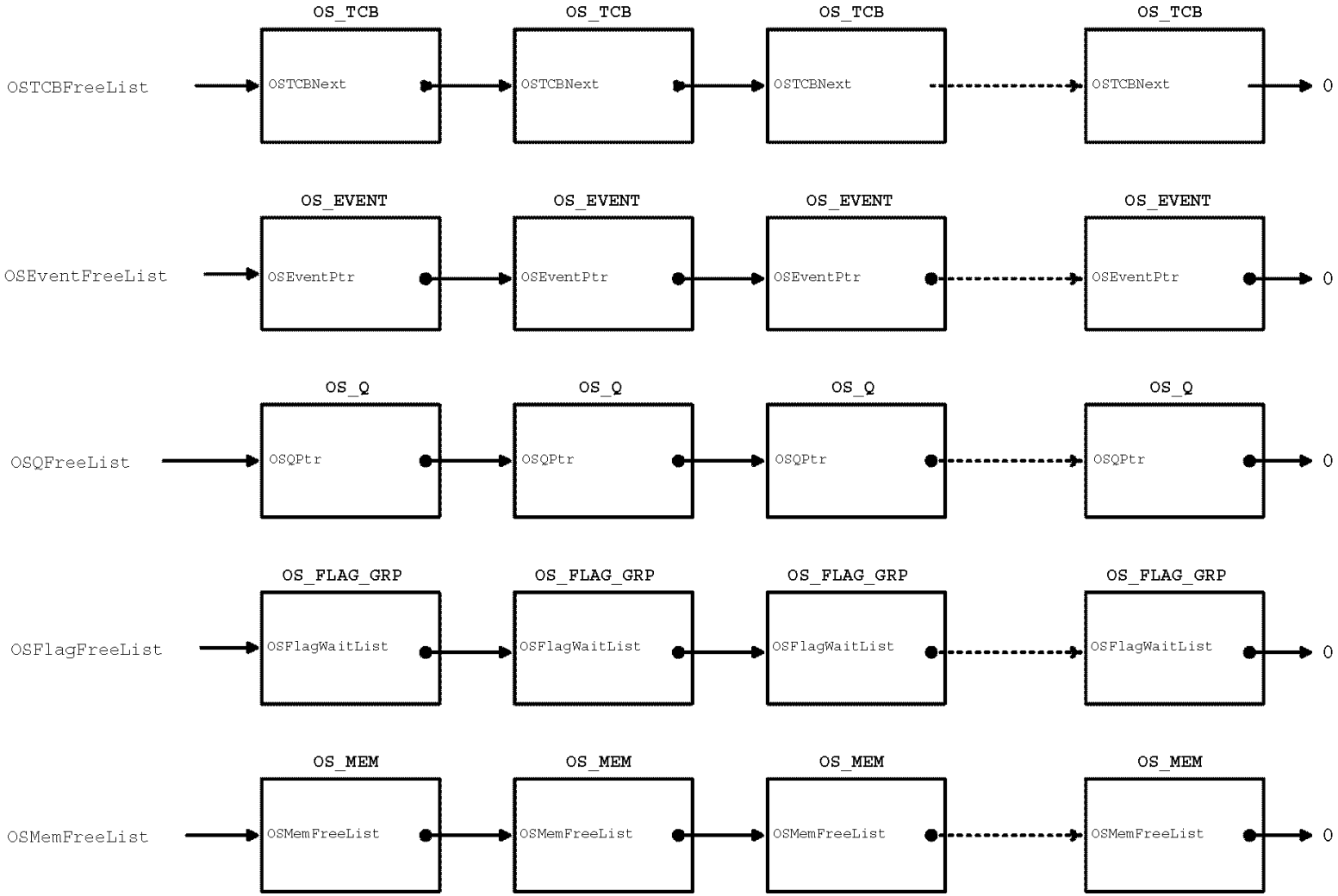
- **OSRdyGrp 및 OSRdyTbl 배열 초기화**
  - task의 우선순위를 이용하여, ready 상태의 task 를 표시함.
  - task의 우선순위는 unique 함에 유의.



OSInit()

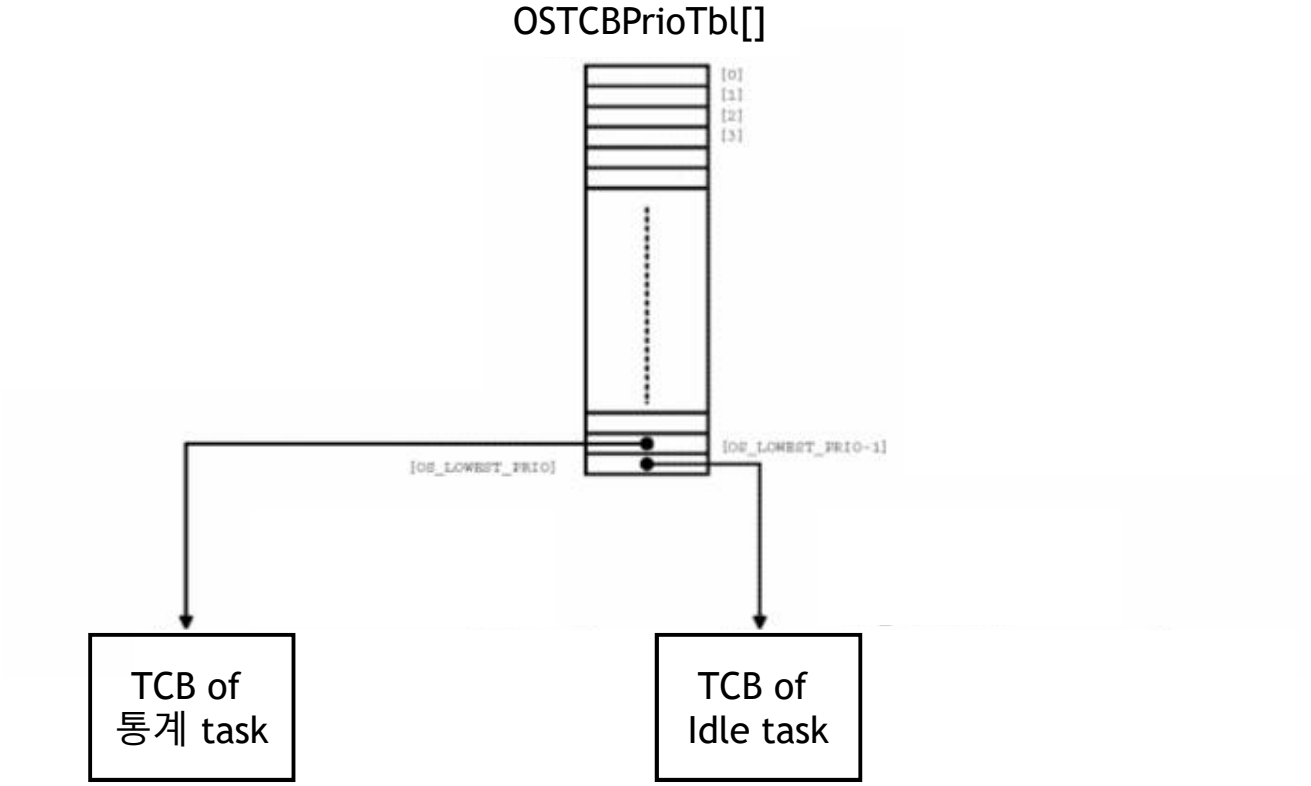
- **OSTCBFreeList, OSEventFreeList, OSQFreeList, OSFlagFreeList, OSMemFreeList 등을 초기화**
  - doubly linked list 연결
  - 위의 객체는 동적 생성/소멸을 수행하지 않고, 미리 free pool을 생성하여 사용함.
  - 가령 task가 최대 64개 이므로 TCB의 free list는 64개의 free TCB를 연결하면 됨.

OSInit()


OS_TCB
OS_TCB
OS_TCB
OS_TCB
OSTCBFreeList
OSTCBNext
OSTCBNext
OSTCBNext
OSTCBNext
0
OS_EVENT
OS_EVENT
OS_EVENT
OS_EVENT
OSEventFreeList
OSEventPtr
OSEventPtr
OSEventPtr
OSEventPtr
0
OS_Q
OS_Q
OS_Q
OS_Q
OSQFreeList
OSQPtr
OSQPtr
OSQPtr
OSQPtr
0
OS_FLAG_GRP
OS_FLAG_GRP
OS_FLAG_GRP
OS_FLAG_GRP
OSFlagFreeList
OSFlagWaitList
OSFlagWaitList
OSFlagWaitList
OSFlagWaitList
0
OS_MEM
OS_MEM
OS_MEM
OS_MEM
OSMemFreeList
OSMemFreeList
OSMemFreeList
OSMemFreeList
OSMemFreeList
0

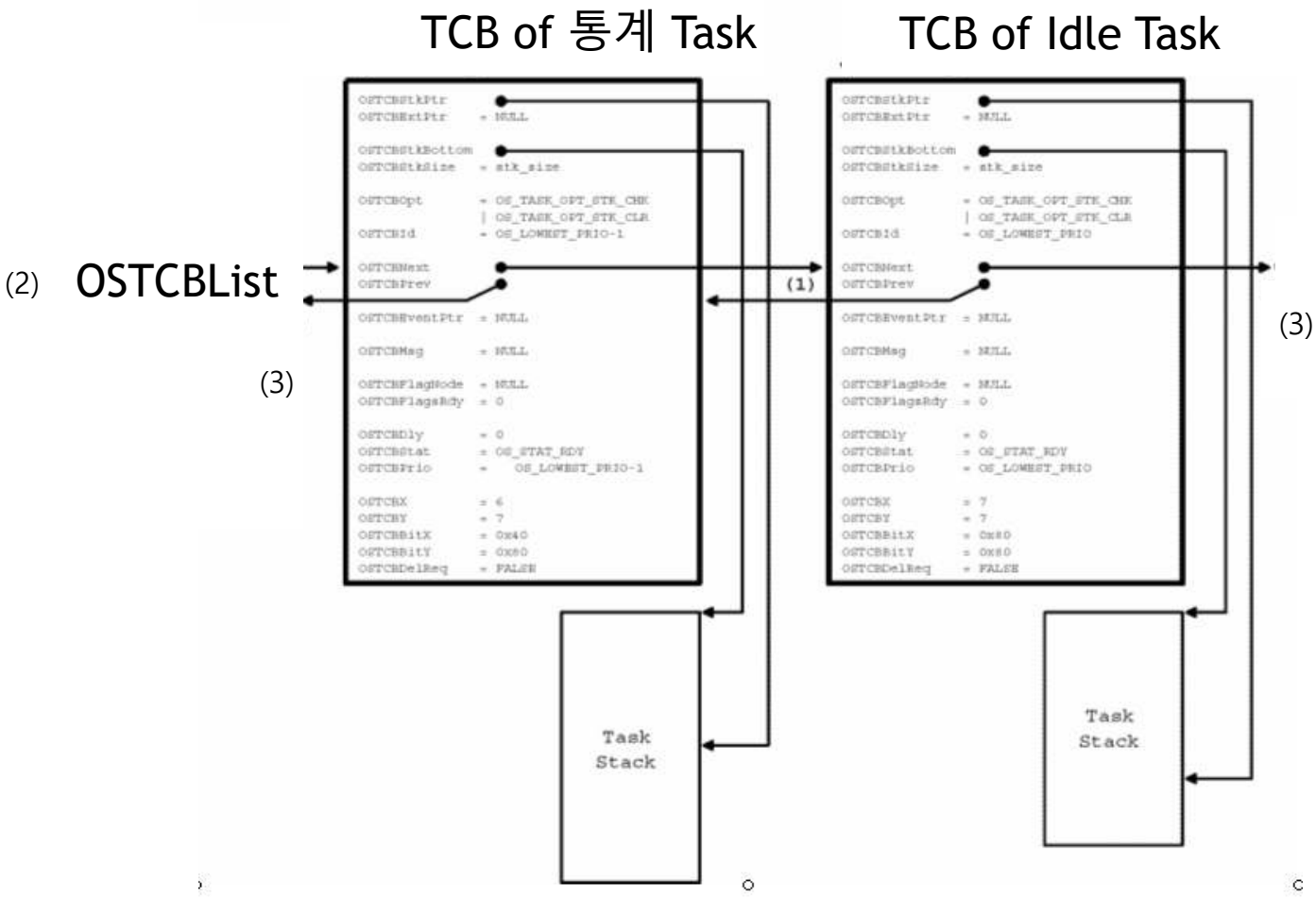
OSInit()

- **IDLE Task – OS_TaskIdle()을 생성**
  - 가장 낮은 우선순위
- **환경 설정에 따라, 통계 태스크 – OS_TaskStat() 생성**
  - IDLE Task보다 1 높은 우선순위



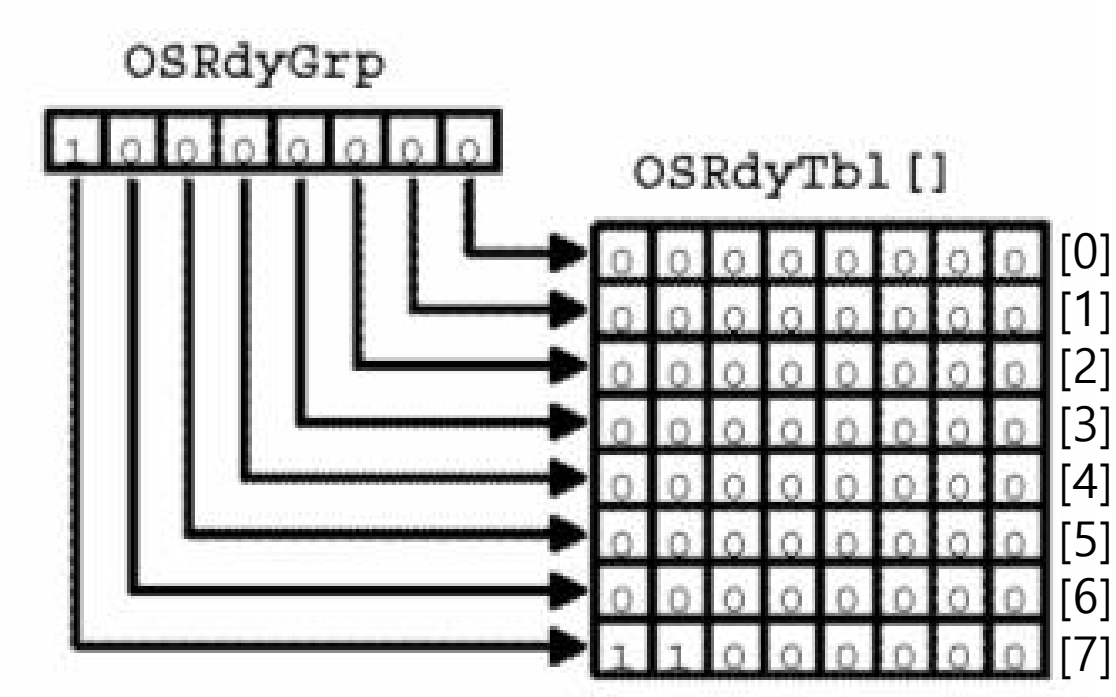
OSInit()

- **OSTCBList 초기화**
  - 생성된 태스크들의 TCB를 doubly linked list로 연결

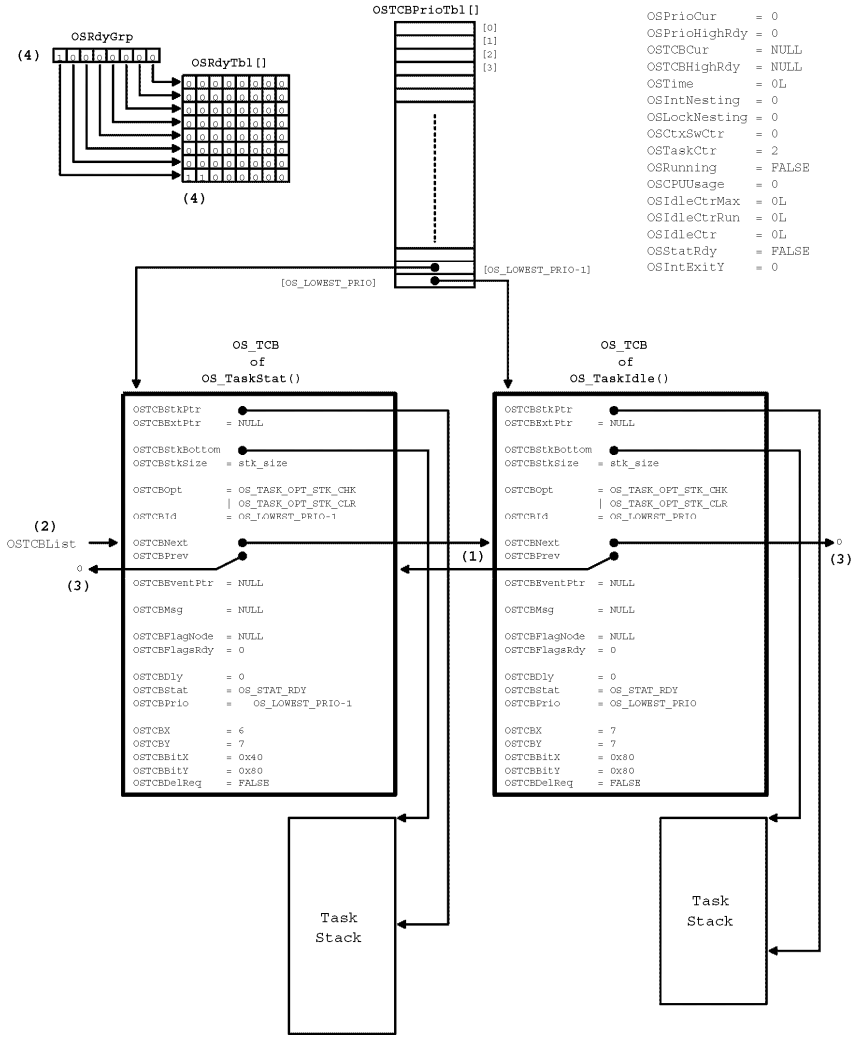


OSInit()

- OSRdyGrp 및 OSRdyTbl 배열 update
  - task의 우선순위를 이용하여, ready 상태의 task 를 표시함 (task의 우선순위는 unique 함에 유의)



OSInit() 완료

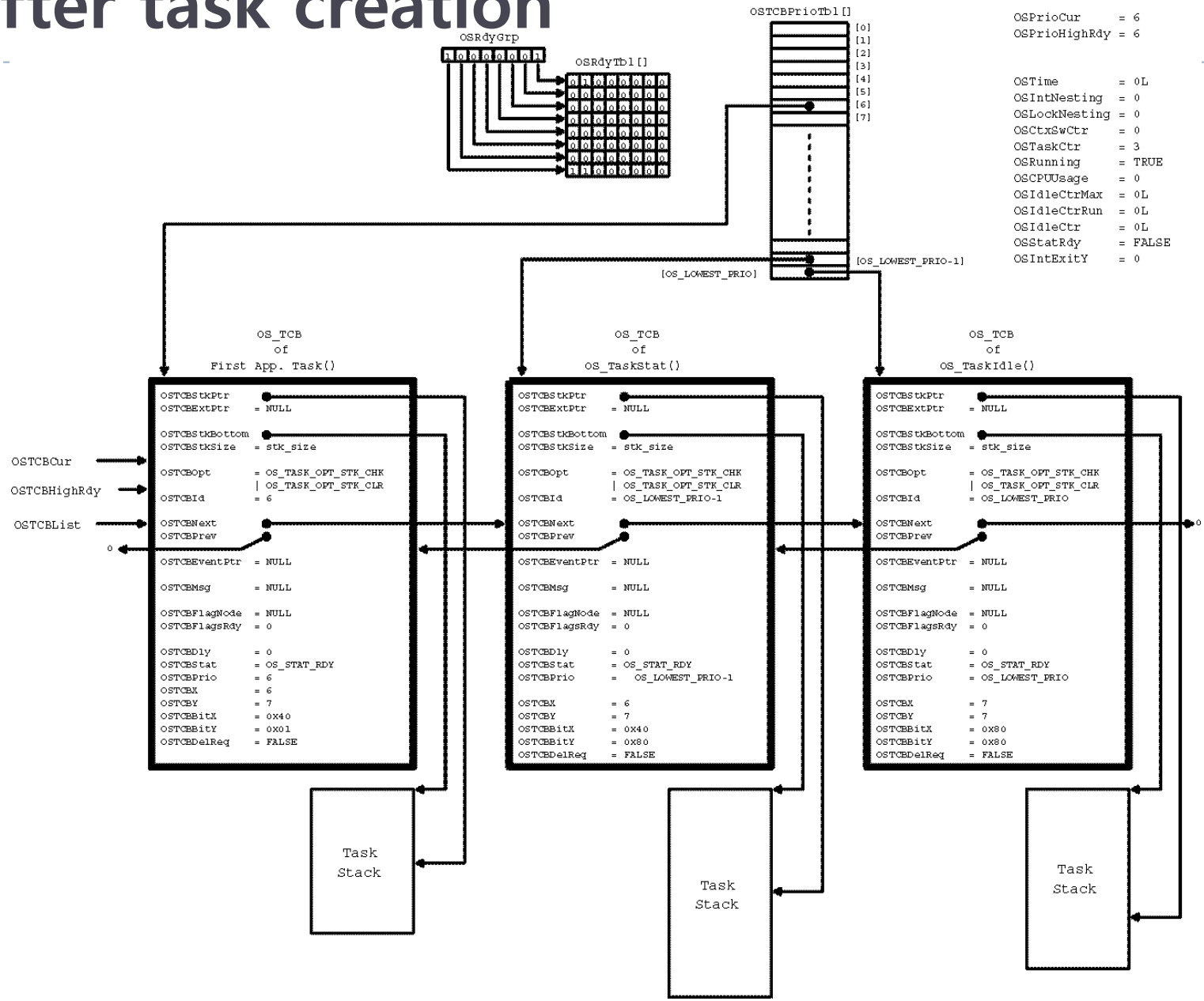

OSRdyGrp
(4)
OSRdyTbl[]
(4)
OSTCBPrioTbl[]
[0]
[1]
[2]
[3]
[OS_LOWEST_PRIO]
[OS_LOWEST_PRIO-1]
OSPrioCur = 0
OSPrioHighRdy = 0
OSTCBCur = NULL
OSTCBHighRdy = NULL
OSTime = 0L
OSIntNesting = 0
OSLockNesting = 0
OSCtxSwCtr = 0
OSTaskCtr = 2
OSRunning = FALSE
OSCPUUsage = 0
OSIdleCtrMax = 0L
OSIdleCtrRun = 0L
OSIdleCtr = 0L
OSStatRdy = FALSE
OSIntExitY = 0
OS_TCB of OS_TaskStat()
OS_TCB of OS_TaskIdle()
(2)
OSTCBList
(3)
(1)
(3)
Task Stack
Task Stack

Task creation

- OSStart()를 호출하기 전에, 적어도 1개 이상의 응용 프로그램 태스크를 생성해야 함

```
void main (void)
{
    OSInit();          /* Initialize uC/OS-II    */
    .
    .
    /* Create at least 1 task using OSTaskCreate(); */
    .
    .
    OSStart();         /* Start multitasking! OSStart() will not return */
}
```

After task creation



uC/OS-II 시작

- uC/OS-II는 OSInit() 이후에, OSStart()를 호출하면서 시작됨.

```
void main (void)
{
    OSInit();          /* Initialize uC/OS-II    */
    .
    .
    /* Create at least 1 task using OSTaskCreate(); */
    .
    .
    OSStart();         /* Start multitasking! OSStart() will not return */
}
```

OSStart() – OS_CORE.C/Kernel

```
void OSStart (void)
{
    INT8U y;
    INT8U x;

    if (OSRunning == FALSE) {
        y             = OSUnMapTbl[OSRdyGrp];
        x             = OSUnMapTbl[OSRdyTbl[y]];
        OSPrioHighRdy = (INT8U)((y << 3) + x);
        OSPrioCur     = OSPrioHighRdy;
        OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
        OSTCBCur      = OSTCBHighRdy;
        OSStartHighRdy();
         //TCB로부터 태스크의 스택 위치를 찾아 스택으로부터 레지스터들을 복구

    }
}
```

- 준비 리스트에서 우선순위가 가장 높은 실행 가능한 태스크를 찾는다. (how?)
- OSTCBCur를 우선순위가 가장 높은 실행 가능한 태스크의 TCB로 설정.
- OSStartHighRdy() 실행

KWANGWOON UNIVERSITY

OSRdyGrp = 01101000

OSRdyTbl[3] = 11100100

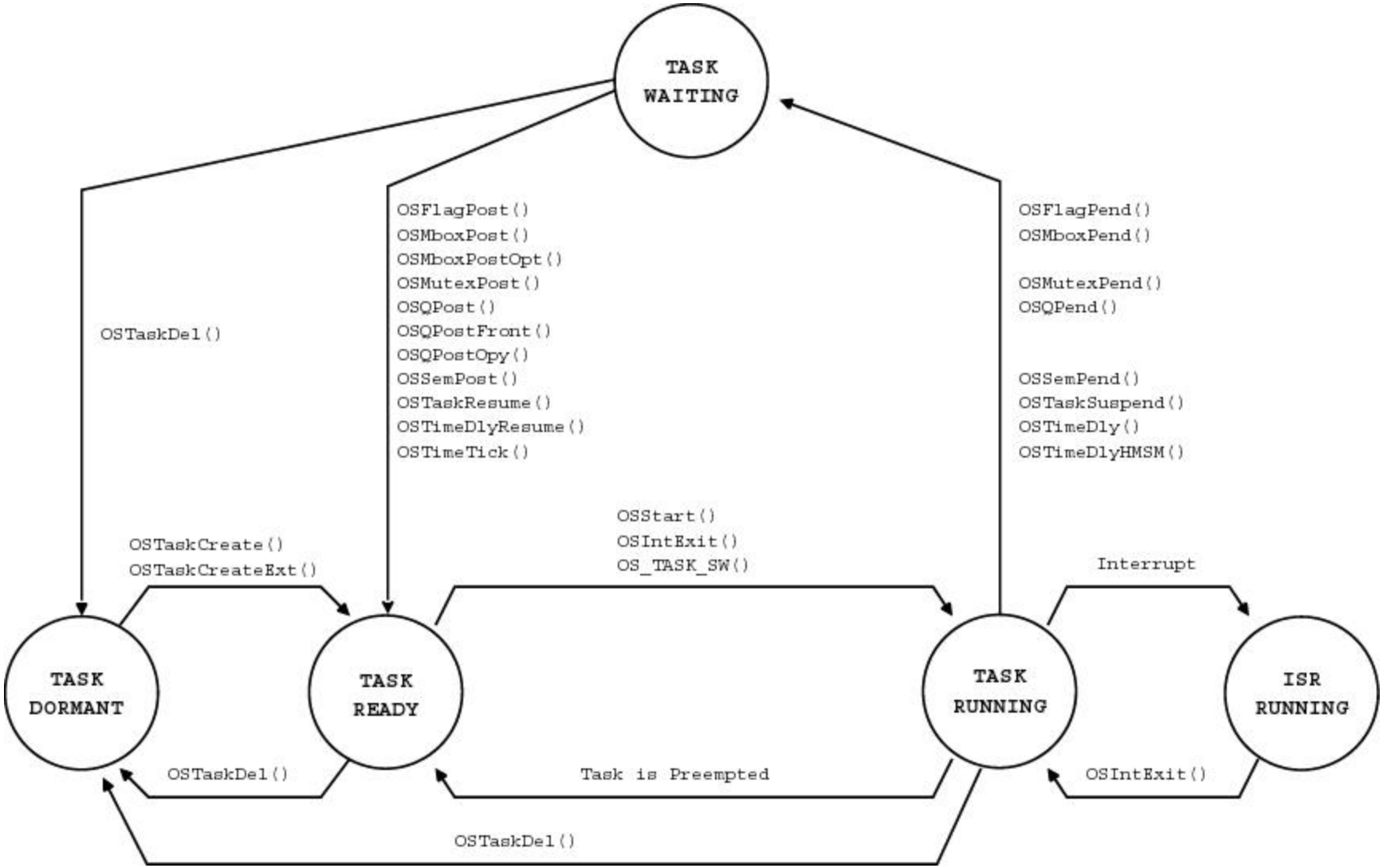
```
INT8U  const  OSUnMapTbl[256] = {
    0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x00 to 0x0F  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x10 to 0x1F  */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x20 to 0x2F  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x30 to 0x3F  */
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x40 to 0x4F  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x50 to 0x5F  */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x60 to 0x6F  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x70 to 0x7F  */
    7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x80 to 0x8F  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0x90 to 0x9F  */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xA0 to 0xAF  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xB0 to 0xBF  */
    6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xC0 to 0xCF  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xD0 to 0xDF  */
    5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,      /* 0xE0 to 0xEF  */
    4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0       /* 0xF0 to 0xFF  */
};
```

3 = OSUnMapTbl [0x68];
2 = OSUnMapTbl [0xE4];
26 = (3 << 3) + 2;

Case study: uCOS-II

Task

Task State Transition



Task Implementation

- 무한루프 또는 임무 종료 후 스스로를 삭제하는 형태

```
void YourTask (void *pdata)
{
   for (;;){
      /* USER CODE */
      Call one of uC/OS-II's services:
      OSFlagPend();
      OSMboxPend();
      OSMutexPend();
      OSQPend();
      OSSemPend();
      OSTaskDel(OS_PRIO_SELF);
      OSTaskSuspend(OS_PRIO_SELF);
      OSTimeDly();
      OSTimeDlyHMSM();
      /* USER CODE */
   }
}
```

(1)

```
void YourTask (void *pdata)
{
   /* USER CODE */
   OSTaskDel(OS_PRIO_SELF);
}
```

(2)

TCB - uCOS_II.H/Kernel

```
typedef struct os_tcb {

    OS_STK         *OSTCBStkPtr;

    struct os_tcb *OSTCBNext;
    struct os_tcb *OSTCBPrev;

    INT16U          OSTCBDly;
    INT8U           OSTCBStat;
    INT8U           OSTCBPrio;
                                    /* scheduling 시 계산시간을 절약하기 위해 미리 계산*/
    INT8U           OSTCBX;         .OSTCBY = priority >> 3;
    INT8U           OSTCBY;         .OSTCBBitY = OSMapTbl[priority >> 3];
    INT8U           OSTCBBitX;      .OSTCBX = priority & 0x07;
    INT8U           OSTCBBitY;      .OSTCBBitX = OSMapTbl[priority & 0x07];

} OS_TCB;
```

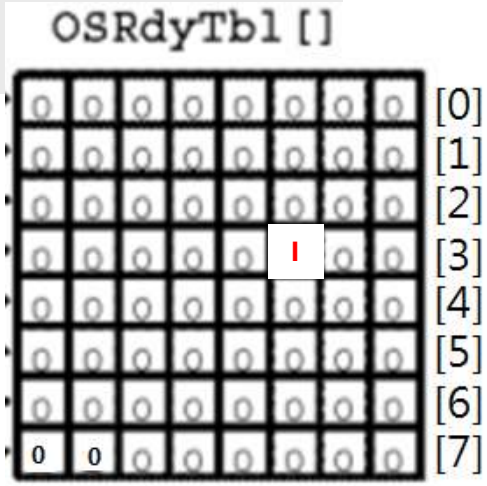
e.g. If prio = 26, binary is 00011010.

OSTCBY = 3

OSTCBBitY = OSMapTbl[3] = 00001000

OSTCBX = 2

OSTCBBitX = OSMapTbl[2] = 00000100



TCB - uCOS_II.H/Kernel

- **OSMapTbl[]**

index	Bit mask (binary)
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

- If prio = 26, binary is 00011010.
  - OSMapTbl[3] = 00001000, OSMapTbl[2] = 00000100

KWANGWOON UNIVERSITY

Task Priority & ID

- **64개의 우선순위 (0-63)**
  - 0 > 1 > 2 > … > 63
- **각 태스크는 고유한 우선순위를 부여 받아야 함**
  - 태스크의 우선순위를 태스크를 구별하는 ID로 사용
- **가장 높은 우선순위 4개와 가장 낮은 우선순위 4개는 예약됨**
  - 실제로는 가장 낮은 우선순위 2개만 사용 중 (idle task & 통계 task)

OSTCBStat

- **Bit definition for OSTCBStat (uCOS_II.h)**
  - #define OS_STAT_RDY 0x00 /* Ready to run */
  - #define OS_STAT_SEM 0x01 /* Pending on semaphore */
  - #define OS_STAT_MBOX 0x02 /* Pending on mailbox */
  - #define OS_STAT_Q 0x04 /* Pending on queue */
  - #define OS_STAT_SUSPEND 0x08 /* Task is suspended */
  - #define OS_STAT_MUTEX 0x10 /* Pending on MUTEX */
  - #define OS_STAT_FLAG 0x20 /* Pending on event flag group */

KWANGWOON UNIVERSITY

OSTCBFreeList & OSTCBList

- **OSTCBFreeList - uCOS_II.H/Kernel**
  - 사용되고 있지 않은 TCB들을 연결하는 리스트 헤더
  - 시스템 초기에는 64개의 TCB들을 연결
  - Task가 생성될 때마다, OSTCBFreeList에서 한 개의 TCB를 떼어 내어 사용함
  - 64개의 Task가 모두 생성되면 empty list가 됨

- **OSTCBList - uCOS_II.H/Kernel**
  - 생성된 태스크의 TCB들을 연결하는 리스트 헤더
  - 시스템 초기에는 empty list.
  - 태스크가 생성될 때, 해당 TCB를 추가로 연결함
  - 태스크가 종료되면, 해당 TCB를 제거하고, OSTCBFreeList에 추가함

태스크 생성

- **OSTaskCreate – OS_TASK.C/Kernel**

```
INT8U OSTaskCreate(     void        (*task) (void *pd),
                        void        *pdata,
                        OS_STK      *ptos,
                        INT8U       prio);
```

  - task : 실행할 태스크 코드의 시작 위치를 가리키는 포인터
  - pdata : 태스크 코드에 대한 전달인자 포인터
  - ptos : 태스크 스택의 시작 주소 포인터 (스택의 확장 방향에 유의해야 함)
  - prio : 태스크의 우선 순위

태스크 생성 예제

```
#include “includes.h”

OS_STK     MyTaskStack[1024];                                    // (1)

void MyTask(void *pdata);

int main(void)
{
    OSInit();                                                    // (2)
    OSTaskCreate(MyTask, NULL, &MyTaskStack[1023], 10);          // (3)
    OSStart();                                                   // (4)
   return 0;
}

void MyTask(void *pdata)                                         // (5)
{
    printf("hello world₩n");                                     // (6)

    while (1);                                                   // (7)
}
```

태스크 우선순위 변경

- **OSTaskChangePrio– OS_TASK.C/Kernel**

  INT8U OSTaskChangePrio(INT8U oldprio,
  INT8U newprio);

  - oldprio : 변경할 태스크의 우선순위
    - IDLE task를 제외한, 임의의 태스크의 우선순위를 바꾸는 것이 가능
  - newprio : 새로운 우선 순위

  - 새로운 우선순위는 할당되지 않은 상태이어야 한다. 그렇지 않은 경우, 에러 반환

Suspending a Task

- **OSTaskSuspend – OS_TASK.C/Kernel**

  INT8U OSTaskSuspend(INT8U prio);

  - prio : suspend할 태스크의 우선순위
    - IDLE task를 제외한, 임의의 태스크를 중지하는 것이 가능

  - 태스크의 실행을 일시적으로 중단함. OS_PRIO_SELF를 전달인자로 호출하면 현재 수행중인 태스크를 중단함.
  - OSTaskResume() 함수를 통해서만, 수행이 재개됨.
  - IDLE 태스크는 중단될 수 없음.

Resuming a Task

- **OSTaskResume – OS_TASK.C/Kernel**

  INT8U OSTaskResume(INT8U prio);

  - prio : resume할 태스크의 우선순위
    - IDLE task를 제외한, 임의의 태스크의 수행을 재개하는 것이 가능

  - OSTaskSuspend()에 의해 중지된 태스크의 실행을 재개

Getting Information about a Task

- **OSTaskQuery– OS_TASK.C/Kernel**

```
INT8U OSTaskQuery (     INT8U       prio,
                        OS_TCB*     pdata);
```

  - prio : 정보를 구할 태스크의 우선순위
    - prio로 OS_PRIO_SELF가 넘어온 경우에는, 현재 수행 중인 태스크의 정보를 구하라는 의미
  - pdata : OS_TCB 정보를 복사할 메모리 영역에 대한 포인터

  - 응용 프로그램은 TCB를 복사할 메모리 영역을 할당한 후, OSTaskQuery()를 호출해야 함
  - TCB의 내용을 인자로 전달한 pdata에 복사

태스크 삭제

- **OSTaskDel – OS_TASK.C/Kernel**

  INT8U OSTaskDel (INT8U prio);

  - prio : 삭제할 태스크의 우선 순위

  - 스스로를 삭제하기 위해서는 prio에 OS_PRIO_SELF를 넘김
  - 삭제된 task는 수면 상태로 들어감
  - 수면 상태의 task는 이후 OSTaskCreate() 함수를 호출해야 다시 활성화됨

Example codes

Task

문제

- **매 클락 틱마다, “Hello world” 메시지를 반복 출력하는 응용 프로그램을 작성하시오.**
  - uC/OS-II에서 임의의 클락 틱 동안 태스크 수행을 중단하는 함수는 OSTimeDly() 이다. 이 함수를 이용하도록 한다.
  - OSTimeDly(1) – 1 클락 틱 동안 태스크 수행 중단

정답

```
#include “includes.h”

OS_STK     MyTaskStack[1024];                                          // (1)

void MyTask(void *pdata);

int main(void)
{
    OSInit();                                                          // (2)
    OSTaskCreate(MyTask, NULL, &MyTaskStack[1023], 10);        // (3)
    OSStart();                                                         // (4)
   return 0;
}
void MyTask(void *pdata)                                               // (5)
{
    while (1)
    {
            printf("hello world₩n");                                   // (6)
            OSTimeDly(1);                                              // (7)
    }
}
```

문제

- **이전 프로그램을 다음과 같이 수정하시오.**
- **10 태스크와 20태스크가 교대로 자신의 메시지를 출력하도록 수정한다.**

- **수행 결과**
  - 10 task
  - 20 task
  - 10 task
  - 20 task
  - …

정답

```
#include “includes.h”

OS_STK     MyTaskStack[1024];                                    // (1)
OS_STK     MyChildStack[1024];

void MyTask(void *pdata);
void MyChildTask(void *pdata);

int main(void)
{
    OSInit();                                                    // (2)
    OSTaskCreate(MyTask, NULL, &MyTaskStack[1023], 10);          // (3)
    OSStart();                                                   // (4)
  return 0;
}
```

정답

```
void MyTask(void *pdata)
{
    OSTaskCreate(MyChildTask, (void *) 20, &MyChildStack[1023], 20);
                                                                    // (5)
    while (1)
    {
            printf("10 task\n”);                                    // (6)
            OSTaskSuspend(10);                                      // (7)
    }
}

void MyChildTask(void *pdata)
{
    while (1)
    {
            printf(“20 task\n”);                                    // (8)
            OSTaskResume(10);                                       // (9)
    }
}
```