TI DSP,Xilinx zynq FPGA,MCU 및 Xilinx zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)
Gccompil3r@gmail.com
학생-윤지완
Yoonjw7894@naver.com

```
#include <time.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct
{
int score;//주소값,이 주소를 어디를 가르키고 있나? 그걸 알라면
char name[20];
}st;
typedef struct
{
int count;
char name[20];
int score[0];//배열처럼 사용할 수 있다. 구조체의 끝은 어딘가 알려주고,새로운 시작점을 알려준다.
}FLEX;
int main(void)
{
FLEX *p= (FLEX*)malloc(4096);//메모리를 4096 을 사용하겠다,계속해서 할당을 맏는게 아니라 미리 크게 할당해놓고 쓰니까 속도가 빠르다.
int i;
for(i=0;i<100;i++)
{
p->score[i]=i+1;
printf("score= %d\n",p->score[i]);//FLEX 의 사이즈를 알려준다= 24byte
}
printf("%d",sizeof(FLEX));
return 0;
}
void enqueue(FLEX *p,int score)
{
int a;
p->score[score]=score+1;
a=p->score[score];
printf("p->score[%d]\n=%d\n",score,p->score[score]);
}



int main(void)
{
FLEX *p= (FLEX*)malloc(4096);//메모리를 4096 을 사용하겠다
int i;
for(i=0;i<100;i++)
enqueue(p,i);
printf("%d",sizeof(FLEX));
return 0;
```

```
}


#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct
{
	int data;
	int idx;
}
queue;

typedef struct
{
	int full_num;
	int free_num;
	int total;//최고값
	int cur_idx;//현제 무슨 인덱스를 가르키고 있나 보는것
	// free idx
	int free[1024];//번호 관리
	int total_free;//별도로 관리하는개 몇개가 있나
	queue head[0];//배열에 들어가는 값들을 넣는것
}

manager;

bool is_dup(int *arr, int cur_idx)
{
	int i, tmp = arr[cur_idx];

	for(i = 0; i < cur_idx; i++)
		if(tmp == arr[i])
			return true;

	return false;
}

void init_data(int *data, int size)
{
	int i;

	for(i = 0; i < size; i++)
	{
redo:
		data[i] = rand() % 100 + 1;

		if(is_dup(data, i))
```

```
   printf("%d dup! redo rand()\n", data[i]);
                    goto redo;
               }
          }
}

void print_arr(int *arr, int size)
{
          int i;

          for(i = 0; i < size; i++)
               printf("arr[%d] = %d\n", i, arr[i]);
}

void init_manager(manager *m, int alloc_size)
{
          m->full_num = 0;//할당한개 아무것도 없다
          // 12: full_num, free_num, cur_idx
          // 8: data, idx
          m->free_num = (alloc_size / sizeof(int) -1029 ) / 2;//
          //-12 가 아니라 -1029 를 빼야 한다.
// 이부분에선 배열인덱스가 얼마나 비었나 확인하는 코드인데,
//여기서 나누기 8 을 나누는게 아니라 나누기 2 를 해야한다. 1533
   m->total = (alloc_size / sizeof(int) - 1029) / 2;
          m->cur_idx = 0;
}

void print_manager_info(manager *m)
{
          int i;

          printf("m->full_num = %d\n", m->full_num);
          printf("m->free_num = %d\n", m->free_num);
          printf("m->total = %d\n", m->total);
          printf("m->cur_idx = %d\n", m->cur_idx);
          printf("m->total_free = %d\n", m->total_free);

          for(i = 0; i < m->total_free; i++)
               printf("m->free = %d\t", m->free[i]);

          printf("\n");
}

void enqueue(manager *m, int data)
{
  m->head[m->cur_idx].data = data;
          m->head[m->cur_idx++].idx = m->cur_idx;
          m->free_num--;
          m->full_num++;
}
```

```
void dequeue(manager *m, int data)
{
	int i;

	for(i = 0; i < m->full_num; i++)
	{
		if(m->head[i].data == data)
		{
			m->head[i].data = 0;
			m->head[i - 1].idx = m->head[i].idx;
			m->free_num++;
			m->full_num--;//1533-1=1532
			m->free[m->total_free++] = i;
		}
	}
}

void print_queue(manager *m)
{
	int i = 0;
	int flag = 0;
	int tmp = i; // m->head[i].idx;

	printf("print_queue\n");

#if 0
	for(; !(m->head[tmp].data);)
		tmp = m->head[tmp].idx;
#endif

	while(m->head[tmp].data)
	{
		printf("data = %d, cur_idx = %d\n", m->head[tmp].data, tmp);
		printf("idx = %d\n", m->head[tmp].idx);

		for(; !(m->head[tmp].data);)
		{
			tmp = m->head[tmp].idx;//중간에 지운값이 있을경우 그 다>음인덱스를 찾아가라는 코
드.

   flag = 1;
		}

		if(!flag)
			tmp = m->head[tmp].idx;

		flag = 0;
	}
}

bool is_it_full(manager *m)
{
```

```
	if(m->total_num < m->cur_idx)
		return true;

	return false;
}

void enqueue_with_free(manager *m, int data)
{
	/*
			m->head[i].data = 0;
			m->head[i - 1].idx = m→head[i].idx;
		  m->free_num++;
			m->full_num--;
			m->free[m->total_free++] = i;
	*/

	m->head[m->cur_idx - 1].idx = m->free[m->total_free - 1];
	m->total_free--;
	m->head[m->free[m->total_free]].data = data;//비어진공간에 값을 넣겠다
	m->head[m->free[m->total_free]].idx = m->free[m->total_free - 1];

	if(!(m->total_free - 1 < 0))
		m->head[m->free[m->total_free]].idx = m->free[m->total_free - 1];
	else
		printf("Need more memory\n");

	m->free_num--;
	m->full_num++;
}

int main(void)
{
	int i;
 bool is_full;
	int alloc_size = 1 << 12;
	int data[10] = {0};
	int size = sizeof(data) / sizeof(int);

	srand(time(NULL));
	init_data(data, size);
	print_arr(data, size);

	manager *m = (manager *)malloc(alloc_size*4);//4096 메모리 할당
	init_manager(m, alloc_size);
	printf("Before Enqueue\n");
	print_manager_info(m);

	for(i = 0; i < size; i++)
		enqueue(m, data[i]);

	printf("After Enqueue\n");
	print_queue(m);
```

```
        dequeue(m, data[1]);
 printf("After Dequeue\n");
        print_queue(m);

        enqueue(m, 777);
        print_manager_info(m);
        print_queue(m);

        dequeue(m, data[4]);
        dequeue(m, data[5]);
        dequeue(m, data[6]);
        enqueue(m, 333);
        print_manager_info(m);
        print_queue(m);

#if 1
        // 강제로 꽉찼다 가정하고 free 공간을 활용 해보자!
        is_full = true;
#endif

        //if(is_it_full(m))
        if(is_full)
                enqueue_with_free(m, 3333);

       print_manager_info(m);
        print_queue(m);

        return 0;
}


           #include "sem.h"
int CreateSEM(key_t semkey)
{
int status = 0, semid;
if((semid = semget(semkey , 1 ,SEMPERM|PC_CREAT|PC_EXCL))==-1)
if(errno ==EEXIST)
semid= semget(semkey, 1, 0);
else
status =semctl(semid, 0, SETVAL, 2);
if(semid ==-1||status == -1)
return -1;
return semid;
}

int p(int semid)
{
struct sembuf p_buf = }{,-1,SEM_UNDO};
if(semop(semid , &p_buf,1)==-1)
return -1;
return 0;
```

```
}
int v(int semid)
{
struct sembuf p_buf= {0,1,SEM_UNDO};
if(semop(semid,&p_buf,1) ==-1)
return  -1;
return 0;
}
```

sem_undo:원래값으로 돌려놓는다.

```
int v(int semid){
struct sembuf p_buf = {0, 1, SEM_UNDO};
if(semop(semid, &p_buf, 1) == -1)
return -1;
return 0;
}
```

<더하기 기능>

```
int p(int semid){
struct sembuf p_buf = {0, -1, SEM_UNDO};
if(semop(semid, &p_buf, 1) == -1)
return -1;
return 0;
}
```

<빼기 기능>

spinlock:cpu 를 지속적으로 잡고있는다. 오직 한가지 작업만 수행한다는 것이다.

Semaphose:대기열이 존재하기에 여러가지 일을 할 수 가 있다. 기다리면서 wate q 로 간다

이말은 contex switching 을 한다는 것이다.저장하고 관히라는데서 clock 손실이 있다.

대규모-Semaphose

단순 간다-spinlock

Critical section

여러 테스트가 동시에 접근해서 데이터 결과값이 달라지는 구간.코드 구현을 어셈블리어로 되어있다.
만약 A 프로세스와 B 프로세스가 동작을 하는데 A 의 데이터를 공유를 한다.그럼 A 의 동작이 끝나고 제어권이 B 프로세스로 넘어가서 B 프로세스가 동작을 해야되는 그 때 B 프로세스의 동작씹히고 다시 제어권이 A 프로세스로 오게되고 결국 최종값이 변하게 되는 상황이다.
그래서 이 상황을 방지하기 위해서 A 프로세스의 동작을 할 때 LOCK 을 걸어주는 것이다. 그럼 A 프로세스가 끝나기 전까지는 아무도 건들일수 없기때문에 정상적인 값이 나온다.

Send 는 물리메모리에서 write 하고 recv 는 읽기만 한다.
ipc 를 쓰는 이유는 프로세스들간에 메모리 공유를 하기 위해서이다.

Send.c

```
#include "shm.h"
int main(void)
{
int mid;
SHM_t *p;
mid =openSHM(0X888);//물리 메모리의 페이지 프레임을 얻은 메모리값을 받는다.
//페이지 프레임을 아무나 막쓰지않기 위해 아이디를 준것이다.
p= GetptrSHM(mid);//쉐어드 메모를 받아와라.공유 메모리에 물리주소를
getchar();
strcpy(p->name, "아무개");//p→name 배열에 아무개라는 이름을 저장하고 복사한다.
p->score = 93;//
freeptrSHM(p);//해제해라. 하지만 바로 해제가 되지는 않는다.
return 0;
}
```

<shmlib.c>

```
#include "shm.h"

int CreateSHM(long key)
{
return shmget(key, sizeof(SHM_t) IPC_CREAT|0777);//0777 의 ipc 권한을 준다.
}
int openSHM(long key)
{
return shmget(key, sizeof(SHM_t),0);
}
SHM_t *GetptrSHM(int shmid)
{
return (SHM_t*)shmat(shmid,(char *)0,0);//맨 첫번째 부터 찾겠다는 소리 0 번째부터
 공유 메모리에 물리주소를 얻어서 SHMH_t 에 공간을 할당한다.
}
int FreeptrSHM(SHM_t *shmptr)
{
return shmdt((char *)shmptr);
}
```

위에 세 가지 코드를 가지고 물리 메모리를 공유하여 send 에서 보낸 아무개 성적 93 점을 recv 에서 읽는 방식이다.그 물리 메모리 공간을 할당하고 권한을 부여하는 것은 shmlib.c 코드에서 주는 것이다.헤어 파일에는 구조체를 만들어 이름고 점수를 만들었다.

<헤어 파일>

```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<errno.h>

typedef struct
{
char name[20];
int score;
}SHM_t;
int CreatSHM(long key);
int openSHM(long kry);
SHM_t*getptrSHM(int shmid);
int FreeptrSHM(SHM_t *shmptr);
```

gcc -o recv shmlib.c recv.c
gcc -o send shmlib.c send.c