TI DSP, MCU 및
Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
19일차 (2018. 03. 20)

* 함수 포인터를 사용하는 이유 (리눅스 커널에서)
 리눅스 지원하는 파일시스템이 1000개가 넘는데, open, read, write 등의 명령어들을 사용할 때 사용되는 인자의 수가 매우 많다. 이를 매번 기록하여 사용할 수 없기 때문에 함수를 가리키는 포인터를 사용한다면 편리하고 효율적으로 사용가능하기에 함수 포인터를 사용한다.
리눅스를 할때 핵심 구조체 : tast_struct

*lseek함수
읽기 – 쓰기 포인터의 위치 즉, 다음에 읽거나 쓸 바이트의 위치를 변경할 수 있다. 파일에 대한 임의 접근을 가능하게 해주는 것이다. [SEEK_SET 파일의 시작 SEEK_CUR 현재 읽기/쓰기 포인터 위치 SEEK_END 파일의 끝] 을 뜻한다. 이동 후의 읽기/쓰기 포인터를 반환하여 SEEK_END를 이용해 파일의 크기를 구할 수 있다. fsize = lseek(fd, 0, SEEK_END)는 SEEK_END에서 0위치에 해당하는 포인터의 위치가 곧 파일의 끝이므로 파일 크기를 구할 수 있다.

* tar 압축 예제
```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
  char fname[20];
  int fsize;
} F_info;

int file_size(int fd)
{
    int fsize, old;
    old = lseek(fd, 0, SEEK_CUR);        // old는 현재 위치를 저장한다.
    fsize = lseek(fd, 0, SEEK_END);      // 전체 파일의 크기를 fsize에 저장한다.
    lseek(fd, old, SEEK_SET);            // 파일 포인터를 원상복귀 시켜라. 시작에서 old까지 이동하므로
    return fsize;                        // 파일 전체 사이즈를 리턴한다.
}

int main(int argc, char*argv[])
{
 int src, dst, ret;
 char buf[1024];
 F_info info;
 int i;
 dst = open(argv[argc - 1], O_WRONLY | O_CREAT | O_TRUNC,0644);
```
/*argv[argc -1]인 이유는 마지막 인자를 구하기 위해서이다. 배열의 시작은 0이니까 -1을 하면 마지막 인자를 구할 수 있다. 쓰기 전용으로 파일을 생성하고 이미 생성된 파일이 있다면 생성되지 않는 조건이 걸려있

다. 압축하는 것이니까 모든 파일을 묶을 파일명이 필요하므로 마지막 인자는 이름이다.*/
```
  for(i=0; i< argc - 2; i++)        //마지막 인자는 이름이니까 그 전 인자까지 묶어야 해서 -2인 것
   {
     src = open(argv[i+1], O_RDONLY); // 실행파일을 제외하기 위해서 i+1인 것이다.
     strcpy(info.fname, argv[i+1]);          /*info.fname 에 argv[i+1]을 저장. 첫번째 파일 이름을 fname에
                                             사한 것. 이는 압축 한다의 개념을 생각해보자! 어떤 파일을 압축할
                                             것인가? 압축할 파일의 이름이 필요하다.*/
     info.fsize = file_size(src);     //처음 연 파일의 사이즈를 fsize에 저장한다.
     write(dst, &info, sizeof(info));         //dst(res.tar)에 구조체의 주소값과 크기 쓴다.
     while(ret = read(src, buf, sizeof(buf)))
           write(dst, buf, ret);
     close(src);   //조건을 만족하면 첫 번째 파일을 닫아준다.
   }
   close(dst);
   return 0;
}
```
> 컴파일을 하고선 ./a.out(실행파일이다) a.txt b.txt c.txt res.tar 로 실행 시켜주면 res.tar라는 압축파일에 a.txt b.txt c.txt가 저장된다. 이때 xxd res.tar 를 치면 각각 파일에 들어있는 내용을 볼 수 있다.

*tar 해제 예제
```
#include <fcntl.h>
#include <unistd.h>

typedef struct{
  char fname[20];
  int fsize;
} F_info;
#define min(x,y)   (((x) < (y)) ? (x) : (y)) //x나 y에 복잡한 수식이 올 수 있으므로 ()가 필요하다.

int main (int argc, char *argv[])
{
    int src, dst, len, ret;
   F_info info;
    char buf[1024];
    src = open(argv[1], O_RDONLY);      //src에 압축을 풀 ret.tar를 넣는다.
    while(read(src, &info, sizeof(info)))
    {
        dst = open(info.fname, O_WRONLY | O_TRUNC | O_CREAT,0644);
        while(info.fsize > 0)
        {
           len = min(sizeof(buf), info.fsize);    //fsize가 위에서 선언한 1024를 넘을 수 있어서 최소값을 찾음
         ret = read(src, buf, len);           //ret에 읽은 값을 저장한다.
```

```
            write(dst, buf, ret);                //src에서 읽은 만큼의 값을 dst에 쓴다.
            info.fsize -=ret;          // 남은 데이터들을 처리한다. Ret이 읽은 바이트 수이기 때문이다.
           }
        close(dst);
    }
     close(src);
    return 0;
}
```

> 컴파일 후에 ./a.out res.tar을 하면 res.tar 압축이 풀려 a.txt b.txt c.txt가 생성된다.

문제 1) 임의의 난수를 10개 발생시켜서 이 값을 배열에 저장하고 배열에 저장된 값을 파일에 기록한다. (중복은 안됨)
그리고 이 값을 읽어서 Queue를 만든다.
이후에 여기 저장된 값 중 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 <stdio.h>
#include <stdlib.h>
#include <time.h>

 int main(void) {
    int i, j;
    int arr[11]; //저장할 배열 선언
    int num = 0;
    int check = 0;

      for(i = 0; i < 10; i++)
       {        /*   arr[i] = (rand()%50)+1;  1~50(n) 범위 난수 생성과 배열 저장
                      그러나 이럴 경우, 중복이 될 수 있다.            */
        num = (rand()%50)+1;  //랜덤 생성
        for(j = 0; j < i; j++)  //중복 확인 반복문
        {          if(arr[j] == num)
           {               i -= 1;         // i를 난수 생성을 한 번 더 하기 위함.
                           check = 1;             }
         if(check ==0)
          {               arr[i] = num;          }
         else
          {               check = 0;             }
        }
                printf("arr[%d] = %d₩n", i, arr[i]); //배열에 저장된 수 확인
        }
```

```
 return 0;                    }
```
> 난수 생성하고 저장하는 것까지 밖에 풀지 못했습니다...

>선생님 답안

```
#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) // 값이 같을 경우, 참이 되고
{                                  // 값이 다를 경우, 거짓이 된다.
    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)    //배열과 크기가 인자
{    int i;

    for(i = 0; i < size; i++) // i가 사이즈보다 작을 때까지 반복하므로 배열을 크기에 맞게 저장된다.
    {
redo:    //반복문을 수행하지 않고, 현재 반복문을 다시 한번 실행한다.
        arr[i] = rand() % 10 + 1; //난수 범위 지정 1~10의 수

        if(is_dup(arr, i))
        {
            printf("%d dup! redo rand()₩n", arr[i]);
            goto redo; //값이 참일 경우 위의 문장을 출력한다.
```

```
        }
    }               //redo로 인해 i값이 증가하지 않고 다시 반복되어 중복값은 입력되지 않음
}

void print_arr(int *arr, int size)   //위의 함수로 저장한 배열을 출력한다.
{    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)); //동적할당으로 값을 계산
    tmp->link = NULL;         //구조체의 링크를 '0'으로 초기화한다.
    return tmp;
}

void enqueue(queue **head, int data){
    if(*head == NULL)
    {
        *head = get_queue_node();  //head 값에 리턴 tmp 값 대입
        (*head)->data = data;       //head가 가르키는 data값에 data 기입
        return;
    }
    enqueue(&(*head)->link, data);
}

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

    while(tmp){
        if(!(tmp->data % 2))        //짝수이면 짝수의 값을 배열에 저장한다.
            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));          // 난수 생성
    init_rand_arr(arr, size);    //난수를 배열에 입력하는 함수
    print_arr(arr, size);

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

    extract_even(head, extract);            // 짝수일 경우 배열에 저장하다.
    printf("₩nExtract:₩n");
    print_arr(extract, extract_idx);

    fd = open("log.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    /* fd(파일식별자?)에서 파일을 쓰기 전용으로 생성한다.
    단, 이미 파일이 만들어진 경우 파일을 생성하지 않는다.
    */
    for(i = 0; i < extract_idx; i++)
        sum += extract[i];                  //짝수를 더해준다.

    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;
}
```