TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/3/21
수업일수	20 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. 함수 정리

- sprintf( )
특정한 규칙에 맞게 문자열로 변환 출력한다.
%d : 인자를 정수로 출력한다.
%f : 인자를 실수로 출력한다.
%s : 인자를 문자열로 출력한다.

- dup( )
( )안의 내용을 복제한다. 성공시 반환하는 갑은 새로운 숫자를 반환하고 오류시 반환하는 값은 -1이다.

- atoi( )
<stdlib.h> 헤더파일에 선언 되어있고 문자열로 표현된 정수를 int 형으로 반환해준다. 매개변수로 포인터를 받는다. 만일 atoi 함수의 인자에 숫자가 아닌 값이 오면 0을 반환한다.

- atof ( )
atoi( )와 같이 마찬가지로 문자열로 표현된 정수를 float 형으로 반환 시켜준다.

- strcmp( ), strncmp( )
strcmp(비교대상 문자열, 비교할 문자열), strncmp(비교대상 문자열, 비교할 문자열, 배교할 대상의 갯수)
문자열이 같을 경우 0을 반환하고 다를 경우 0이 아닌 수를 반환한다. 인자로 문자열을 받는다.

- strlen( )
NULL 문자를 제외한 문자열의 길이를 구해준다. <string.h>헤더파일에 선언되어 있다.

- strcpy( )

문자열 전체를 모두 복사시키는 함수이다.

- strncpy( )

원하는 문자열 길이 만큼을 복사하는 함수이다. strncpy(복사받을 변수 명, 복사 할 변수명, 복사할 길이)로 사용하면 된다.

- gets( )

1 줄을 입력할 때 사용한다. 엔터키를 입력할 때 까지 그대로 읽어 들이고 읽은 문자열은 줄 바꿈에서 ₩0 이 붙여진 배열에 저장된다.

2. Quiz 해석

```
#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));//모니터에 str1을 출력한다.
	read(0, name, size);//키보드로부터 이름을 입력 받는다.
	write(1, str2, strlen(str2));//모니터에 str2를 출력한다.
	read(0, tmp, sizeof(tmp));//키보드로부터 tmp 에 성적을 입력받음

	*score = atoi(tmp);//tmp 에 받은 성적을 int 형으로 표현해줌
}

void confirm_info(char *name, int score)//이름과 성적을 보여준다.
{
	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;
                                // 이름1,성적1₩n
                                // 이름2,성적2₩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;
}
```

3. 예제풀이

예제 1)

```
#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);
        //1이 닫히면 출력 안됨
        dup(fd);
        //종료 된 것을 복사,1번의 역할을 함
        //3번인 a.txt 에 기록됨
        printf("출력될까?₩n");
        return 0;
}
```

~결과

printf("출력될까?₩n");가 출력되지않고 a.txt 에 저장된다.

예제 1-2)

```
#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);//   씹힌다.
        printf("출력될까?₩n");
        return 0;
}
```

~결과

```
출력될까?
```

gets()는 씹히게 된다. 아무런 동작을 하지 못한다.

예제 2)

```
#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(buff);
        return 0;
}
```

~결과

a.txt 있는 내용을 바꾼 뒤 출력하면 '바꾼 a.txt'라고 바뀐 a.txt 가 출력됨

4. pipe communication

ps - ef : 현재 실행되는 프로세스들을 보여준다.

```
xeno@xeno-NH:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
.
.
.(프로세스가 많이 보여진다)
```

ps -ef | grep bash : 찾을 bash 와 구동되는 bash 가 보여진다. 구동시킨 프로세스를 보여줌.

```
xeno@xeno-NH:~$ ps -ef | grep bash
xeno        5602   5595   0 13:58 pts/2    00:00:00 bash
xeno        5615   5602   0 13:58 pts/2    00:00:00 grep --color=auto bash
```

ps -ef | grep bash | grep -v greb : bash 를 찾는 프로세스를 제외한 구동시킨 프로세스를 보여준다.

```
xeno@xeno-NH:~$ ps -ef | grep bash | grep -v greb
xeno        5602   5595   0 13:58 pts/2    00:00:00 bash
xeno        5619   5602   0 13:58 pts/2    00:00:00 grep --color=auto bash
```

ps -ef | grep bash | grep -v greb | awk '{print $2}' : PID(프로세스 아이디, 고유한 식별번호)를 보여준다.

```
xeno@xeno-NH:~$ ps -ef | grep bash | grep -v greb | awk '{print $2}'
5602
5625
```

5. tail command

tail 명령어는 파일 내용의 마지막부터 읽을 때 주로 사용한다.

-tail -c 20 1.c
뒤에서부터 20단어를 출력한다.

```
xeno@xeno-NH:~/proj/0321$ tail -c 50 1.c
   printf("출력될까?₩n");
          return 0;
}
```

-tail -n 10 /var/log/messages
뒤에서부터 10줄을 출력한다.

```
xeno@xeno-NH:~/proj/0321$ tail -n 10 1.c
          fd=open("a.txt",O_WRONLY|O_CREAT|O_TRUNC,0644);
          close(1);
         //1이 닫히면 출력 안됨
          dup(fd);
         //종료 된 것을 복사,1번의 역할을 함
         //3번인 a.txt 에 기록됨
          printf("출력될까?₩n");
          return 0;
}
```

예제 3)

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

int main(int argc, char *argv[])
{
        int i;
        char ch='a';
        int fd = open(argv[1],O_WRONLY|O_CREAT|O_TRUNC,0644);
        lseek(fd,512-1,SEEK_SET);
        //512번째에 가져다 놓음
        write(fd,&ch,1);
        //a 라는 것을 씀
        close(fd);

        //이런 코드는 마스터 부트 레코드 라는 곳에서 사용됨

        return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0321$ ./a.out mbr.txt
xeno@xeno-NH:~/proj/0321$ xxd mbr.txt
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

00000050: 0000 0000 0000 0000 0000 0000 0000 0000	00000160: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000	00000170: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000	00000180: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 0000 0000 0000 0000 0000 0000 0000 0000	00000190: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000090: 0000 0000 0000 0000 0000 0000 0000 0000	000001a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000	000001b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000	000001c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000	000001d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000	000001e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000	000001f0: 0000 0000 0000 0000 0000 0000 0000 0061 .....a

lseek 의 사용을 잘 몰라 512-1 대신 1, -1, -2를 입력해보았다.

```
xeno@xeno-NH:~/proj/0321$ vi 3.c
xeno@xeno-NH:~/proj/0321$ gcc 3.c
xeno@xeno-NH:~/proj/0321$ ./a.out mbr.txt
xeno@xeno-NH:~/proj/0321$ xxd mbr.txt
00000000: 0061                                     .a
xeno@xeno-NH:~/proj/0321$ vi 3.c
xeno@xeno-NH:~/proj/0321$ gcc 3.c
xeno@xeno-NH:~/proj/0321$ ./a.out mbr.txt
xeno@xeno-NH:~/proj/0321$ xxd mbr.txt
00000000: 61                                       a
xeno@xeno-NH:~/proj/0321$ vi 3.c
xeno@xeno-NH:~/proj/0321$ gcc 3.c
xeno@xeno-NH:~/proj/0321$ gcc 3.c
xeno@xeno-NH:~/proj/0321$ ./a.out mbr.txt
xeno@xeno-NH:~/proj/0321$ xxd mbr.txt
00000000: 61                                       a
```

0이하로는 위치가 0으로 고정되어 있는 것을 확인했다

예제 4)

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

int main(void)
{
        int fd, ret;
        char buf[1024];
        mkfifo("myfifo");
        fd=open("myfifo",O_RDWR);
        for(;;)
        {
                ret=read(0,buf,sizeof(buf));
                //blocking - 입력이 다 끝날 때 까지
                buf[ret-1]=0;
                printf("Keyboard Input : [%s]₩n",buf);
                read(fd,buf,sizeof(buf));
                //blocking - 입력이 다 끝날 때 까지
                buf[ret-1]=0;
                printf("pipe input : [%s]₩n",buf);

                return 0;
        }
}
```

~결과
xeno@xeno-NH:~/proj/0321$ ./a.out
hello!!
Keyboard Input : [hello!!]
pipe input : [hello!
]

터미널에서 mkfifo myfifo 입력하면 노란색의 myfifo 가 생성된다. 이후 실행파일을 실행시키면 새로운 터미널을 열어 cat > myfifo 를 입력한 후 메인 터미널에서 문자를 보내고 새로운 터미널에서 문자를 보내면 메인 터미널에서 각 터미널의 쓴 문자들을 볼 수 있음.

-blocking VS nonblocking
다수가 빠르게 통신할 때에는 nonblocking 이 좋고, 순차적으로 진행 되어야 할 때는 blocking 이 좋다.
read()함수 같은 경우는 blocking 이어서, 위의 4번 예제를 보면 키보드로 입력받을 때 까지 제어권을 넘기지 않는다.

ls -al /dev
ls -al

- 로 시작하는 것 : 파일
b 로 시작하는 것 : 블록 디바이스
c 로 시작하는 것 : 캐릭터 디바이스
d 로 시작하는 것 : 디렉토리
p 로 시작하는 것 : 파이프

캐릭터 디바이스와 블록 디바이스 차이점

: c 는 순서를 가지고 b 는 특정 단위를 가지고 움직인다. c 에 해당하는 것은 키보드, 모니터, 비디오 등이고, b 에 해당하는 것은 하드디스크, DRAM 이 해당한다.

DRAM 이 블록 디바이스인 이유는 메모리 내의 기계어 분석을 하게 되면 call ,jmp 같은 명령어들은 순서에 관계없이 실행되는 것을 생각해 보면 알 수 있다.

예제 5)

```
#include<stdio.h>
#include<fcntl.h>
int main(void)
{
        int fd, ret;
        char buf[1024];
        fd = open("myfifo",O_RDWR);
        fcntl(0,F_SETFL, O_NONBLOCK);//여기 두 문장에서 NONBLOCK 으로 권한 설정을 해주었다
        fcntl(fd,F_SETFL, O_NONBLOCK);
        for(;;)
        {
                if((ret=read(0,buf,sizeof(buf)))>0)
                {
                        buf[ret-1]=0;
                        printf("Key board Input : [%s]₩n)",buf);
                }
                if((ret=read(fd,buf,sizeof(buf)))>0)
```

```
                {
                        buf[ret-1]=0;
                        printf("Pipe Input : [%s]₩n",buf);
                }
        }
        return 0;
}
```

~결과

```
xeno@xeno-NH:~/proj/0321$ ./a.out
Pipe Input : [adfafadf]
asfadfa
Key board Input : [asfadfa]
)Pipe Input : [dafadf]
```

Pipe Input 을 먼저 보낸 것을 확인할 수 있고 따라서 순서에 관계 없이 통신 할 수 있다. Key board Input 이 치기 전에 Pipe Input 을 보내도 Pipe Input 이 나타날 수 있게 되었다.

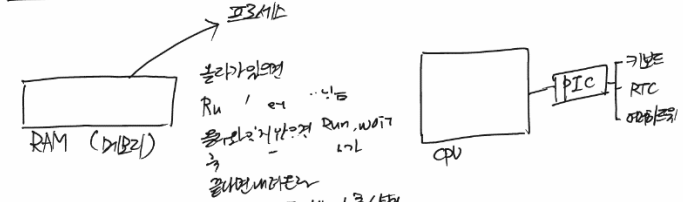
6. Principle of Multi-Tasking

3/21 ★그림설명하시오.

톤 CPU에는 Wait Queue와 Run Queue 있음.

CPU는 프로세스를 관리함. 프로세스는 CPU의 추상화이다. (= 가상의 CPU이다)

실행파일 a.out → 내부 → 소스코드 int main() { } → 내부 → 기계어 push 「범용레지스터」 add call



ps-ef 로 본 프로세스들이 모두 프로세스의 추상화

CPU 안에는 범용레지스터가 안에 있는데 기계어안에도 범용레지스터가 사용

Q 프로세스가 여러개 있는데 어떻게 동시에 실행되는가?

ps-ef에 보면 프로세스들이 동시다발적으로 실행됨

- 정말 동시다발적인가? → 싱글코어
- 방송댄서는 무시? → 멀티코어

context switching ★ 컴퓨터 구조론

↓

multi-tasking 가능

CPU는 오직 한순간에 한가지의 연산만수행

CPU 주파수 : 2.4GHz (f) / 주파수가 커지면 주기가 짧아짐.

주기 : $\frac{1}{f}$

1 clock : 명령어하나를 수행하는데 소요하는시간

$G : 10^9$ $T = \frac{1}{2\times10^9} = \frac{5}{10^{10}} = 5\times10^{-10}$

1초 : T ⇒ 20억번.

1초에 20억개를 처리한다.

이주 빠른속도로 여러프로세스들이 제어권을 넘겨주며 CPU를 사용한다면 우리가 느끼지 못하는 순간에 모든 작업이 완료된다

task_struct

- thread_struct
- thread_union

] 에 레지스터정보들을 저장.

수동시라면 하드웨어 레지스터들이 저장되어 제어권이 돌아오면 저장되었던 레지스터를 다시 실행

↳ context_switching

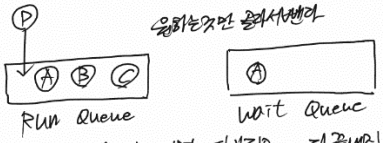
context switching

CPU를 뺏기위해경쟁. CPU의 프로세스들로부터 실행될수있음.

그래서 우선순위가 존재함. 범용 OS는 우선순위가 고정될수없음. (인공지능은 X)

우선순위마다 주어진 할당시간이 있음.

multitasking



· unQueue해야지 실행한프로세스면 파기되고 다른데로 이동하면 wait queue에 넣어

system call 하면 wait Queue 에 저장되고 인터럽트처리. CPU의 PIC에 들어옴