2018. 3. 28 수 – 25 회차

과정 : TI, DSP, Xilinx Znq FPGA, MCU 기반의 프로그래밍 전문가 과정

Prof. 이상훈
gcccompil3r@gmail.com
Stu. 정상용
fstopdg@gmail.com

Linux 운영체제

1. 인덱스가 없는 배열(arr[0])

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

→ 위의 경우를 예로 들어보자. 우리의 상식대로라면, int score[0]은 배열이 될 수가 없다. 그러나 score[0], score[1]을 compile 하면 error 가 나오지 않는다. 즉 int score[0]도 배열처럼 사용가능하다는 이야기다.

→ int score[0]은 구조체의 끝이자 새로운 시작

→ malloc 을 자주 사용하면 좋지 않다.(할당받는 시간 & 해제하는 시간이 오래 걸림.) 그러므로 malloc 을 한 번만 사용후 배열로 계속 집어넣으면 된다.

```
(FLEX *)malloc(4096)
```

→ 위의 경우에 4096byte 가 넘어도 상관없다. 다른 malloc 을 침범하기 전까지는 문제가 없다. 그러나 침범할 경우 심각한 문제를 야기하므로 조심해야 한다.

Ex.

```
#include <stdio.h>
#include <stdlib.h>

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

int main(void)
{
   int i;
   FLEX *p = (FLEX *)malloc(4096);
```

```
    p -> score[0];
    p -> score[1];
    printf("%d\n", sizeof(FLEX));
    return 0;
}
```

2. arr[0]을 이용한 queue 구현

→ enqueue, dequeue

→ loop(while, for...)는 비효율적임

Ex. What I made

```
#include <stdio.h>
#include <malloc.h>

#define EMPTY 0

typedef struct __queue
{
    int mas[0];
} queue;

queue *get_node()
{
    queue *tmp;
    tmp = (queue *)malloc(10000);
    return tmp;
}

void enqueue(queue **head, int data, int idx)
{
        (*head) -> mas[idx] = data;
        return ;
}

void print_queue(queue *head)
{
    int i = 0;
    while(1)
    {
      if(head -> mas[i] != EMPTY)
          printf("%d\n", head -> mas[i]);
      else if(head -> mas[i] == EMPTY)
          break;
      i++;
    }
    return ;
}

queue *dequeue(queue *head, int idx)
{
    int i = 0;
```

```
    int *adr = 0;
    while(1)
    {
        if(idx != i)
            printf("Searching.....\n");
        else if(idx = i)
        {
            &(head -> mas[i]) = adr;
            *adr = 0;
            break;
        }
        i++;
    }


    return head;
}
int main(void)
{
    queue *head = get_node();
    enqueue(&head, 10, 0);
    enqueue(&head, 20, 1);
    enqueue(&head, 30, 2);
    enqueue(&head, 40, 3);
    enqueue(&head, 50, 4);
    print_queue(head);

    head = dequeue(head, 2);

    print_queue(head);


}
```

3. Lock

→ OS 내에서 Lock 메커니즘은 2 개 존재(상황에 따라 선택적 사용)

1. semaphore : 대기열이 존재

2. spinlock : CPU 를 지속적으로 잡고있음(polling, 내가 화장실에 있는데 누군가 3sec 마다 문을 두들긴다고 상상)

4. IPC(Inter – Process Communication)

→ 반드시 숙지

→ 사용하는 이유 : 한 Process 에서 가지고 있는 정보를 다른 Process 로 넘겨주기 위하여(Ex. 자율주행자동차의 경우, 영상처리 process 에서 얻은 정보를 모터제어 process 에 넘겨주어 차량이 멈추거나 속도를 바꾸는 동작을 할수있음)

→실행방법 : send.c & recv.c 를 동시에 실행

1. gcc -o send shmlib.c send.c
2. gcc -o shmlib.c recv.c
3. Terminal 2 개를 띄우고 각각의 창에서 ./send & ./recv 실행