TI DSP, MCU, Xilinx Zynq FPGA Based Programming Expert Program

Instructor – Innova Lee(Sanghoon Lee)
gcccompil3r@gmail.com
Student – Hyungju Kim
mihaelkel@naver.com

Advanced memory allocation.

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

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

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

int main(void){
    FLEX* p = (FLEX*)malloc(4096);
    int i;
    //34807 -> segmentation fault.
    for(i=0;i<34806;i++)
        p->score[i] = i;
    for(i=0;i<34806;i++)
        printf("data : %d\n",p->score[i]);

    printf("size : %d\n",sizeof(FLEX));
    return 0;

}

```


When implementing a data structure, we use dynamic allocation.
it can save memory, but, the dynamic allocation has a demerit for speed.
So, there are 1 solution, which use dynamic allocation only once.

Semaphore

- OS LOCK mechanism

A global variable in a process can be used at the child process. in that case, the global variable, which is in a “Data area“ , be ”Critical section“. the variable in the Critical section might be misused by ”context switching“. So, there is a solution, OS Lock. there are 2 methods to lock. Semaphore, Spinlock. Spinlock is used when the process is relatively simple. otherwise, Semaphore is suitable for heavy program, especially never allow any data loss.

System call : semget();
Prototype : int semget(key_t key, int nsems, int semflg)
returns : semaphore id when it successes.
-1 on error.
System call : semop();
Prototype : int semop(int semid, struct sembuf *sops, unsigned nsops)
returns : 0 on success(all operation performed)
-1 on error.
struct sembuf{
ushort sem_num;
short sem_op;
short sem_flg;
}

```c
#include "sem.h"

int CreateSEM(key_t semkey){
    int status = 0,semid;
    //IPC : inter process communication.
    if((semid = semget(semkey, 1, SEMPERM|IPC_CREAT|IPC_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 = {0, -1, SEM_UNDO};// SEM_UNDO initialyze sem, when process exits.
    if(semop(semid, &p_buf, 1) == -1) // semaphore value +1
        return -1;
    return 0;
}

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

```

```c
#include "sem.h"

int main(void){
	int sid;

	//semaphore id
	sid = CreateSEM(0x777);

	printf("before\n");

	p(sid);

	printf("Enter Critical Section\n");

	getchar();

	v(sid);

	printf("after\n");

	return 0;

}

```



```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

#define SEMPERM 0777

int CreateSEM(key_t semkey);
int p(int semid);
int v(int semid);

```

Shared Memory

think about make an autonomous car. There are a lot of processes running at the same time, and some data should be shared. for example, Spacing with a front car shoul be sended to the process that controls a motor. but, We can't share data to other process because virtual memory which in "task_struct" are exclusive. in that case, We can share data with accessing the physical memory directly.

```
#include "shm.h"

int CreateSHM(long key){
    return shmget(key,sizeof(SHM_t), IPC_CREAT | 0777);
}

int OpenSHM(long key){
    return shmget(key,sizeof(SHM_t), 0);
}

SHM_t* GetPtrSHM(int shmid){
    return (SHM_t*)shmat(shmid,(char*)0,0);
}

int FreePtrSHM(SHM_t* shmptr){
    return shmdt((char*)shmptr);
}
```



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

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

int CreateSHM(long key);
int OpenSHM(long ket);
SHM_t* GetPtrSHM(int shmid);
int FreePtrSHM(SHM_t* shmptr);
```

```
#include "shm.h"

int main(void){
    int mid;
    SHM_t* p;
    mid = OpenSHM(0x888);

    p = GetPtrSHM(mid);

    getchar();
    strcpy(p->name,"Who.");
    p->score = 93;

    FreePtrSHM(p);
```

```
16      return 0;
17  }
18
```

```
1   #include "shm.h"
2
3   int main(void){
4       int mid;
5       SHM_t* p;
6       mid = CreateSHM(0x888);
7
8       p = GetPtrSHM(mid);
9
10      getchar();
11
12      printf("name : [%s], score : [%d]\n",p->name,p->score);
13
14      FreePtrSHM(p);
15
16      return 0;
17  }
18
```