TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018.03.21
20 일차
강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

프로세스는 cpu 의 추상화다.

cpu 는 오로지 한 순간에 한 가지의 연산만 수행한다.

cpu 주파수:2GHz = f(주파수)
주기 1/f = T
주파수가 올라가면 주기가 내려가고 주파수가 내려가면 주기가 올라간다.

1clock = G = 10e9

T = 1/2x10e9 = 5/10e10 = 5x10e-10 = 5/10000000000

1 초: T --> 20 억 분의 1 초 ;;

아주빠른속도로 여러 프로세스들이 제어권을 넘겨주면서
cpu 를 사용한다면 우리가 느끼지 못하는 순간에 모든 작업이 완료된다.

Run Queue	Wait Queue
A B C	A E G F

CPU
-PIC(Programable Interrupt Clock)
-장치
-키보드
-RTC(Real Time Clock)
DRAM(물리)
-Process

Run Queue	Wait Queue
A B C	D E G F

프로세스가 여러개 있을 때 프로그램이 동시에 돌아가게 하려면

빠른속도로 컨텍스트 스위칭을 하면서 동시에 가깝게 프로그램들을
실행해준다. 컨텍스트 스위칭이란 CPU 가 오로지 한순간에 한가지의
연산만을 실행 할 수 있는데 이것을 Run Queue 와 Wait Queue 를 나눠서
우선순위가 높은 프로세스를 Run 하고 다른 프로세스들을 Wait 시키고
우선순위가 높은 프로세스가 연산이 끝나면 Wait 하고있던 프로세스를 실행시킨다.
근데 이 실행속도가 인간이 감지하기 어려운 속도로 돌아가고 있기 때문에
동시에 돌아가는 것 처럼 느껴지는 것이다.

메모리라는 물리공간에 Process 를 갖다놓고, CPU 는 Fetch(가져온다)하고
Decode(해석)하고,Excution(실행,연산)한다.

예를들면 A , B 프로세스가 있다.
A 프로세스를 실행하고 있는데 ax 레지스터에 7 이 들어가 있는 상태였고
주도권이 B 프로세스로 넘어갔다 B 프로세스는 ax 레지스터에 3 을 넣어야 하는 상황이어서
ax 레지스터에 3 을 넣었다. 그리고나서 bx 에 10 을 넣고 다시 주도권이 a 로 넘어갔다
그러면 프로세스 A 는 3 이 들어가있어야 하는데 7 이 들어가있어버려서 결과가 4 가 더 나온
예상보다 다른 값이 나온거다. 그래서 이럴 때 task_struct 에다가 레지스터 값을
주도권이 넘어갈 때 마다 저장해 놓고 다시 주도권이 돌아오면 레지스터 값을 다시
불러서 사용하는것이다.

pcb == Process Controll Block

여기서의 c b 는 콘솔창에 ls 를 눌렀을 때 앞에 나오는 글자를 뜻함.

메모리의최소단위 4k 근데 B 는 블록단위(dRAM)인데 단위별로
메모리의주소단위 페이지 4k
RAM == Random Access Memory
말그대로 램은 랜덤단위다. 실제로 함수 호출시에 스택에 저장되는 주소는 랜덤으로 할당된다.
c 는 캐릭터인데 순서단위인데(캐릭터디바이스라고 하고 키보드,모니터 등이 이곳에 속함)

fifo.c

```
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>

int main(void){
	int fd, ret;
	char buf[1024];
	mkfifo("myfifo");
	fd = open("myfifo", O_RDWR);
	for(;;)
	{
		ret = read(0, buf, sizeof(buf));
		buf[ret - 1] = 0;
		printf("Keyboard Input : [%s]\n",buf);
		read(fd, buf, sizeof(buf));
		buf[ret -1] = 0;
		printf("Pipe Input : [%s]\n",buf);
	}
	return 0;
```

```
}
```

dup.c

```
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>

int main(void)
{
	int fd;
	fd	=	open("a.txt",O_WRONLY|O_CREAT|O_TRUNC,
0644);
	close(1);
	dup(fd);
	printf("출력될까 ? \n");
	return 0;
	/*
	fd
	*/
}
```

dup2.c

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>

int main(void)
{
	int fd;
```

```
	char buff[1024];
	fd = open("a.txt",O_RDONLY);
	close(0);
	dup(fd);
	gets(buff);
	printf("출력될까? \n");
	printf("%s",buff);
	return 0;
	/*
	open 함수를 사용하고 리턴으로 0 을 fd 에다가 줬다. 그런데
	close 함수가 0 을 닫아버렸다. 0 은 표준입력인데 입력을 못하게 막는 거다.
	근데 dup 함수가 fd 를 인자로 받아서 실행하니까 close 가
	닫은 0 을 대체하게된다. 0 을 다시 사용할 수 있게 되는거다.0 대신에 fd 근데 fd 에는 0 이 들어있음.:
	gets 는 원래 입력을 받는 함수인데 0 이 닫혀버려서 입력을 못받고 대체된 fd 로 넘어온 a.txt 의
	내용을 입력받아서 buff 에 저장하고있다.
	그래서 위의 printf(%s , buff)는 a.txt 의 내용을 출력한다.

	*/
}

/*
Quiz. 1

임의의 난수를 10 개 발생시켜서 이 값을 배열에 저장하고
배열에 저장된 값을 파일에 기록한다(중복은 안됨).
그리고 이 값을 읽어서 Queue 를 만든다.
이후에 여기 저장된 값중 짝수만 선별하여 모두 더한 후에
더한 값을 파일에 저장하고
저장한 파일을 읽어 저장된 값을 출력하도록 한다.
```

(반드시 System Call 기반으로 구현하도록 함 - 성능이 압도적임)

*/

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

#include <unistd.h>
#include <fcntl.h>

int extract_idx;

typedef struct __queue
{
	int data;
	struct __queue *link;
} queue;

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_rand_arr(int *arr, int size)
```

```
{
	int i;

	for(i = 0; i < size; i++)
	{
redo:
		arr[i] = rand() % 10 + 1;

		if(is_dup(arr, i))
		{
			printf("%d dup! redo rand()\n", arr[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]);
}

queue *get_queue_node(void)
{
	queue *tmp;
	tmp = (queue *)malloc(sizeof(queue));
	tmp->link = NULL;
	return tmp;
}

void enqueue(queue **head, int data)
{
	if(*head == NULL)
```

```
	{
		*head = get_queue_node();
		(*head)->data = data;
		return;
	}

	enqueue(&(*head)->link, data);
}

void extract_even(queue *head, int *extract)
{
	queue *tmp = head;

	while(tmp)
	{
		if(!(tmp->data % 2))
			extract[extract_idx++] = tmp->data;
		tmp = tmp->link;
	}
}

int main(void)
{
	int i, fd, len, sum = 0;
	char *convert[10] = {0};
	int arr[11] = {0};
	char tmp[32] = {0};
	int extract[11] = {0};
	int size = sizeof(arr) / sizeof(int) - 1;
	queue *head = NULL;

	srand(time(NULL));

	init_rand_arr(arr, size);
	print_arr(arr, size);
```

```
	for(i = 0; i < size; i++)
		enqueue(&head, arr[i]);

	extract_even(head, extract);
	printf("\nExtract:\n");
	print_arr(extract, extract_idx);

	fd = open("log.txt", O_CREAT | O_WRONLY | O_TRUNC,
0644);

	for(i = 0; i < extract_idx; i++)
		sum += extract[i];

	sprintf(tmp, "%d", sum);
	write(fd, tmp, strlen(tmp));
	close(fd);

#if 0
	for(i = 0; i < extract_idx; i++)
	{
		int len;
		char tmp[32] = {0};

		sprintf(tmp, "%d", extract[i]);
		len = strlen(tmp);
		convert[i] = (char *)malloc(len + 1);
		strcpy(convert[i], tmp);
		printf("tmp = %s\n", tmp);
	}
#endif

	return 0;
}
```

/*

Quiz. 2

카페에 있는 50 번 문제(성적 관리 프로그램)을 개조한다.
어떻게 개조할 것인가 ?
기존에는 입력 받고 저장한 정보가 프로그램이 종료되면 날아갔다.
입력한 정보를 영구히 유지할 수 있는 방식으로 만들면 더 좋지 않을까 ?

* 조건
1. 파일을 읽어서 이름 정보와 성적 정보를 가져온다.
2. 초기 구동시 파일이 없을 수 있는데
   이런 경우엔 읽어서 가져올 정보가 없다.
3. 학생 이름과 성적을 입력할 수 있도록 한다.
4. 입력된 이름과 성적은 파일에 저장되어야 한다.
5. 당연히 통계 관리도 되어야한다(평균, 표준 편차)
6. 프로그램을 종료하고 다시 키면
   파일에서 앞서 만든 정보들을 읽어와서 내용을 출력해줘야 한다.
7. 언제든 원하면 내용을 출력할 수 있는 출력함수를 만든다.
   [특정 버튼을 입력하면 출력이 되게 만듬]
(역시 System Call 기반으로 구현하도록 함)

*/

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

typedef struct __queue
{
```

```
	int score;
	char *name;
	struct __queue *link;
} queue;

void disp_student_manager(int *score, char *name, int size)
{
	char *str1 = "학생 이름을 입력하시오: ";
	char *str2 = "학생 성적을 입력하시오: ";
	char tmp[32] = {0};

	write(1, str1, strlen(str1));
	read(0, name, size);
	write(1, str2, strlen(str2));
	read(0, tmp, sizeof(tmp));

	*score = atoi(tmp);
}

void confirm_info(char *name, int score)
{
	printf("학생 이름 = %s\n", name);
	printf("학생 성적 = %d\n", score);
}

queue *get_queue_node(void)
{
	queue *tmp;

	tmp = (queue *)malloc(sizeof(queue));
	tmp->name = NULL;
	tmp->link = NULL;

	return tmp;
```

```
}

void enqueue(queue **head, char *name, int score)
{
	if(*head == NULL)
	{
		int len = strlen(name);
		(*head) = get_queue_node();
		(*head)->score = score;
		(*head)->name = (char *)malloc(len + 1);
		strncpy((*head)->name, name, len);
		return;
	}

	enqueue(&(*head)->link, name, score);
}

void print_queue(queue *head)
{
	queue *tmp = head;

	while(tmp)
	{
		printf("name = %s, score = %d\n", tmp->name, tmp->score);
		tmp = tmp->link;
	}
}

void remove_enter(char *name)
{
	int i;

	for(i = 0; name[i]; i++)
		if(name[i] == '\n')
```

```
                name[i] = '\0';
}

int main(void)
{
        int cur_len, fd, btn = 0;
        int score;

        // Slab 할당자가 32 byte 를 관리하기 때문에 성능이 빠름
        char name[32] = {0};
        char str_score[32] = {0};
        char buf[64] = {0};

        queue *head = NULL;

        for(;;)
        {
                printf("1 번: 성적 입력, 2 번: 파일 저장, 3 번: 파일 읽기, 4 번:
종료\n");
                scanf("%d", &btn);

                switch(btn)
                {
                        case 1:
                                disp_student_manager(&score,            name,
sizeof(name));
                                remove_enter(name);
                                confirm_info(name, score);

                                enqueue(&head, name, score);
                                print_queue(head);
                                break;

                        case 2:
```

```
                    // 만약 파일 없다면 생성
                    // 있다면 불러서 추가
                    if((fd = open("score.txt", O_CREAT | O_EXCL | O_WRONLY, 0644)) < 0)
                        fd = open("score.txt", O_RDWR | O_APPEND);

                    /* 어떤 형식으로 이름과 성적을 저장할 것인가 ?
                       저장 포맷: 이름,성적\n */
                    strncpy(buf, name, strlen(name));
                    cur_len = strlen(buf);
                    //printf("cur_len = %d\n", cur_len);
                    buf[cur_len] = ',';
                    sprintf(str_score, "%d", score);
                    strncpy(&buf[cur_len + 1], str_score, strlen(str_score));
                    buf[strlen(buf)] = '\n';
                    //printf("buf = %s, buf_len = %lu\n", buf, strlen(buf));

                    write(fd, buf, strlen(buf));

                    close(fd);

                    break;

                case 3:
                    if((fd = open("score.txt", O_RDONLY)) > 0)
                    {
                        int i, backup = 0;
                        // 이름 1,성적 1\n
                        // 이름 2,성적 2\n
                        // .....
                        // 이름 n,성적 n\n
```

```
                        read(fd, buf, sizeof(buf));

                        for(i = 0; buf[i]; i++)
                        {
                            if(!(strncmp(&buf[i], ",", 1)))
                            {
                                strncpy(name, &buf[backup], i - backup);
                                backup = i + 1;
                            }
                            if(!(strncmp(&buf[i], "\n", 1)))
                            {
                                strncpy(str_score, &buf[backup], i - backup);
                                backup = i + 1;
                                enqueue(&head, name, atoi(str_score));
                            }
                        }

                        print_queue(head);
                    }
                    else
                        break;

                    break;

                case 4:
                    goto finish;
                    break;

                default:
                    printf("1, 2, 3, 4 중 하나 입력하셈\n");
                    break;
            }
```

```
        }

finish:
        return 0;
}
```