**과목명: 시스템프로그래밍**

**2 분반**

**<<Project #3>>**

**서강대학교 [컴퓨터공학과]**

**[20161566]**

**[권형준]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. 흐름도 설명
3. **(새로 추가된) 모듈 정의**
   1. Void Progaddr()
   2. Void Loader()
   3. Int Loader\_Pass1()
   4. Int Loader\_Pass2()
   5. Int Add\_ESTAB(char tlable[], int tloc, int 시두홋)
   6. Void BP
   7. Int BP\_Match(int target)
   8. Void Run()

1. **(새로 추가된) 전역 변수 정의**
   1. Int Loader\_Run\_start\_address
   2. Int running
   3. Int R[REGISTER\_NUM]
   4. Int breakpoint[MAX\_LOCCTR\_NUM]
   5. Int breakpoint\_num
   6. Struct external\_node
   7. External\_node\* estab\_head
2. **코드 설명**
3. **프로그램 개요**

앞서서 프로젝트2에서 구현한 assemble기능에 linking과 loading 기능을 추가하는 프로그램이다. Assemble로 생성된 obj파일들을 linking하고 loading하여 프로젝트 1에서 구현한 메모리에 올리며, run이라는 명령어를 통해서 직접 수행시킨다.

1. **프로그램 설명**
   1. **프로그램 흐름도(새로 추가된 부분)**

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

* Loader()

텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

Return 1

* Run()

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

* 1. **흐름도 설명**

원래의 프로그램에서 새로 추가된 부분만 설명하겠다. 우선 loader는 2개의 pass로 이루어져 있다. Pass1에서는 Definition들의 Label과 주소를 저장한다. 이가 성공적으로 돌아가면 -1이외의 값, 즉 load된 프로그램의 길이를 반환한다. Pass2에서는 pass 1에서 만들어진 ESTABLE을 가지고 실행된다. 만약 ESTABLE에 Reference하는 LABEL이 있다면 그 주소를 사용하여 modification과정을 거치게 된다. 성공하게 되면 1을 반환하게 된다. 이번 프로젝트에서 loading에서의 오류는 고려하지 않아서 따로 오류는 흐름도에 구현하지 않았지만, 프로그램 상 만약 Label의 중복이 감지되면 loading이 실패되고, reference 하는 Label이 존재하지 않는 경우에도 실패한다. 다음 Run은 조금 더 복잡한 과정을 거친다. PC가 프로그램의 길이와 같아질 때까지 반복을 하는데, PC위치의 1byte를 읽어온다. 그 1byte로 opcode를 판별하고 format과 nixbpe에 따라서 새로운 PC나 instruction을 수행한다. 각 단계별로 레지스터가 업데이트 될 수 있고, BP를 만나면 멈춘다.

1. **모듈 정의**
   1. Void Progaddr()

변수

🡪

함수 설명

🡪Loader\_Run\_Start\_address라는 변수에 입력 받은 주소를 대입한다. Add\_History를 호출하여 history에 추가한다. Loader\_Run\_Start\_address는 loader, run 명령어에서 사용된다.

* 1. Void Loader()

변수

🡪external\_node\* temp: ESTABLE을 초기화하기 위한 변수

함수 설명

🡪이전에 loading, 또는 loading 실패하여 남아있는 estable이 있을 수 있으니 estab\_head부터 free를 하여 estable을 초기화한다. 다음 새로운 estable을 만들고 loader\_pass1과 loader\_pass2를 실행한다. 성공적으로 실행이 완료되면, load map 즉 estable을 하나씩 호출하여 출력한다. Loader가 끝난 이후에 estable을 삭제한다. Laoding이 완료되면 모든 register들은 0으로 초기화되고 PC와 L만 프로그램의 시작주소와 프로그램의 길이로 초기화된다.

* 1. Int Loader\_Pass1()

변수

🡪int CSLTH: 현재 control section의 길이

🡪int CSADDR: 현재 control section의 시작 주소

🡪int LENGTH: 전체 프로그램의 길이를 저장하기 위한 변수

🡪int cycle: 총 obj파일의 수

🡪char token, int token\_num, size: 한 줄을 token으로 나누기 위한 변수

🡪int record\_style: head인지 definition인지 분별하기 위한 변수

함수 설명

🡪인자로 들어온 obj파일을 하나씩 돌면서 한 줄씩 살펴본다. 한 줄씩 받아서 token으로 줄을 나누고 처음 token을 살펴서 H와 D를 구분하고 나머지는 넘기고 다음 줄을 읽는다. H인 경우 control section의 이름과 시작 주소를 ESTABLE에 추가하고 그 control section의 길이를 저장한다. 만약 D로 들어왔을 경우 LABEL과 함께 그 주소를 해당 control section시작 주소와 더해서 ESTABLE에 저장한다. 해당 control section의 시작에 control section의 길이를 더해서 다음 control section의 시작 주소를 지정한다.

* 1. Int Loader\_Pass2()

변수

🡪int CSLTH: 현재 control section의 길이

🡪int CSADDR: 현재 control section의 시작 주소

🡪int cycle: 총 obj파일의 수

🡪int External\_Reference[20]: reference number의 값을 저장하는 변수

함수 설명

🡪loader에서 호출을 받는다. 각 obj파일별로 돌면서 각 줄을 확인한다. 이 부분에서는 E, R, M, T을 수행한다. E를 받기 전까지 계속 수행을 하며, E뒤에 저장된 값을 프로그램의 시작 주소로 저장한다. 하지만 이번 프로젝트는 copy.obj만 run을 수행한다고 하였기 때문에 E 뒤의 주소를 저장하는 것을 따로 하지 않았다. 어차피 run의 시작 주소와 처음 시작 instruction의 위치가 같기 때문이다. H를 만나면 control section의 길이를 저장한다. 이는 다음 control section의 시작 주소를 정하기 위해 읽어야 된다. T를 읽게 되면 해당하는 주소에 각 메모리를 1byte단위로 올린다. M을 만나면 해당하는 memory의 값을 받아와서 reference\_number에 해당하는 LABEL의 주소를 ESTAB에서 읽어와서 더하거나 뺀다. 그리고 그 결과를 다시 memory에 넣는다. 이를 모든 obj파일과 동일하게 수행한다.

* 1. Int Add\_ESTAB(char tlable[], int tloc, int tlength)

변수

🡪

함수 설명

🡪Label, 주소, 길이를 받아서 ESTABLE에 저장한다. 만약 저장하려고 하는 것이 control section이면 주소와 길이가 둘 다 양수를 받으며, 저장하려고 하는 것이 단순 label이라면 loc은 양수이지만 length는 음수이다. 처음부터 돌면서 만약 같은 이름을 가진 원소가 존재한다면 그 주소를 반환한다. 이 함수는 처음 ESTABLE을 추가할 때 사용할 수 있고 loader\_pass2에서 해당하는 label의 주소를 찾을 때 사용할 수 있다.

* 1. Void BP

변수

🡪

함수 설명

🡪인자가 없다면 존재하는 bp를 입력 받은 순서대로 출력한다. 만약 뒤의 인자가 clear이면 모든 bp를 초기화한다. 만약 뒤의 인자가 주소라면(모든 입력 주소는 유효하다는 프로젝트의 가정 때문에) 바로 마지막 breakpoint배열에 추가한다.

* 1. Int BP\_Match(int target)

변수

🡪

함수 설명

🡪breakpoint라는 배열을 처음부터 돌면서 해당 주소가 breakpoint라면 1을 아니라면 0을 반환한다.

* 1. Void Run()

변수

🡪int PC: 현재 돌고 있는 PC의 값을 저장한다. R[6]도 PC를 뜻하지만 프로그램 내에서 가독성을 위해서 따로 변수를 사용하였다.

함수 설명

🡪이미 수행 중인 run이 없다면(breakpoint 때문에 run이 중단된 상황이면) 레지스터를 초기화하고 PC와 L의 값을 프로그램이 돌아갈 수 있게 수정한다. 만약 L에 저장된 값이 0이라면, 즉 loading된 것이 없다면 오류를 출력하고 종료한다. PC가 program length가 아니라면 반복문을 돌면서 1byte를 읽어서 opcode를 해석하고 format 2인경우에는 뒤에 register의 값들을 읽어서 수행하고 다음 1byte를 읽어서 다음 opcode를 해석한다. 만약 format 3또는 4이면 nixbpe의 값을 1bit씩 읽으면서 정한다. 그에 따라서 target address를 구하고 instruction에 따라서 memory와 상호 작용하여 instruction을 수행한다. 한 개의 instruction을 수행한 뒤 PC가 가리키는 지점이 breakpoint에 해당한다면(BP\_Match를 호출한다) B[6]에 PC의 값을 저장하고 레지스터들을 출력하고 run을 종료한다. 프로그램이 끝난 이후에는 똑같이 레지스터들을 출력하고 bp를 초기화한다.

1. **전역 변수 정의**
   1. Int Loader\_Run\_start\_address

🡪loader와 run의 수행 시작 주소를 저장한다.

* 1. Int running

🡪현재 run이 수행되고 있는지 아닌지 저장한다. 예를 들어서 run을 돌리던 도중 breakpoint를 만나서 프로그램이 중단된 상황이라면 running이 1이 된다.

* 1. Int R[REGISTER\_NUM]

🡪레지스터의 현재 값을 저장한다. 0부터 순서대로 a, x, l, b, s, t, pc, CC(status word), program length이다. CC는 compare의 결과를 Program length는 가장 최근에 load된 프로그램의 전체 길이를 저장한다.

* 1. Int breakpoint[MAX\_LOCCTR\_NUM]

🡪1차원 배열 형태로 breakpoint를 저장한다. Run을 수행하지 않고는 breakpoint의 수행 순서를 알 수 없다. 따라서 어떠한 형태로 저장하여도 탐색 속도는 크게 차이가 없어서 1차원 배열로 구현하였다.

* 1. Int breakpoint\_num

🡪breakpoint의 개수를 출력한다.

* 1. Struct external\_node

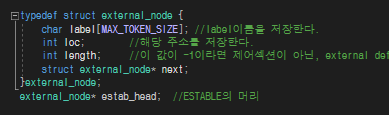
🡪ESTABLE을 구현하기 위한 node이다. Label을 저장할 공간, 주소, control section이라면 길이를 저장할 공간을 구현했다.

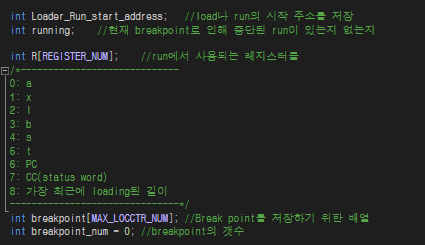
* 1. External\_node\* estab\_head

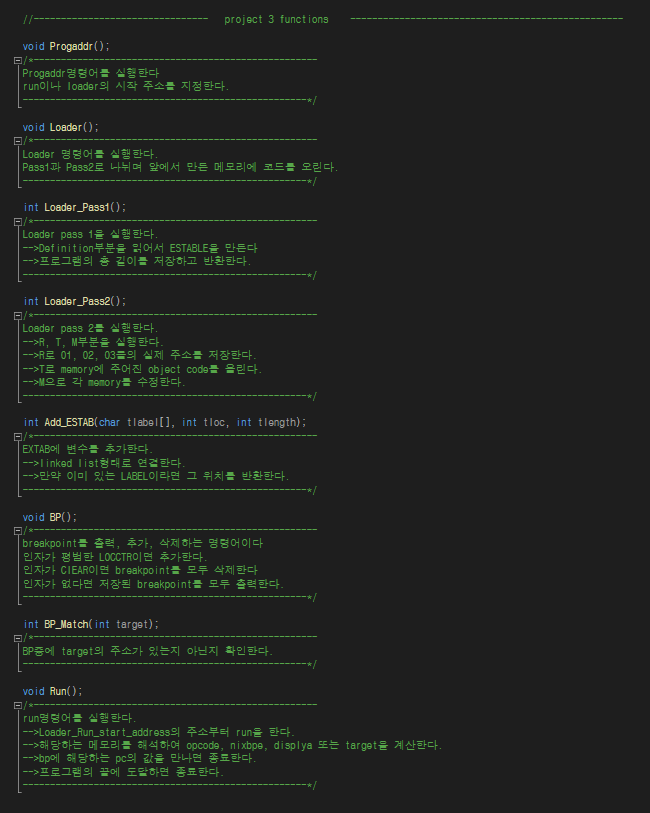
🡪ESTABLe의 머리에 해당한다.

1. **코드 설명(새로 추가된 부분)**

**<20161566.h>**







**<20161566.c>**

