소프트웨어실습2

과제1

2014313303, 홍태하

파일이 unix인지 dos인지 판별 :

변수 fd 에 input파일을 오픈합니다. 1칸짜리 buf배열을 선언하여 1byte씩 buf에 읽습니다. \r뒤에 바로 \n이 나오는지 판별하기 위해 buf에 \r이 나올 때 flag를 1로 바꾸고 flag가 1일 때 다음 buf에 \n이 들어오면 dos로 판별하는 count인 변수 countD를 하나 올리고 flag가 0일 때 다음 buf에 \n이 들어오면 unix로 판별하는 count인 변수 countU를 하나 올립니다. 그렇게 한 후 countU가 countD보다 크면 input을 유닉스로 판별하고 countD가 countU보다 크거나 같으면 input을 도스로 판별하여 변환합니다. (인풋을 끝까지 읽었기 때문에 다시 앞에서 읽을 수 있도록 lseek을 사용하여 포인터를 맨 앞으로 돌려줍니다.)

Unix를 dos로 바꾸기(U2D함수) :

Input을 open한 fd와 output으로 생성할 파일을 오픈한 fd2를 parameter로 받습니다. Input을 u2d\_buf배열에 하나씩 읽습니다. \r이 나오면 쓰지 않고 r\_count를 하나 올려준 뒤 continue를 해줍니다. 그리고 r\_count가 0이 아니면 r\_count의 개수만큼 \r을 써줍니다. 이것은 unix와 dos가 섞여서 나오는데 unix로 판별하여서 unix를 dos로 바꿀 때 \r\n이 있는 경우 \r\r\n이 되지 않고 그대로 \r\n이 될 수 있도록 하기 위함입니다. 그리고 \n이 나왔는데 r\_count가 0이면 \r을 미리 하나 써줌으로써 \n을 \r\n으로 변환합니다. 또 \t이 나오면 space 4개를 써주고 continue함으로써 \t을 space4개로 변환합니다. while문의 끝에는 r\_count를 0으로 초기화해주고 버퍼에 들어온 값을 써줍니다. size변수를 만들어서 write를 실행할 때마다 앞에서 하나씩 증가시켜줌으로써 output파일의 size를 계산합니다. 마지막으로 fd2를 close해주고 size값을 리턴합니다.

원래는 이 과정으로 U2D함수를 구현하였지만 추가로 trimming을 수행하기 위해 먼저 write를 하지 않고 저 과정 그대로 size만 count합니다. Lseek를 이용하여 다시 맨 처음부터 읽을 수 있도록 합니다. Size를 구한 뒤 이 size크기만큼의 배열을 선언합니다. 그리고 다시 저 과정대로 배열에 내용을 옮겨 담습니다. 그 후에 dos파일을 trimming하는 U2D\_trimming함수를 호출합니다. 이 함수는 trimming을 진행하는데 trimming이 진행되지 않고 변환이 된 파일의 size를 parameter로 받아서 그 size만큼 for문을 돌립니다. 그러다가 배열에 \t나 space가 있으면 카운트를 하고 continue를 해줍니다. 그리고 그 내용들은 size만큼의 크기를 가진 temp라는 배열에 앞에서부터 저장됩니다. 그 후에 \r\n이 나오면 그냥 그대로 진행시켜서 \t과 space가 사라지도록 하고 만약에 \r\n이 나타나지 않는다면 temp에 저장되어있던 내용들을 write해줍니다. 그리고 write할 때마다 r\_size를 count하여 trimming한 후의 size를 계산하여 마지막에 size로 출력해줍니다.

Dos를 unix로 바꾸기(D2U함수) :

Input의 길이를 계산하기 위해 input을 parameter로 받고 fd와 fd2도 parameter로 받습니다. Input을 d2u\_buf배열에 하나씩 읽습니다. 이 경우에도 \r이 나오면 r\_count를 하나 증가시켜주고 continue해줍니다. 그리고 r\_count가 1일 때 다음 버프에 \n값이 아니면 다시 \r을 써줌으로써 \r\n을 \n으로 바꿉니다. 원 파일의 확장자가 .c, .h, .java 인 경우에만 space8개를 \t로 바꿔야 하므로 input의 길이를 변수 len에 저장해서 input의 끝에서부터 저 확장자가 맞는지 비교합니다. 확장자가 맞을 경우 space가 나오면 s\_count를 하나 증가시키고 continue시킵니다. Space가 8개 연속으로 있는지 판단하기 위해서 s\_count를 증가시키면서 판단하다가 8개가 된 순간 \t를 써줍니다. Space가 연속으로 8\*n개를 넘어갈 때마다 변수 n을 하나 증가시킵니다. 만약 space가 연속으로 8의 배수개가 있지 않을 경우 space가 8개 이하인 경우에는 s\_count만큼 써주고 8개 이상인 경우 8개가 될 때마다 \t으로 변환되었을 것이기 때문에 그 만큼을 빼고 써줍니다. 이 작업을 한 후 n은 1로 초기화 해주고 s\_count는 0으로 초기화 해줍니다. 끝으로 r\_count도 0으로 초기화 해주고 버퍼에 있는 값을 써줍니다. D2U도 U2D와 마찬가지로 write할 때마다 size를 증가시켜 output파일의 size를 계산하고 리턴해줍니다. 리턴하기 전에는 fd2를 close해줍니다.

원래는 이 과정으로 D2U함수를 구현하였지만 추가로 trimming을 수행하기 위해 먼저 write를 하지 않고 저 과정 그대로 size만 count합니다. Lseek를 이용하여 다시 맨 처음부터 읽을 수 있도록 합니다. Size를 구한 뒤 이 size크기만큼의 배열을 선언합니다. 그리고 다시 저 과정대로 배열에 내용을 옮겨 담습니다. 그 후에 unix파일을 trimming하는 D2U\_trimming함수를 호출합니다. 이 함수는 trimming하기 전의 size를 받아서 그 만큼 for문을 돌리고 \t이나 space가 나오면 각각 카운트 해주고 temp배열에 저장해주고 continue해줍니다. 그리고 그 후에 \n이 나온다면 그대로 \t와 space가 사라지는 효과가 나타나게 되고 만약에 \n이 나오지 않는다면 temp에 저장되어 있던 것들을 다시 write해줍니다. 이런 식으로 trimming을 진행하고 trimming한 후의 size를 반환하여 마지막에 출력해줍니다.

output파일 만들 때 input파일 뒤에 .out붙이기 :

output파일을 만들기 위해서 일단 배열의 길이를 계산하는 strlength함수를 만듭니다. 배열의 끝에는 \0이 저장되에 있는 것을 이용해서 \0이 나올 때까지 하나씩 세면서 배열의 길이를 계산합니다. output파일은 malloc으로 input크기에 4를 더한만큼을 지정합니다. name이라는 배열에 저장했는데 name의 앞부터 input을 넣고 뒤 4자리에 .out을 써줍니다.

처리 결과 메시지 출력하기 :

파일이 정상적으로 처리 된 경우 input파일과 output파일과 size를 출력합니다. Printf를 쓰면 간단히 출력할 수 있지만 printf를 쓸 수 없으므로 배열을 잡아 write해줍니다. input\_file이라는 배열을 잡아서 앞쪽부터 Input:이라는 String을 넣고 그 뒤에 input파일 이름을 넣은 후 space까지 하나 넣어서 저장합니다. output\_file이라는 배열을 잡아서 앞쪽부터 Output:이라는 String을 넣고 그 뒤에 output파일 이름을 넣은 후 space까지 하나 넣어서 저장합니다. Size를 write하기 위해서는 숫자를 스트링으로 바꿔야 합니다. 숫자를 스트링으로 바꾸기 위해서 size를 0보다 클 때까지 10으로 나누고 count하면서 size가 몇자리 수 인지 알아냅니다. 그 후 size라는 배열을 잡아서 앞에서부터 Size:라는 스트링을 넣고 맨 뒤에는 \n을 넣고 그 뒤부터 size를 10으로 나눈 나머지를 넣고 size를 10으로 나누고 다시 10으로 나눈 나머지를 넣으면서 저장합니다. 그 후 이 3개의 배열을 write해주면 파일이 정상적으로 처리 된 경우 처리 결과 메시지를 출력할 수 있습니다.

에러 처리하기, 결과 반환 :

에러가 발생한 경우 stderr에 “Error occurred!\n”라는 문자열을 출력해야 합니다. 먼저 4가지 에러를 처리해야 하는데 “입력 파일이 존재하지 않음(E\_INPUT\_NOT\_EXIST)”, “입력 파일을 이용할 권한이 없음(E\_INPUT\_NOT\_ALLOWED)”, “같은 이름을 가진 파일이 이미 존재함(E\_OUTPUT\_EXIST)”, “해당 위치에 파일을 생성할 권한이 없음(E\_OUTPUT\_NOT\_ALLOWED)” 이렇게 4가지가 있다. Errno.h헤더를 include해서 errno에 들어오는 값을 보고 판단한다. 그러나 errno는 가장 마지막의 에러만을 값으로 가지므로 위치를 잘 정해야 한다. errno이 2일 때 입력파일이 존재하지 않는 오류이므로 1을 반환하고 “Error occurred!\n”을 stderr에 출력한다. 이것은 input파일을 열었을 때 판단할 수 있으므로 input파일을 연 바로 뒤에 위치하게 한다. errno이 13일 때 입력 파일을 이용할 권한이 없는 오류이므로 2를 반환하고 “Error occurred!\n”을 stderr에 출력한다. 이것도 input파일을 열었을 때 판단할 수 있으므로 input파일을 연 바로 뒤에 위치하게 한다. errno이 17일 때 같은 이름을 가진 파일이 이미 존재해있는 오류이므로 3을 반환하고 “Error occurred!\n”을 stderr에 출력한다. 이것은 output파일을 열었을 때 판단할 수 있으므로 output파일을 연 바로 뒤에 위치하게 한다. errno이 13일 때 같은 해당 위치에 파일을 생성할 권한이 없는 오류이므로 4를 반환하고 “Error occurred!\n”을 stderr에 출력한다. 이것은 output파일을 열었을 때 판단할 수 있으므로 output파일을 연 바로 뒤에 위치하게 한다. 파일이 정상적으로 처리된 경우에는 0을 반환한다.