TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
20일차 (2018. 03. 21)

학습 내용 복습

Quiz. 2 카페에 있는 50번 문제 (성적 관리 프로그램)을 개조한다. 어떻게 개조할 것 인가? 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. 입력한 이름과 성적은 파일에 저장 되어야 한다.
5. 당연히 통계 관리도 되어야 한다(평균, 표준 편차)
6. 프로그램을 종료 하고 다시 키면 파일에서 앞서 만든 정보들을 읽어와서 내용을 출력해줘야 한다.
7. 언제든 원하면 내용을 출력할 수 있는 출력함수를 만든다. 특정 버튼을 입력하면 출력이 되게 만든다.

(역시 System Call 기반으로 구현하도록 함)

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

//헤더파일

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

//구조체 선언. 원래는 RB 트리로 관리한다.

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

    write(1, str1, strlen(str1));    // 위에 입력한 str1(이름)을 모니터에 (기본) 출력한다.
    read(0, name, size);             // 키보드로부터 이름을 입력 받는다.
    write(1, str2, strlen(str2));    // 위에 입력한 str2(성적)을 모니터에 (기본) 출력한1다.
    read(0, tmp, sizeof(tmp));       // 키보드로부터 이름을 입력 받는다.
                                     // 정수를 int형으로 선언해 저장했기 때문에 tmp를 atoi를 통해 int
    *score = atoi(tmp);              }   형으로 변환해 반환해준다. 즉, tmp에 받은 성적을 int형으로 표현

void confirm_info(char *name, int score)        // (확인을 위해) 이름과 성적을 창에 출력해 준다.
{   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) // 중간에 개행이 나오면서 문자열이 이상해지는 것을 막기 위해
 {  int i;                                    ₩n 일 때 ₩0 으로 바뀌게 한다.
    for(i = 0; name[i]; i++)
         if(name[i] == '₩n')
         name[i] = '₩0';          }

int main(void) {
      int cur_len, fd, btn = 0;
       int score;
     char name[32] = {0};         // Slab 할당자가 32 byte 를 관리하기 때문에 성능이 빠르다.
     char str_score[32] = {0};
    char buf[64] = {0};
    //for문을 통해서 계속 동작을 할 수 있게 해준다. 스위치로 계속 동작을 확인을 위해.
    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:        // 위에 만든 함수를 그대로 보여준다.
```

```
    disp_student_manager(&score, name, sizeof(name));
        remove_enter(name);                // 중간에 개행이 보기 안 좋아서 넣어준 것.
    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);          // append를 이용해서 이어붙이기를 한다.
 /* 어떤 형식으로 이름과 성적을 저장할 것인가 ?  저장 포맷: 이름, 성적₩n */
          strncpy(buf, name, strlen(name));        // strncpy (복사할 곳, 복사할 거, 몇 개)
          cur_len = strlen(buf);                   // 현재 위치를 저장한다. buf의 길이가 결국 현재 위치
          //printf("cur_len = %d₩n", cur_len);
          buf[cur_len] = ',';                      // , 로 구분을 위에 넣는다.
          sprintf(str_score, "%d", score);         //  숫자를 저장하기 위해서 sprint로 문자로 바꿔준다.
          strncpy(&buf[cur_len + 1], str_score, strlen(str_score));
          //  이제 앞에 저장된 위치에 뒷부분 부터 저장을 위해 +1 부터 score를 받는다.
          buf[strlen(buf)] = '₩n';                 // 마지막으로 개행이 되게 한다.
           //printf("buf = %s, buf_len = %lu₩n", buf, strlen(buf));
          write(fd, buf, strlen(buf));             // 저장한것을 열어 놓은 score.txt 안에 쓴다.
           close(fd);
           break;
case 3:
     if((fd = open("score.txt", O_RDONLY)) > 0) {        // 읽기 위해 파일을 연다.
          int i, backup = 0;
 // 이름1,성적1₩n
 // 이름2,성적2₩n
 // ......
 // 이름n,성적n₩n
           read(fd, buf, sizeof(buf));             // 파일의 내용을 읽는다.
 for(i = 0; buf[i]; i++) {                         // 안에 있는 내용을 다 돌려서 본다.
          if(!(strncmp(&buf[i], ",", 1))) {        // 비교함수로 ','가 오면 이름을 출력하고 백업 위치를 저장
           strncpy(name, &buf[backup], i - backup);
           backup = i + 1; }
           if(!(strncmp(&buf[i], "₩n", 1))) {  // 비교함수로 ','가 오면 성적 값만 뽑아 출력하고 백업위치를 저장
           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;
}
```

* man -s2 함수명 =시스템 콜 관련 명령어, 함수 등의 메뉴얼을 알려준다.
* man 함수명 = 명령어 , 함수 등의 메뉴얼을 알려준다.

ps – ef = 현재 실행되는 프로세스들을 보여준다.
ps -ef | grep bash = 찾을 bash 와 구동되는 bash 가 보여진다. 구동시킨 프로세스를 보여준다.
ps -ef | grep bash | grep -v greb = bash를 찾는 프로세스를 제외한 구동시킨 프로세스를 보여준다.
ps -ef | grep bash | grep -v greb | awk '{print $2}' = PID(프로세스 아이디, 고유한 식별번호)를 보여준다.

* tail 명령어는 파일 내용의 마지막부터 읽을 때 주로 사용한다. 파일이 많을 때, 어떤 파일인지 쉽게 찾기 위한 명령어이다.
tail -c 20 1.c = tail –c [갯수] [파일명] = 파일 뒤에서부터 20단어를 출력한다.
tail -n 20 /var/log/messages = tail –n [개수] [파일명] = 뒤에서부터 20줄을 출력한다.

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.

학습 예제

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

int main (void)
{   int fd;
   fd = open("a.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    close (1);  // 표준 출력 1 , 출력이 닫힘
    dup(fd);  //dup는 중복시키다는 뜻, 종료 된 것을 복사하게 된다. fd가 1번의 역할을 하게 된다.
                  모니터의 역할을 fd(3)이 받아감.
    printf("출력될까?₩n");
   return 0;                }
      > printf("출력될까?₩n");가 출력되지않고 a.txt 에 저장된다.

#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(buff);
return 0;          }
          > 출력될까? 가 출력 된다.
```

gets()는 무시하게 된다. 아무런 동작을 하지 못한다. '출력될까?'는 파일에서 입력을 받은 것이다. 0을 닫아서 키보드가 입력을 할 수가 없기 때문이다. dup가 대체 하게 되는 것으로 0번을 지우고 fd가 대체된다.
입력 자체가 파일이 된다는 것이다. a.txt 자체 내용을 받게 되는 것이다.
gets는 화면에서 뭔가를 입력 받을 수 있게 해준다. 또한 gets는 보안상 문제가 있는 함수이기 때문에 warning 뜨는 것은 어쩔 수 없다.

위의 예제에서 dup()가 대체되는 것은 프로그램에서 한번 0이나 1번 닫아버리면 다시 열 수 없기 때문이다. 다시 여는 작업은 해킹과 비슷한 개념이다. 사용자가 한 번 닫으면 두 번 다시 열 수 없다. 만약 사용자가 임의로 닫고 열고의 변경을 할 수 있게 해준다고 가정해보자. 큰 서버에서 사용자의 데이터를 입력 받다가 더 이상 받을 수 없게 되기 때문이다.

* blocking VS nonblocking : 어떤 때이냐에 따라 다르다. 다수가 빠르게 통신할 때에는 nonblocking 이 좋고, 순차적으로 진행 되어야 할 때는 blocking 이 좋다. read()함수 같은 경우는 blocking 이어서, 밑의 예제를 보면 키보드로 입력 받을 때까지 제어권을 넘기지 않는다.

> 이 예제의 실행방법은 조금 다르다. 터미널에서 mkfifo myfifo 입력하면 노란색의 myfifo가 생성된다. 이후 실행파일을 실행시킨다. 그 후, 그 창에 맞물린 새로운 터미널을 열어 cat > myfifo 를 입력한다. 이후 메인 터미널에서 문자를 보내고 새로운 터미널에서 문자를 보내면 메인 터미널에서 각 터미널의 쓴 문자들을 볼 수 있다.

```
#include <fcntl.h>
#include <stdio.h>
#include <string.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));
       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;
}
```

> 위의 실행 방법으로 실행시키면, 서로 한 번씩 주고 받기만 가능하다.

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

int main (void) {
    int fd, ret;
    char buf[1024];
    fd = open("myfifo", O_RDWR);
    fcntl(0, F_SETFL, O_NONBLOCK);
    fcntl(fd, F_SETFL, O_NONBLOCK);
    for(;;)
```

```
    {       if((ret=read(0,buf,sizeof(buf))) > 0)
            {   buf[ret-1] =0;
                printf("Keyboard Input :[%s]₩n", buf);          }
            if ((ret = read(fd, buf, sizeof(buf))) >0)
            {   buf[ret-1] =0;
                printf("Pipe Input :  [%s]₩n", buf);            }
    }
    close(fd);
    return 0;
}
```

시스템 프로그래밍