

# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

강사 - Innova Lee(이상훈)

[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 - 은태영

[zero\\_bird@naver.com](mailto:zero_bird@naver.com)

# malloc 사용법

```
typedef struct
{
    int count;
    char name[20];
    int score[0];
} FLEX;

int main(void)
{
    int i;
    FLEX *p = (FLEX *)malloc(4096);
    printf("%d\n", sizeof(FLEX));

    for(i = 0; i < 10000; i++) {
        p->score[i] = i * 10;
        printf("%d : %d : \n", i, p->score[i]);
    }

    return 0;
}
```

- ❖ 배열 형식을 이용하여 동작한다.
- ❖ score[] : 이 구조체의 끝이 어디인지 나타낸다.
- ❖ malloc 으로 4096의 공간을 잡았지만, 사용은 그 이상으로 가능하다. 하지만, 프로세스들 사이에 동적 메모리 할당을 하는 것이 있다면, 충돌이 발생하게 된다.
- ❖ malloc : 할당 및 해제 시간이 길다.
- ❖ 많이 사용하면 속도가 느려 진다.
- ❖ 한번에 많이 잡아 놓고, 배열처럼 사용한다.
- ❖ 서버에서 많이 사용된다.

# semaphore

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include "sem.h"
2
3 int main(void)
4 {
5     int sid;
6     sid = CreateSEM(0x777);
7
8     printf("before\n");
9
10    p(sid);
11    printf("Enter Critical Section\n");
12
13    getchar();
14
15    v(sid);
16
17    printf("after\n");
18
19    return 0;
20 }
-- INSERT --
```

- ❖ createSEM 에 권한 값으로 0x777 을 주고, sid 에 semaphore ID 를 받아온다.
- ❖ before 출력 후, p 를 실행한다.
- ❖ 텍스트 출력 후 문자열을 받고, v 를 실행한다.
- ❖ 그 후 after 출력 후 종료한다.

# semaphore

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include "sem.h"
2
3 int CreateSEM(key_t semkey)
4 {
5     int status = 0, semid;
6     if(semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL) == -1) {
7         if(errno == EEXIST) {
8             semid = semget(semkey, 1, 0);
9         }
10    } else {
11        status = semctl(semid, 0, SETVAL, 2);
12    }
13    if(semid == -1 || status == -1) {
14        return -1;
15    }
16    return semid;
17 }
18
19
-- INSERT --
```

- ❖ key 값으로 0x777을 받는다.
- ❖ semaphore 얻어오는데 key 값 0x777, 권한을 1, 옵션으로 ipc 생성 과 해당 키 값의 ipc 가 존재 하는지 확인한 하도록 한다.
- ❖ 얻게 된 semaphore ID 를 semid 에 저장 후, if 문을 통하여 오류가 있는지 체크한다.
- ❖ 만약 조건이 만족할 경우, 존재하는 semaphore 를 사용한다.
- ❖ 아닐 경우, semctl 을 통해 semaphore 를 0으로 변경한다.

# semaphore

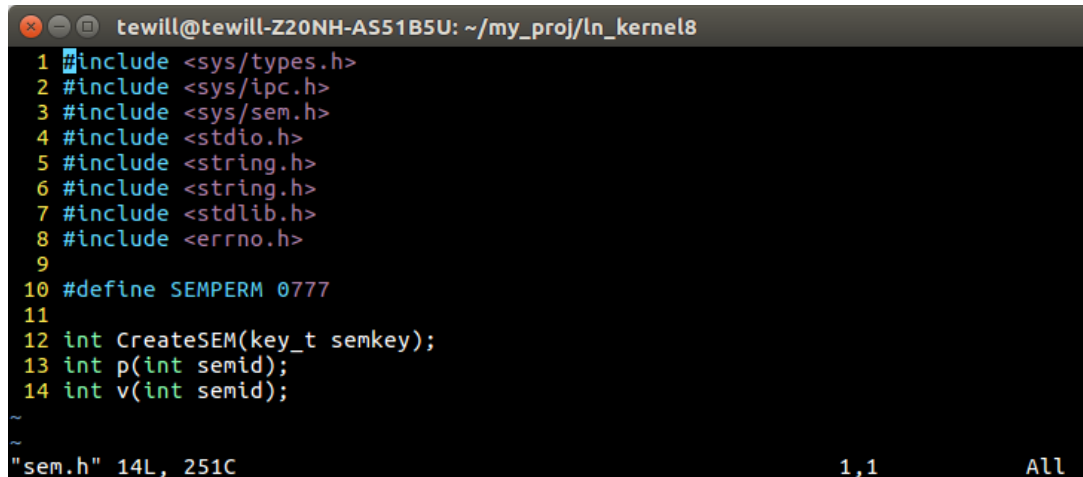
```
tewill@tewill-Z20NH-A551B5U: ~/my_proj/ln_kernel8
20 int p(int semid)
21 {
22     struct sembuf p_buf = {0, -1, SEM_UNDO};
23     if(semop(semid, &p_buf, 1) == -1)
24         return -1;
25     return 0;
26 }
27
28
29 int v(int semid)
30 {
31     struct sembuf p_buf = {0, 1, SEM_UNDO};
32     if(semop(semid, &p_buf, 1) == -1)
33         return -1;
34     return 0;
35 }
-- INSERT --
```

35,1

Bot

- ❖ 두가지 함수는 semaphore 값을 + 하거나, - 할 때 사용되는 일종의 패턴이다.
- ❖ SEM\_UNDO : 프로세스가 종료되면 semaphor를 원래 값으로 변경한다.
- ❖ p 가 증가 패턴이고, v 가 감소 패턴이다.

# semaphore



```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/sem.h>
4 #include <stdio.h>
5 #include <string.h>
6 #include <string.h>
7 #include <stdlib.h>
8 #include <errno.h>
9
10 #define SEMPERM 0777
11
12 int CreateSEM(key_t semkey);
13 int p(int semid);
14 int v(int semid);
~
"sem.h" 14L, 251C 1,1 All
```

- ❖ 위 프로세스를 구동하기 위한 헤더 파일이다.
- ❖ semaphore 는 대기열이 존재하며, 프로세스가 data 를 접근할 때, 다른 프로세스가 사용하고 있는 경우 lock 을 거는 것을 말한다.
- ❖ 대규모일 때는 semaphore 가 좋지만, 소규모일 경우 spinlock 이 좋다.
- ❖ spinlock 은 구분이 안되어 있어, 한곳에만 대기 하는 것을 말한다.

# Shared memory

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include "shm.h"
2
3 int CreateSHM(long key)
4 {
5     return shmget(key, sizeof(SHM_t), IPC_CREAT | 0777);
6 }
7
8 int OpenSHM(long key)
9 {
10    return shmget(key, sizeof(SHM_t), 0);
11 }
12
13 SHM_t *GetPtrSHM(int shmid)
14 {
15     return (SHM_t *)shmat(shmid, (char *)0, 0);
16 }
17
18 int FreePtrSHM(SHM_t *shmptr)
19 {
20     return shmdt((char *)shmptr);
21 }
-- INSERT --
```

- ❖ IPC\_CREAT 를 통하여 IPC 를 생성한다.
- ❖ key 값과 메모리 사이즈, 옵션과 권한이 들어간다.
- ❖ key 값을 통하여 Shared memory 의 ID 를 얻어온다.
- ❖ 물리 메모리를 찾는다. 인자로 ID 값, 찾는 위치, 끝 을 나타낸다.
- ❖ 리턴 값으로 물리 메모리의 주소를 준다.
- ❖ Shared memory 를 해제한다.

# Shared memory

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include "shm.h"
2
3 int main(void)
4 {
5     int mid;
6     SHM_t *p;
7
8     mid = OpenSHM(0x888);
9
10    p = GetPtrSHM(mid);
11
12    getchar();
13    strcpy(p->name, "아무개");
14    p->score = 93;
15
16    FreePtrSHM(p);
17
18    return 0;
19 }
-- INSERT --
```

- ❖ 해당 키 값으로 Shared memory ID 를 받는다.
- ❖ ID를 통하여 물리 메모리의 주소를 찾는다.
- ❖ char 를 입력 받는다.
- ❖ 물리 메모리의 주소에 데이터를 입력한다.
- ❖ 더이상 공유할 게 없을 때 공유를 해제한다.



# Shared memory

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include "shm.h"
2
3 int main(void)
4 {
5     int mid;
6     SHM_t *p;
7
8     mid = CreateSHM(0x888);
9
10    p = GetPtrSHM(mid);
11
12    getchar();
13    printf("이름 : [%s], 점수 : [%d]\n", p->name, p->score);
14
15    FreePtrSHM(p);
16    return 0;
17 }
```

1,1 All

- ❖ 같은 방식으로 shared memory 를 만든다.
- ❖ 물리 메모리의 주소를 받아온다.
- ❖ 물리 메모리에 저장된 것을 출력한다.
- ❖ 그 후, 메모리 공유를 종료한다.

# Shared memory

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
1 #include <sys/ipc.h>
2 #include <sys/shm.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <errno.h>
7
8 typedef struct
9 {
10     char name[20];
11     int score;
12 } SHM_t;
13
14 int CreateSHM(long key);
15 int OpenSHM(long key);
16 SHM_t *GetPtrSHM(int shmid);
17 int FreePtrSHM(SHM_t *shmptr);
~
1,1 All
```

❖ 위 프로세스를 구동 시키는 헤더 파일이다.

```
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/ln_kernel8
tewill@tewill-Z20NH-AS51B5U:~/my_proj/ln_kernel8$ gcc -o send 3_shm1ib.c 3_send.c
tewill@tewill-Z20NH-AS51B5U:~/my_proj/ln_kernel8$ gcc -o recv 3_shm1ib.c 3_recv.c
tewill@tewill-Z20NH-AS51B5U:~/my_proj/ln_kernel8$
```

- ❖ 다음과 같은 방법으로 실행 파일을 만든다.
- ❖ send 를 먼저 실행한다.
- ❖ 다른 터미널을 통해 recv 를 실행한다.
- ❖ 데이터가 공유되는 것을 확인할 수 있다.

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
1 #include <time.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdbool.h>
5
6 typedef struct
7 {
8     int data;
9     int idx;
10 }
11 queue;
12
13 typedef struct
14 {
15     int full_num;
16     int free_num;
17     int total;
18     int cur_idx;
19
20     int free[1024];
21     int total_free;
22     queue head[0];
23 }
24 manager;
25
-- INSERT --
```

1,1 Top

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
162
163 int main(void)
164 {
165     int i;
166     bool is_full;
167     int alloc_size = 1 << 12;
168     int data[10] = {0};
169     int size = sizeof(data) / sizeof(int);
170
171     srand(time(NULL));
172     init_data(data, size);
173     print_arr(data, size);
174
175     manager *m = (manager *)malloc(alloc_size * 4);
176     init_manager(m, alloc_size);
177     printf("Before Enqueue\n");
178     print_manager_info(m);
179
-- INSERT --                                162,1      81%
```

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
179
180     for(i = 0; i < size; i++)
181         enqueue(m, data[i]);
182
183     printf("After Enqueue\n");
184     print_queue(m);
185
186     dequeue(m, data[1]);
187
188     printf("After Dequeue\n");
189     print_queue(m);
190
191     enqueue(m, 777);
192     print_manager_info(m);
193     print_queue(m);
194
195     dequeue(m, data[4]);
196     dequeue(m, data[5]);
197     dequeue(m, data[6]);
198     enqueue(m, 333);
199     print_manager_info(m);
200     print_queue(m);
201
-- INSERT --
```

179,1 92%

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
201 █
202 #if 1
203     // 강제로 꽉찼다 가정하고 free 공간을 활용 해보자!
204     is_full = true;
205 #endif
206
207     //if(is_it_full(m))
208     if(is_full)
209         enqueue_with_free(m, 3333);
210
211     print_manager_info(m);
212     print_queue(m);
213
214     return 0;
215 }
-- INSERT --
```

201,1

Bot

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
25
26 bool is_dup(int *arr, int cur_idx)
27 {
28     int i, tmp = arr[cur_idx];
29
30     for(i = 0; i < cur_idx; i++)
31         if(tmp == arr[i])
32             return true;
33
34     return false;
35 }
36
37 void init_data(int *data, int size)
38 {
39     int i;
40
41     for(i = 0; i < size; i++)
42     {
43 redo:
44         data[i] = rand() % 100 + 1;
45
46         if(is_dup(data, i))
47         {
48             printf("%d dup! redo rand()\n", data[i]);
49             goto redo;
50         }
51     }
52 }
53
-- INSERT --
```

25,1 12%



```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
53
54 void print_arr(int *arr, int size)
55 {
56     int i;
57
58     for(i = 0; i < size; i++)
59         printf("arr[%d] = %d\n", i, arr[i]);
60 }
61
62 void init_manager(manager *m, int alloc_size)
63 {
64     m->full_num = 0;
65     m->free_num = ((alloc_size / sizeof(int)) - 1029) / 2;
66     4096 + 20 (매니저 자료형 크기) / 4
67     m->total = ((alloc_size / sizeof(int)) - 1029) / 2;
68     m->cur_idx = 0;
69 }
70
71 void print_manager_info(manager *m)
72 {
73     int i;
74
75     printf("m->full_num = %d\n", m->full_num);
76     printf("m->free_num = %d\n", m->free_num);
77     printf("m->total = %d\n", m->total);
78     printf("m->cur_idx = %d\n", m->cur_idx);
79     printf("m->total_free = %d\n", m->total_free);
80
81     for(i = 0; i < m->total_free; i++)
82         printf("m->free = %d\t", m->free[i]);
83
84     printf("\n");
85 }
```

53,0-1 26%

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
86
87 void enqueue(manager *m, int data)
88 {
89     m->head[m->cur_idx].data = data;
90     m->head[m->cur_idx++].idx = m->cur_idx;
91     m->free_num--;
92     m->full_num++;
93 }
94
95 void dequeue(manager *m, int data)
96 {
97     int i;
98
99     for(i = 0; i < m->full_num; i++)
100     {
101         if(m->head[i].data == data)
102         {
103             m->head[i].data = 0;
104             m->head[i - 1].idx = m->head[i].idx;
105             m->head[i].idx = -1;
106             m->free_num++;
107             m->full_num--;
108             m->free[m->total_free++] = i;
109         }
110     }
111 }
112
-- INSERT --
```

86,1 42%

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
112
113 void print_queue(manager *m)
114 {
115     int i = 0;
116     int flag = 0;
117     int tmp = i;
118
119     printf("print_queue\n");
120
121     while(m->head[tmp].data)
122     {
123         printf("data = %d, cur_idx = %d\n", m->head[tmp].data, tmp);
124         printf("idx = %d\n", m->head[tmp].idx);
125
126         for(; !(m->head[tmp].data);)
127         {
128             tmp = m->head[tmp].idx;
129             flag = 1;
130         }
131
132         if(!flag)
133             tmp = m->head[tmp].idx;
134
135         flag = 0;
136     }
137 }
```

112,0-1 56%

```
tewill@tewill-Z20NH-AS51B5U: ~/Desktop
138
139 bool is_it_full(manager *m)
140 {
141     if(m->full_num < m->cur_idx)
142         return true;
143
144     return false;
145 }
146
147 void enqueue_with_free(manager *m, int data)
148 {
149     m->head[m->cur_idx - 1].idx = m->free[m->total_free - 1];
150     m->total_free--;
151     m->head[m->free[m->total_free]].data = data;
152     m->head[m->free[m->total_free]].idx = m->free[m->total_free - 1];
153
154     if(!(m->total_free - 1 < 0))
155         m->head[m->free[m->total_free]].idx = m->free[m->total_free
- 1];
156     else
157         printf("Need more memory\n");
158
159     m->free_num--;
160     m->full_num++;
161 }
162
```

162,0-1 72%

