운영체제 - 과제 0

프로세스 로더

202246109
김기현
2024년 3월 29일

1. os0.c 의 전체 코드

C os0.c

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

typedef struct{
    unsigned char op;
    unsigned char len;
} code_tuple;

typedef struct{
    int pid;
    int arrival_time;
    int code_bytes;
} process;

int main(int argc, char* argv[]){
    process cur; //고정크기이기 때문에 malloc이 필요가 없다!
    code_tuple *codes;

        while(fread(&cur, sizeof(process), 1, stdin)){
            // fread()의 리턴값
            // 읽기 성공 : 읽은 전체 항목의 수   true
            // 읽기 실패 : count 보다 작음      false

            printf("%d %d %d\n", cur.pid, cur.arrival_time, cur.code_bytes);
            codes = (code_tuple*)malloc(cur.code_bytes);
            //가변크기이기 때문에 malloc으로 code_bytes의 값 만큼 메모리 할당

            fread(codes, cur.code_bytes, 1, stdin);
            for(int i = 0; i < cur.code_bytes / 2; i++){
                //tuple의 수는 code_bytes의 절반 (code_bytes가 10이다 -> 튜플은 5개)
                printf("%d %d\n", codes[i].op, codes[i].len);
            }
            free(codes); //혹시모를 free
        }
    return 0;
}
```

2. 주요 코드 설명

코드의 주요 로직

count

```
while(fread(&cur, sizeof(process), 1, stdin)){
```

while문을 사용해 fread()로 파일이 끝날 때 까지 읽는다.

fread()의 반환값

읽기 성공 : 읽은 전체 항목의 수

읽기 실패 : count 보다 작음

읽기 오류와 파일 끝 조건을 구분하려면 feof 또는 ferror 함수를 사용

참고 :https://learn.microsoft.com/ko-kr/cpp/c-runtime-library/reference/fread?view=msvc-170#return-value

```
codes = (code_tuple*)malloc(cur.code_bytes);
//가변크기이기 때문에 malloc으로 code_bytes의 값 만큼 메모리 할당

fread(codes, cur.code_bytes, 1, stdin);
for(int i = 0; i < cur.code_bytes / 2; i++){
    //tuple의 수는 code_bytes의 절반 (code_bytes가 10이다 -> 튜플은 5개)
    printf("%d %d\n", codes[i].op, codes[i].len);
}
free(codes); //혹시모를 free
```

cur.code_bytes 의 값의 크기로 codes에 메모리를 할당한다.

(cur.code_bytes / 2) 번 operation, length 를 출력한다.

free(codes)를 사용해 메모리 누수 방지

처음으로 돌아가 파일이 끝날 때 까지 반복

3. JOTA 제출 및 결과

1, 2차시도 - 오류 (stdin을 잘 이해하지 못함)

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

typedef struct{
    unsigned char op;
    unsigned char len;
} code_tuple;

typedef struct{
    int pid;
    int arrival_time;
    int code_bytes;
} process;

int main(int argc, char* argv[]){
    process *cur = (process*)malloc(sizeof(process));
    code_tuple *codes;
    FILE* fp;

    if((fp = fopen(argv[1], "rb")) == NULL){
        printf("File open error\n");
        return -1;
    }
        // printf("file : %s\n", argv[1]); //파일 읽혔는지 지울예정

        while(fread(cur, sizeof(process),1 , fp)){


            printf("%d %d %d\n", cur->pid, cur->arrival_time, cur->code_bytes);

            codes = (code_tuple*)malloc(cur->code_bytes);
            fread(codes, cur->code_bytes, 1, fp);
            for(int i = 0; i < cur->code_bytes / 2; i++){
                        // 20 / 2 10개 code[0] ~ code[9]

                printf("%d %d\n", codes[i].op, codes[i].len);
            }
        }


    return 0;
}
```

error!

1차 output 오류

Execution Results

✖ ✖ ✖ ✖ ✖

Test case #1: IR [0.003s,980.00 KB] (0/1)

Your output
File open error

2차 segmentation fault

Execution Results

✖ ✖ ✖ ✖ ✖

- Test case #1: RTE (segmentation fault) [0.136s,980.00 KB] (0/1)
- Test case #2: RTE (segmentation fault) [0.107s,980.00 KB] (0/1)
- Test case #3: RTE (segmentation fault) [0.113s,980.00 KB] (0/1)
- Test case #4: RTE (segmentation fault) [0.119s,980.00 KB] (0/1)
- Test case #5: RTE (segmentation fault) [0.106s,980.00 KB] (0/1)

3차시도 - 성공

```c
int main(int argc, char* argv[]){
    process *cur = (process*)malloc(sizeof(process));
    code_tuple *codes;
    //FILE* fp;
    /*
    if() == NULL){
        printf("File open error\n");
        return -1;
    }
    */

       // printf("file : %s\n", argv[1]); //파일 읽혔는지 지울예정

        while(fread(cur, sizeof(process),1 , stdin)){
                                          변경점

            printf("%d %d %d\n", cur->pid, cur->arrival_time, cur->code_bytes);

            codes = (code_tuple*)malloc(cur->code_bytes);
            fread(codes, cur->code_bytes, 1, stdin);
            for(int i = 0; i < cur->code_bytes / 2; i++){
                        // 20 / 2 10개 code[0] ~ code[9]

                printf("%d %d\n", codes[i].op, codes[i].len);
            }
        }
```

4차시도 - 성공 (최적화)

```c
int main(int argc, char* argv[]){
    process cur; //고정크기이기 때문에 malloc이 필요가 없다!
    code_tuple *codes;

        while(fread(&cur, sizeof(process), 1, stdin)){
            // fread()의 리턴값
            // 성공적으로 읽었을 때 : 읽은 바이트 수  true
            // 파일의 끝 : 0                    false
            // 읽기 도중 오류 : EOF              false

            printf("%d %d %d\n", cur.pid, cur.arrival_time, cur.code_bytes);
            codes = (code_tuple*)malloc(cur.code_bytes);
            //가변크기 이기 때문에 malloc 으로 code_bytes 의 size로 메모리 할당

            fread(codes, cur.code_bytes, 1, stdin);
            for(int i = 0; i < cur.code_bytes / 2; i++){
                //tuple의 수는 code_bytes의 절반 (code_bytes가 100이다 -> 튜플은 5개)
                printf("%d %d\n", codes[i].op, codes[i].len);
            }
            free(codes); //혹시모를 free
    }
```

4. 어려웠던 점 및 해결방안

1. stdin으로 입력을 받는다는게 처음에는 이해가 안갔다

$ ./os.c test.bin 형태로 받는건가?

Fp = fopen(argv[1], “rb”); 형태인 줄 알고 코드를 작성했으나

JOTA에서 입력 오류가 떴다.

해결 : pdf의 참고를 보고 바로 이해가 갔다!

2. code_tuple의 사이즈가 가늠이 안갔다

- 3개의 프로세스 정보가 저장됨
  - 0번 프로세스: 도착시간=0 코드길이=20 (0x14)
  - 1번 프로세스: 도착시간=0 코드길이=46 (0x2e)
  - 2번 프로세스: 도착시간=1 코드길이=30 (0x1e)
  - (코드 정보의 저장 순서에 유의: Little endian 방식 때문)

```
ubuntu@41983:~$ hexdump test.bin
0000000 0000 0000 0000 0000 0014 0000 0400 ac01
0000010 0600 8401 0900 9501 0700 b501 0100 9401
0000020 0001 0000 0000 0000 002e 0000 0200 7801
0000030 0500 7e01 0500 ba01 0900 6901 0100 ad01
0000040 0400 a001 0500 ad01 0600 a701 0100 7f01
0000050 0500 6c01 0400 a101 0300 0002 0000 0001
0000060 0000 001e 0000 0700 9901 0900 7101 0400
0000070 8701 0100 8901 0200 6501 0500 c101 0100
0000080 7a01 0200
```

처음엔 test.bin 내용의 순서가 왜 이러나 이해가 안갔다.

해결 : 제공해주신 영상을 보고 이해가 갔다!

3. best솔루션들을 보니 0.01s 가 나오고 나는 0.02s가 나왔다

- 프로세스 정보 (고정 크기)
  - PID: 프로세스 ID
  - 도착시간: 프로세스가 실
  - 코드길이: 프로그램 코드
- 코드 (가변 크기)

아하!

쓸데없이 cur을 malloc으로 할당하고 있었다.

기존에 없던 free(codes)도 추가해 주었다.

수정 결과