운영체제 과제0 : 프로세스 로더

201619460 이성규

코드:

전체 코드입니다.

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

typedef struct {
    int pid;            //ID
    int arrival_time;   //도착시간
    int code_bytes;     //코드길이(바이트)
} process;

typedef struct {
    unsigned char movement; // 동작
    unsigned char time;     // 길이(동작 수행 시간)
} tuple;


int main(int argc, char* argv[]) {

    process data; //파일 읽은거 담을 구조체(프로세스) 선언

    while(fread(&data, sizeof(process), 1, stdin) == 1) { //프로세스 읽기
        fprintf(stdout, "%d %d %d\n", data.pid, data.arrival_time, data.code_bytes); //프로세스 정보 출력

        int i = data.code_bytes/2; //i는 튜플의 갯수. 한개의 튜플이 2바이트 이므로 2로 나누어줌
        int j;

        for( j = 0 ; j < i ; j++ ) // 튜플 갯수만큼 반복.
        {
            tuple temporary; //파일 읽은거 담을 구조체(튜플) 선언

            if(fread(&temporary, sizeof(tuple) , 1, stdin) == 0 )    // 튜플 읽기, 예외처리
            {
                fprintf(stdout, "읽을 항목이 없습니다.\n");
                return 0;
            }

            fprintf(stdout, "%d %d\n", temporary.movement, temporary.time); // 튜플 출력
        }
    }

    return 0;
}
```

10번 라인 ~ 13번 라인에 튜플로 받을 구조체를 정의하였습니다. 원래는 구조체가 아닌 그냥 변수 두 개를 선언하고 각자 따로 1바이트씩 정보를 읽었으나 (아래의 오른쪽) 구조체같은 객체로 변수를 관리하는 게 더 좋겠다고 생각하여 코드를 수정하였습니다. (아래의 왼쪽)

```c
typedef struct {
    unsigned char movement; // 동작
    unsigned char time;     // 길이(동작 수행 시간)
} tuple;
```

```c
unsigned char movement = 0; // 동작
unsigned char time = 0;     // 길이(동작 수행 시간)
```

메인 함수의 20라인에서는 while 문을 돌며 fread함수를 통해 파일(stdin)로부터 정보를 받아옵니다.
fread함수가 성공하면 읽어온 항목 수를 리턴, 실패하면 0을 리턴하는 것을 이용하여 파일로부터 정보를 받아올게 더 이상 없을 때까지(0을 리턴할 때 까지) 파일을 읽습니다. 이를 통해 파일을 끝까지 읽습니다.

```c
20      while(fread(&data, sizeof(process), 1, stdin) == 1) { //프로세스 읽기
```

그 이후 읽어온 프로세스의 code_bytes를 통해 이후 이어지는 튜플 정보의 개수를 파악하고, 그 개수만큼 for문을 돌며 튜플 정보를 파일로부터 받고 출력합니다. (23라인 ~ 34라인)

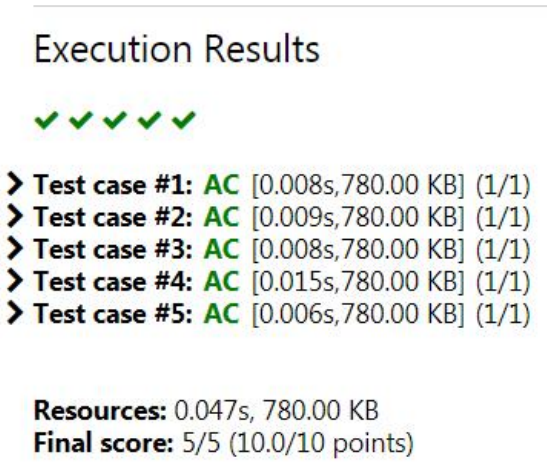
```
23          int i = data.code_bytes/2; //i는 튜플의 갯수. 한개의 튜플이 2바이트 이므로 2로 나누어줌
24          int j;
25
26          for( j = 0 ; j < i ; j++ ) // 튜플 갯수만큼 반복.
27          {
28              tuple temporary; //파일 읽은거 담을 구조체(튜플) 선언
29
30              if(fread(&temporary, sizeof(tuple) , 1, stdin) == 0 )    // 튜플 읽기, 예외처리
31              {
32                  fprintf(stdout, "읽을 항목이 없습니다.\n");
33                  return 0;
34              }
35
36              fprintf(stdout, "%d %d\n", temporary.movement, temporary.time); // 튜플 출력
37          }
```

딱히 필요하지는 않겠지만, 혹여나 파일로부터 읽을 항목이 없을 경우를 대비하여 (잘못된 정보가 입력된 경우) 예외처리를 해주었습니다.
(30라인 ~34라인)

실행 결과:

LMS에 주어진 bin 파일로 위 코드를 실행해본 결과, 그리고 JOTA 제출 결과입니다.

```
ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
0 4
1 172
0 6
1 132
0 9
1 149
0 7
1 181
0 1
1 148
1 0 46
0 2
1 120
0 5
1 126
0 5
1 186
0 9
1 105
0 1
1 173
0 4
1 160
0 5
1 173
0 6
1 167
0 1
1 127
0 5
1 108
0 4
1 161
0 3
2 1 30
0 7
1 153
0 9
1 113
0 4
1 135
0 1
1 137
0 2
1 101
0 5
1 193
0 1
1 122
0 2
```



진행을 하며 :

과제를 진행할 때, 이진파일로부터 정보를 받아 구조체에 저장하는 것 자체는 크게 어려움이 없었습니다.
리턴값을 이용하여 받는 파일의 끝이 되면 그만 읽게 하는 것도 교수님께서 수업에서 힌트와 예시코드를 주셔서 상대적으로 편하게 작업하였습니다.

오히려 저는 알고리즘적 요소보다는, 프로세스라는 주어진 구조체와는 달리 특정 타입을 과제에서 지정해주지 않은 튜플을 어떻게 읽어야할지 고민을 많이 하였습니다.
이후 나오는 코드들은 위 코드에서 for문(26라인 ~37라인) 부분을 바꾼 코드들입니다.
또한 페이지수를 위하여 결과 값은 위쪽 일부만 기재하였습니다.

처음에는 튜플의 요소들이 1바이트니까 튜플의 각 요소를 1바이트인 char형으로 선언하고 받아 보았습니다. →

```
for( j = 0 ; j < i ; j++ )
{
    char movement;
    char time;
    if(fread(&movement, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d ", movement);

    if(fread(&time, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d\n", time);
}
```

그러자 튜플 결과가 음수로 나오며, 생각대로 결과가 나오지 않았습니다.↓

```
ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
0 4
1 -84
0 6
1 -124
0 9
1 -107
0 7
1 -75
0 1
1 -108
1 0 46
0 2
1 120
0 5
1 126
0 5
1 -70
```

저는 char형이 문자형이라 그런가 싶어 이번엔 정수형인 int형으로 정보를 받아보았습니다. ↓

```
for( j = 0 ; j < i ; j++ )
{
    int movement;
    int time;
    if(fread(&movement, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d ", movement);

    if(fread(&time, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d\n", time);
}
```

비록 int형이 1바이트가 아니라도 같은 정수형이므로 제대로 된 결과가 나올 줄 알았으나, 결과는 예상외였습니다. ↓

```
ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
21760 4
21761 172
21760 6
21761 132
21760 9
21761 149
21760 7
21761 181
21760 1
21761 148
1 0 46
21760 2
```

(참고자료)

C언어 기본자료형의 크기와 및 범위

자료형	키워드	메모리 크기
문자형	**char**	1 Bytes
정수형	short	2 Bytes
	int	4 Bytes
	long	4 Bytes

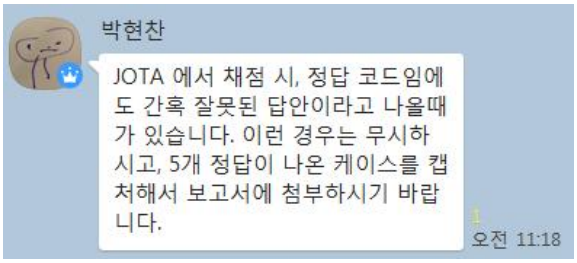
전 이때에는 왜 int형에서 이런 결과가 나오는지 알지 못했습니다. 결국 받는 정보는 1바이트인 것에 비해 저장하는 메모리의 사용크기 차이 때문에 이런 결과가 나왔다고 생각을 하여, 가장 1바이트와 차이가 적은 정수형인 short형으로 다시 실행을 해보았습니다. →

```
ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
0 4
1 172
0 6
1 132
0 9
1 149
0 7
1 181
0 1
1 148
1 0 46
0 2
1 120
```

```
for( j = 0 ; j < i ; j++ )
{
    short movement;
    short time;
    if(fread(&movement, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d ", movement);

    if(fread(&time, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d\n", time);
}
```

← 결과는 보기엔 잘 나오는 거 같아 보였습니다. 심지어 JOTA에 제출했을 때도 통과가 되었습니다.

근데 뭔가 이상했습니다. 동일 코드로 반복하여 JOTA에 제출을 해보았는데 어떤 경우는 일부분 오답이 뜨는 것이였습니다. 물론 교수님께서 카톡방에서 JOTA 오류가 있다고는 하셨지만, 4바이트인 int 가 바이트 차이로 결과가 다르게 나온 거라면 아무리 차이가 적더라도 1바이트가 아닌 short도 다른 결과가 있어야하지 않을까 하는 생각을 하였습니다.



내가 놓친 부분이 도대체 무엇일까 고민을 해보았습니다. 혹시 뭔가 자료저장방식의 차이 때문에 그런걸까? 하는 마음에 2의보수법도 잠시 다시 복습해보고 코드를 unsigned int와 unsigned short로 바꾸어서도 해보았는데, 결과는 딱히 큰 차이는 없었습니다.

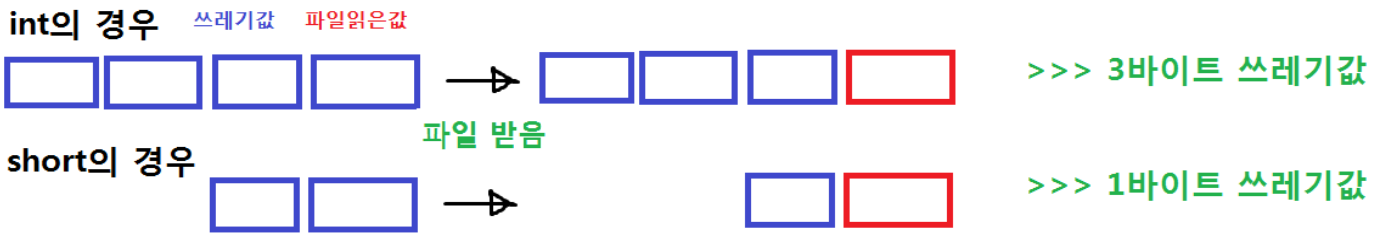
```
20      for( j = 0 ; j < i ; j++ )
21      {
22          unsigned int movement;
23          unsigned int time;
24          if(fread(&movement, 1 , 1, stdin) == 0 )
25          {
26              fprintf(stdout, "들어온 프로세스의 정보가 올바르지 않습니다.\n");
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
22016 4
22017 172
22016 6
22017 132
22016 9
22017 149
22016 7
22017 181
22016 1
```

```
20      for( j = 0 ; j < i ; j++ )
21      {
22          unsigned short movement;
23          unsigned short time;
24          if(fread(&movement, 1 , 1, std
25          {
26              fprintf(stdout, "들어온 프
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
0 4
1 172
0 6
1 132
0 9
1 149
0 7
1 181
0 1
1 148
```

곰곰이 생각하던 중 문뜩 제가 변수를 초기화하지 않고 정보를 읽었다는 걸 깨달았습니다.
int movement = 0; int time = 0; 으로 초기화하고 실행해보니 처음 제가 생각했던 결과가 나왔습니다!
계속 이상한 결과 값이 나왔던 이유는 int의 4바이트 중 1바이트만 파일 정보를 저장하고, 나머지 정보를 읽지 않은 3바이트에는 쓰레기 값이 있었기 때문이었습니다. short가 int에 비해 상대적으로 결과 값이 잘 나온 것처럼 보인 이유는 int의 값의 경우 3바이트가 쓰레기 값이고, short는 1바이트만 쓰레기 값 이여서 단순히 확률적으로 예상치 못한 값이 덜 나왔을 뿐 이였습니다.



아 그럼 char도 초기화가 안 되어서 값이 이상했나? 하는 생각이 들어 char변수도 초기화를 해보았습니다.

```c
for( j = 0 ; j < i ; j++ ) // 튜플 갯수만큼 반복.
{
    char movement = 0; // 동작
    char time = 0;     // 길이(동작 수행 시간)

    if(fread(&movement, 1 , 1, stdin) == 0 )    // 1바이트 읽기 : 동작 저장, 예외처리
    {
        fprintf(stdout, "읽을 항목이 없습니다.\n");
        return 0;
    }
    if(fread(&time, 1 , 1, stdin) == 0 )        // 1바이트 읽기 : 길이 저장, 예외처리
    {
        fprintf(stdout, "읽을 항목이 없습니다.\n");
        return 0;
    }
```

```
ubuntu@201619460:~/hw0$ gcc -o os0 os0.c
ubuntu@201619460:~/hw0$ cat test.bin | ./os0
0 0 20
0 4
1 -84
0 6
1 -124
0 9
1 -107
0 7
```

그러나 char는 0으로 초기화 하여도 딱히 변동이 없었습니다. 다시 생각해보니 char는 당연히 같은 1바이트이므로 쓰레기 값 때문에 그런 것이 아니라 자료저장방식차이 때문일 거라는 생각을 하게 되었고, 이에 char로 선언한 것을 unsigned char 로 선언해보았더니 원하는 결과가 잘 나왔습니다.

그래서 최종적으로 타입은 unsigned char을 쓰게 되었습니다. ↓

```c
for( j = 0 ; j < i ; j++ ) // 튜플 갯수만큼 반복.
{
    unsigned char movement = 0; // 동작
    unsigned char time = 0;     // 길이(동작 수행 시간)

    if(fread(&movement, 1 , 1, stdin) == 0 )    // 1바이트 읽기 : 동작 저장, 예외처리
    {
        fprintf(stdout, "읽을 항목이 없습니다.\n");
        return 0;
    }
    if(fread(&time, 1 , 1, stdin) == 0 )        // 1바이트 읽기 : 길이 저장, 예외처리
    {
        fprintf(stdout, "읽을 항목이 없습니다.\n");
        return 0;
    }

    fprintf(stdout, "%d %d\n", movement, time); // 튜플 출력
}
```

이 밖에
코드를 제작하며 더 간단하게 만들 수 없을까 생각을 하게 되어 변수를 한 개로 통일해 보기도 하였으나 ↓

```c
for( j = 0 ; j < i ; j++ )
{
    short temporarily;
    if(fread(&temporarily, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "파일 읽기에 실패하였습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d ", temporarily);

    if(fread(&temporarily, 1 , 1, stdin) == 0 )
    {
        fprintf(stdout, "파일 읽기에 실패하였습니다.\n");
        return 0;
    }
    fprintf(stdout, "%d\n", temporarily);
}
```

JOTA에서 문제로 주어진 조건(코드는 “%d %d\n” 으로 출력)에 맞추는 게 좋겠다 생각하여 변수는 두 개로하기로 하였습니다. ↓

stdin 으로부터 Binary 형태의 프로세스 정보와 코드를 읽어들여 아래와 같이 순서대로 출력하시오.

출력 형식

프로세스 정보 `"%d %d %d\n"`

코드 `"%d %d\n"`

그리고 변수는 위의 **코드:** 부분에서 설명할 때 잠깐 언급 하였듯 변수 두 개를 쓰는 거 보단 객체로 관리하는 게 더 좋겠다는 생각을 하여 튜플 구조체를 정의함으로써 최종적으로 지금의 코드가 나오게 되었습니다.