실습 4

MicroC/OS-II: 스케줄러 이해하기

Outline

- Overview
- What to do?
  - 숫자 게임

Overview

- 목적
  - Microc/os II O(1) 스케줄러 이해하기
  - 지난 실습 시 작성한 코드를 활용할 것

- 구체적으로, 다음의 data structure 이해하기
  - Data structure
    - OSRdyGrp
      - Bit indicating group
    - OSRdyTbl[]
      - 8 task in the same group
    - OSMapTbl[]
      - Array used for bit mask
    - OSUnMapTbl[]
      - Array used to search the highest priority in ready list
  - (os_core.c 참조)

- 비트맵

OSMapTbl[]

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

OSUnMapTbl[]

```
INT8U const OSUnMapTbl[] = {
   0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
};
```

Binary									Hexadecimal
OSRdyGrp									
	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	0x00
OSRdyTbl[8]									
	7	6	5	4	3	2	1	0	
[0]	7	6	5	4	3	2	1	0	0x00
[1]	15	14	13	12	11	10	9	8	0x00
[2]	23	22	21	20	19	18	17	16	0x00
[3]	31	30	29	28	27	26	25	24	0x00
[4]	39	38	37	36	35	34	33	32	0x00
[5]	47	46	45	44	43	42	41	40	0x00
[6]	55	54	53	52	51	50	49	48	0x00
[7]	63	62	61	60	59	58	57	56	0x00

Create 4 tasks (36 → 23 → 19 → 30)

Binary		7	6	5	4	3	2	1	0	Hexadecimal
OSRdyGrp		0	0	0	0	0	0	0	0	0x00
OSRdyTbl[8]	[0]	0	0	0	0	0	0	0	0	0x00
	[1]	0	0	0	0	0	0	0	0	0x00
	[2]	0	0	0	0	0	0	0	0	0x00
	[3]	0	0	0	0	0	0	0	0	0x00
	[4]	0	0	0	0	0	0	0	0	0x00
	[5]	0	0	0	0	0	0	0	0	0x00
	[6]	0	0	0	0	0	0	0	0	0x00
	[7]	0	0	0	0	0	0	0	0	0x00

Priority: 36

	7	6	5	4	3	2	1	0
B	0	0	1	0	0	1	0	0
O			y	=	4	x	=	4

OSMapTbl[8]

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

Create 4 tasks (36 → 23 → 19 → 30)

Binary									Hexadecimal
OSRdyGrp									
	7	6	5	4	3	2	1	0	
	0	0	0	1	0	0	0	0	0x10
OSRdyTbl[8]									
	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	0	0	0	0	0	0	0	0	0x00
[3]	0	0	0	0	0	0	0	0	0x00
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

Priority: 36

	7	6	5	4	3	2	1	0
B	0	0	1	0	0	1	0	0
O			y	=	4	x	=	4

OSMapTbl[8]

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

Create 4 tasks (36 → 23 → 19 → 30)

Binary	Hexadecimal
OSRdyGrp	

	7	6	5	4	3	2	1	0	
	0	0	0	1	0	1	0	0	0x14

OSRdyTbl[8]

	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	0	0	0	0	0x80
[3]	0	0	0	0	0	0	0	0	0x00
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

Priority: 23

	7	6	5	4	3	2	1	0
B	0	0	0	1	0	1	1	1
O			y	=	2	x	=	7

OSMapTbl[8]

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

Create 4 tasks (36 → 23 → 19 → 30)

Binary	Hexadecimal
OSRdyGrp	

	7	6	5	4	3	2	1	0	
	0	0	0	1	0	1	0	0	0x14

OSRdyTbl[8]

	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	1	0	0	0	0x88
[3]	0	0	0	0	0	0	0	0	0x00
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

Priority: 19

	7	6	5	4	3	2	1	0
B	0	0	0	1	0	0	1	1
O			y	=	2	x	=	3

OSMapTbl[8]

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

Create 4 tasks (36 → 23 → 19 → 30)

Binary	Hexadecimal

OSRdyGrp

7	6	5	4	3	2	1	0	
0	0	0	1	1	1	0	0	0x14

OSRdyTbl[8]

	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	1	0	0	0	0x88
[3]	0	1	0	0	0	0	0	0	0x40
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

Priority: 30

	7	6	5	4	3	2	1	0
B	0	0	0	1	1	1	1	0
O			y	=	3	x	=	6

OSMapTbl[8]

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

Find the minimum number (= The highest priority)

Binary									Hexadecimal
OSRdyGrp									
	7	6	5	4	3	2	1	0	
	0	0	0	1	1	1	0	0	0x14
OSRdyTbl[8]									
	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	1	0	0	0	0x88
[3]	0	1	0	0	0	0	0	0	0x40
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

OSRdyGrp = 0x14
OSUnMapTbl[0x14] = 2

```
INT8U const OSUnMapTbl[] = {
   0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
};
```

Find the minimum number (= The highest priority)

Binary									Hexadecimal
OSRdyGrp									
	7	6	5	4	3	2	1	0	
	0	0	0	1	1	1	0	0	0x14
OSRdyTbl[8]									
	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	1	0	0	0	0x88
[3]	0	1	0	0	0	0	0	0	0x40
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

OSRdyTbl[2] = 0x88
OSUnMapTbl[0x88] = 3

```
INT8U const OSUnMapTbl[] = {
   0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
   4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
};
```

Find the minimum number (= The highest priority)

Binary									Hexadecimal
OSRdyGrp									
	7	6	5	4	3	2	1	0	
	0	0	0	1	1	1	0	0	0x14
OSRdyTbl[8]									
	7	6	5	4	3	2	1	0	
[0]	0	0	0	0	0	0	0	0	0x00
[1]	0	0	0	0	0	0	0	0	0x00
[2]	1	0	0	0	1	0	0	0	0x88
[3]	0	1	0	0	0	0	0	0	0x40
[4]	0	0	0	1	0	0	0	0	0x10
[5]	0	0	0	0	0	0	0	0	0x00
[6]	0	0	0	0	0	0	0	0	0x00
[7]	0	0	0	0	0	0	0	0	0x00

	7	6	5	4	3	2	1	0
B	0	0	0	1	0	0	1	1
O			y	=	2	x	=	3

Priority: $23_{(8)} = 19_{(10)}$

Create 4 tasks (36 → 23 → 19 → 30)

<Hint>
To understand task scheduler,
It's good to see OS_CORE.C

Assignments

- I-Class에 다음 슬라이드의 과제 제출
- 제출 파일
  - 소스 코드(EX1.C) - Mandatory + Optional
  - 보고서(.pdf)
  - 실행 결과 화면 캡쳐
  - You have to capture and submit the screen on your report
- 파일명
  - 4주차_학번_이름(압축하여 하나의 파일로 제출)
- 제출 기한
  - 다음주 화요일 자정까지

What to do ? (Mandatory)

- 1개의 태스크 생성.
- 임의로 4개의 숫자 (0 ~63 중 1개의 숫자) 를 생성하고, 그 중 제일 작은 숫자를 반드시 스케줄러의 OSRdyGrp, OSRdyTbl[] OSMapTbl[] 과 OSUnMapTbl[] 네가지 자료구조를 사용해서 구함.
  - 단순히 4개의 숫자를 직접 비교하는 방법 사용시 과제 점수를 부여하지 않음
- final이라는 변수에 저장된 값보다 작은 숫자가 구해질 경우 색깔을 칠하고 final 값을 갱신.
- 위의 과정을 final에 0이 대입될때까지 반복하되 칠하는 숫자를 다음과 같이 바꿀 것.
  - Red-> Blue -> Green -> Brown -> Red …
- 색칠 과정을 확인 가능하도록 과정을 최소 2회 이상 반복. 단, 각 반복마다 칠해지는 색상 순서는 유지할 것.

What to do ? (Mandatory)

- Create a task.
- Create 4 numbers randomly(0 to 63), pick the smallest number using OSRdyGrp, OSRdyTbl[] OSMapTbl[] and OSUnMapTbl[] structure.
  - If you implement your assignment comparing 4 numbers directly, you wouldn't get any score in this week.
- When the smallest number is smaller than the value of "final", paint color and renew "final" value.
- Repeat the process above until final value becomes zero and colorizing cycle is below.
  - Red-> Blue -> Green -> Brown -> Red …
- Repeat the entire process for visualization. You must keep the color rotation sequence although each iteration is complete.

Task

LOOP

초기화

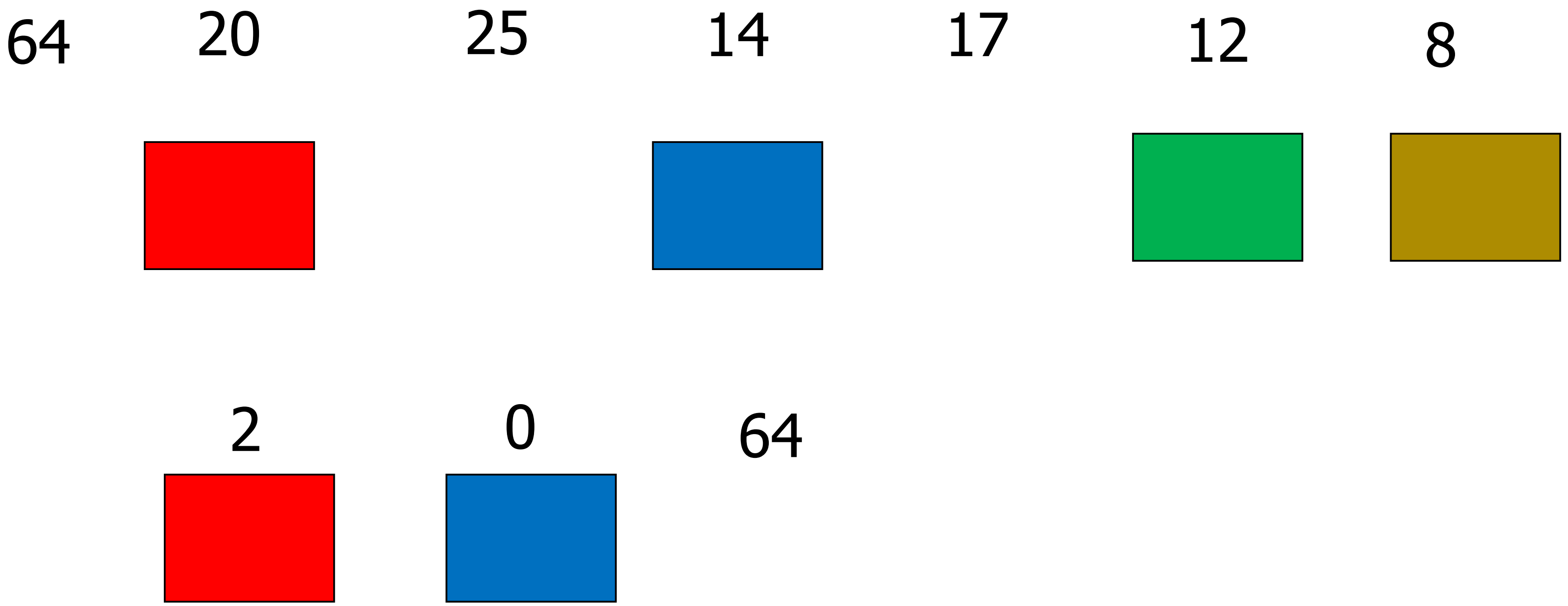
int final=64;

랜덤 생성 30,24,17,20

OSRdyGrp, OSRdyTbl[]
OSMapTbl[] OSUnMapTbl[]

temp=17

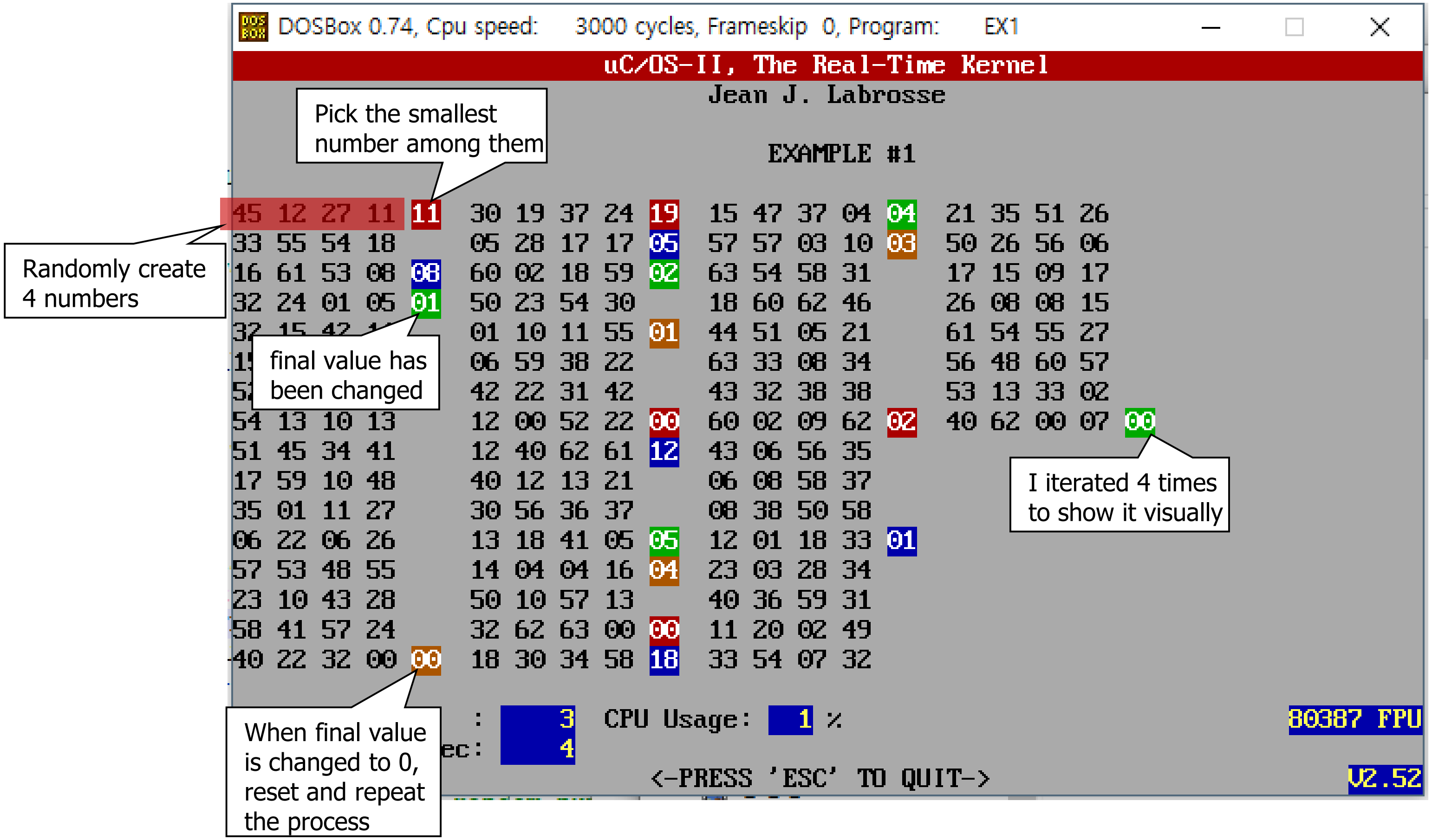
```
if(temp<final) {
    red->blue->green->brown->red …
      final 에 temp 값 대입
}
  if(final==0)
    final=64;


    OSTimeDly(200);
```



64 20 25 14 17 12 8

2 0 64

Assignment example



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: EX1

```
                    uC/OS-II, The Real-Time Kernel
                          Jean J. Labrosse

                             EXAMPLE #1

45 12 27 11 11   30 19 37 24 19   15 47 37 04 04   21 35 51 26
33 55 54 18      05 28 17 17 05   57 57 03 10 03   50 26 56 06
16 61 53 08 08   60 02 18 59 02   63 54 58 31      17 15 09 17
32 24 01 05 01   50 23 54 30      18 60 62 46      26 08 08 15
32 15 42 1       01 10 11 55 01   44 51 05 21      61 54 55 27
15               06 59 38 22      63 33 08 34      56 48 60 57
52               42 22 31 42      43 32 38 38      53 13 33 02
54 13 10 13      12 00 52 22 00   60 02 09 62 02   40 62 00 07 00
51 45 34 41      12 40 62 61 12   43 06 56 35
17 59 10 48      40 12 13 21      06 08 58 37
35 01 11 27      30 56 36 37      08 38 50 58
06 22 06 26      13 18 41 05 05   12 01 18 33 01
57 53 48 55      14 04 04 16 04   23 03 28 34
23 10 43 28      50 10 57 13      40 36 59 31
58 41 57 24      32 62 63 00 00   11 20 02 49
40 22 32 00 00   18 30 34 58 18   33 54 07 32

          :     3   CPU Usage:  1 %                           80387 FPU
        ec:     4
                       <-PRESS 'ESC' TO QUIT->                    V2.52
```

Pick the smallest number among them

Randomly create 4 numbers

final value has been changed

I iterated 4 times to show it visually

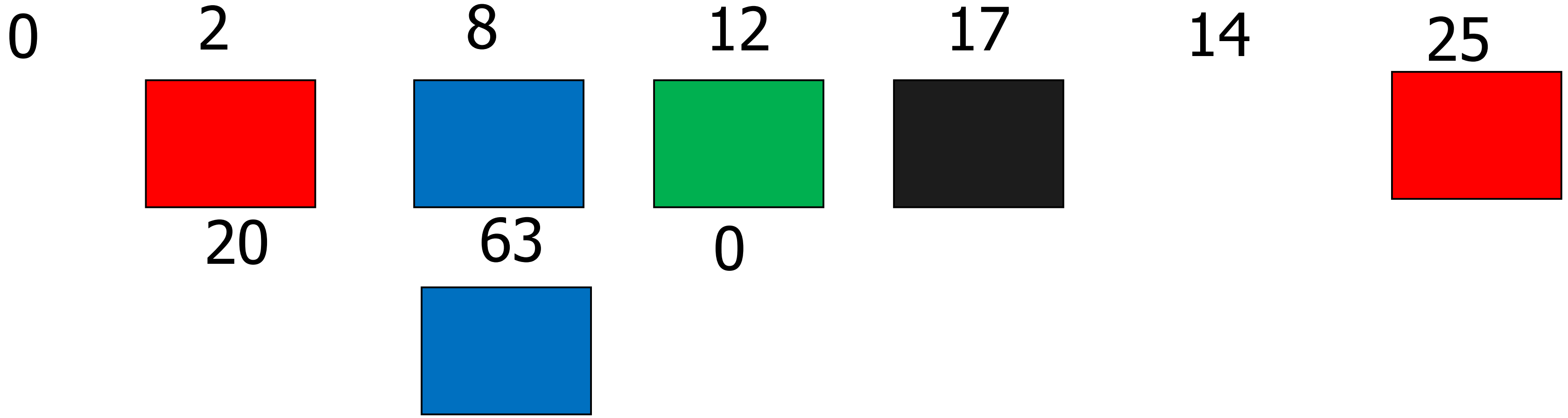
When final value is changed to 0, reset and repeat the process

DON'T COPY AND PASTE THIS IMAGE ON YOUR REPORT!

Optional assignment (for your practice)

Colorizing in increasing order

1. Randomly create 4 numbers
   30, 16, 8, 15
2. Find and remove the smallest number 3 times
   30
3. Compare the remaining number with final one
   1. Colorize in order
4. Remove the remaining number
5. Repeat the whole process until final becomes 63



Optional assignment (hint)

큰수를 찾을 때 다음을 활용하기

<Task deletion from ready list>

```
if ((OSRdyTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07]) == 0)
        OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

1. Randomly create 4 numbers 30, 16, 8, 15

   Final = 0

Ready queue

	8	**15**	**16**	**30**		

Display

2. Find and remove the smallest number 3 times

Final = 0

Smallest = 8

Ready queue

	8	**15**	**16**	**30**		

Display

2. Find and remove the smallest number 3 times

Final = 0

Smallest = 15

Ready queue

	15	**16**	**30**			

Display

2. Find and remove the smallest number 3 times

Final = 0

Smallest = 16

Ready queue

	16	30				

Display

3. Compare the remaining number with final one

Final = 0

Smallest = 30

Ready queue

	30					

Display

3-1. Colorize in order and renew final value

Final = 30

Smallest = 30

Ready queue

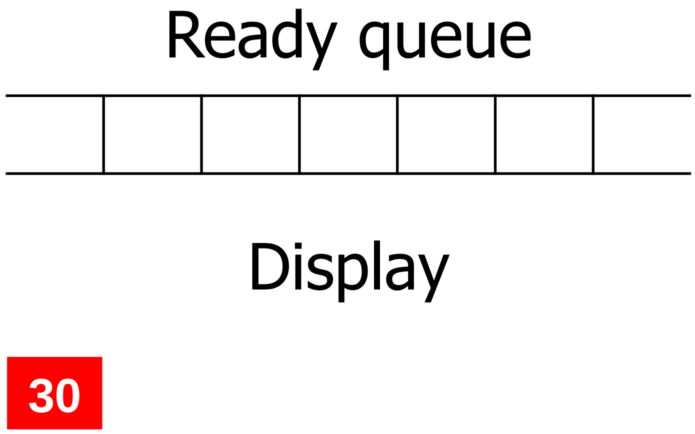
	30					

Display

30

4. Remove the remaining number

Final = 30



Ready queue

Display

30

5. Repeat the whole process until final becomes 63

Final = 30

Ready queue

	11	14	18	24		

Display

30

5. Repeat the whole process until final becomes 63

Final = 49

Ready queue

	0	5	32	49		

Display

30	49

5. Repeat the whole process until final becomes 63

   Final = 52

Ready queue

	2	3	15	52		

Display

30	49	52

5. Repeat the whole process until final becomes 63

Final = 52

Ready queue

	4	5	6	8		

Display

30	49	52

5. Repeat the whole process until final becomes 63

   Final = 59

Ready queue

	7	19	33	59		

Display

30	49	52	59

5. Repeat the whole process until final becomes 63

Final = 63

Ready queue

	3	10	52	63		

Display

30	49	52	59	63