**과목명: 시스템프로그래밍**

**1분반**

**<<Project #3>>**

**서강대학교 커뮤니케이션학과**

**20171101**

**박근영**

목 차

1. **프로그램 개요**
2. **프로그램 설명**

2.1 알고리즘 설계

2.2 자료구조

2.3 프로그램 흐름도

1. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
2. **전역 변수 정의**
3. **코드 설명**

**1. 프로그램 개요**

프로젝트 2에서 구현한 shell위에 linking loader기능, 프로그램 실행(run) 기능을 추가한 프로그램이다. Linking loader는 프로젝트2에서 assemble명령을 통해 만든 object file들을 읽어 링킹 작업을 수행하고, 코드를 가상 메모리에 올리게 된다. run 명령어는 메모리에 올라간 프로그램을 실행한다. 프로그램 실행 시 디버깅을 위한 breakpoint를 설정할 수 있다.

따라서 구현해야 할 명령어는 다음과 같다.

① 주소 지정 명령어 ( progaddr )

② Linking Loader ( loader )

③ 프로그램 실행 ( run )

④ debug 명령어 ( bp )

**2. 프로그램 설명**

**2.1 알고리즘 설계**

**① Linking Loader**

Linking Loader는 두 개의 pass를 통해 object file들을 받아 각각 control section들을 linking하여 메모리에 load하는 역할을 한다. 첫 번째 pass에서는 링킹 과정에 사용되는 external symbol들을 ESTAB에 추가하고, 두 번째 pass에서는 실제 linking과 loading과정을 수행하게 된다. 각 pass가 해야 하는 일을 자세히 보면 다음과 같다.

**Pass 1:**

- progaddr부터 첫 번째 control section의 시작주소(CSADDR)을 지정하고, 각 object 파일을 읽어 들인다.

- H record를 읽어 해당 control section의 정보를 ESTAB에 추가한다.

- D record에서 정의된 external symbol을 ESTAB에 추가한다.

- E record를 읽으면, CSADDR에 CSLTH를 더해 다음 control section 주소를 얻는다.

**Pass 2:** (csaddr 지정방식은 pass 1과 동일)

**-** R record를 읽어 해당 control section에서 참조한 external symbol의 reference number를 지정한다. (modification record에서 symbol name을 대체)

**-** T record에 있는 object code를 메모리 주소에 올린다.

**-** M record를 읽으면, symbol을 ESTAB에서 찾아 메모리의 relocation 작업을 수행한다.

- E record에서는 instruction의 시작주소를 얻을 수 있다.

**① Run**

Linking Loader를 통해 메모리에 올라간 프로그램을 읽어 실행하는 명령어이다. 프로그램의 실행 시작 address는 progaddr이며, 해당 프로그램이 종료되거나 breakpoint를 만난다면 실행을 멈춘다. 이후 다시 run을 하면 PC, 레지스터 값들이 저장되어 있어 중단된 부분부터 다시 실행되도록 하였다.

구현 방법은 먼저 PC를 기준으로 메모리에서 object code형태로 된 instruction을 읽는다. 그 다음 instruction을 포맷에 맞게 해석하고, opcode에 맞는 명령을 수행해 레지스터 값이나 메모리 값을 변화시킨다. 그리고 다음 PC로 넘어가 다시 위의 절차를 수행한다. breakpoint는 PC가 변화할 때마다 체크한다.

자세한 알고리즘은 아래 첨부한 프로그램 흐름도를 통해 볼 수 있다.

**2.2 자료구조**

**① ESTAB**

linking시에 external symbol과 control section에 대한 정보를 저장하는 external symbol record는 Hashtable 형태의 자료구조를 통해 구현하였다. 해시함수는 프로젝트 1에서 OPTAB을 구현할 때 만든 함수를 그대로 사용하였다.

typedef struct ex\_node\* estab\_pointer;

typedef struct ex\_node{

char name[10];

int address;

int length;

char type;

estab\_pointer link;

}ex\_node;

estab\_pointer ESTAB[ESTAB\_SIZE];

name, address, length는 각각 external symbol의 이름과, 시작 메모리주소, 길이(control section에만 적용)을 가리킨다. 그리고 type은 해당 노드가 control section을 가리키는지, symbol을 가리키는지 정의한다. control section은 type=’c’이고 symbol은 type=’s’이다.

**② Breakpoint**

run함수 구동 시 디버깅을 위해 설정하는 breakpoint들은 동적 배열을 통해 구현하였다. 배열은 해당 breakpoint의 위치를 저장한다.

int\* break\_point;

int bp\_size; //breakpoint의 총 개수

**2.1 프로그램 흐름도**

**① Linking Loader – pass 1**

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

**② Linking Loader – pass 2**

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

**③ Run**

**텍스트, 지도이(가) 표시된 사진

자동 생성된 설명**

**3. 모듈 정의**

**3.1 정의한 모듈에 대한 간략적인 설명**

프로젝트 3에서 새로 추가한 모듈에 대한 설명이다.

**int LinkingLoader(char args[][ARG\_MAX\_LEN], int file\_num);**

: 입력 받은 오브젝트 파일을 링킹하여 메모리에 로드하는 과정을 수행합니다.

|  |  |
| --- | --- |
| input | (char) args[][ARG\_MAX\_LEN] (파일명 목록) |
| return | (int) 1: success, -1: failure |

**void InitLinkingLoader()**

: Linking Loader function에 필요한 변수와 자료형들을 초기화합니다. (ESTAB)

|  |  |
| --- | --- |
| input | none |
| return | none |

**estab\_pointer FindExternal(char\* s)**

: 입력받은 symbol을 가리키는 pointer를 ESTAB에서 찾아 반환합니다. symbol이 없을 경우 NULL을 반환합니다.

|  |  |
| --- | --- |
| input | (char\*) symbol name |
| return | (estab\_pointer)해당 external symbol을 가리키는 pointer (NULL:failure) |

**void InsertExternal(char\* s, int addr, int len, char type)**

: ESTAB에 해당 symbol/control section의 이름과 address, length를 추가합니다.

|  |  |
| --- | --- |
| input | (char\*) name |
|  | (int) address |
|  | (int) length (control section일 경우에만 유효) |
|  | (char) type (‘c’ -> control section, ‘s’ -> symbol) |
| return | none |

**void PrintExternal()**

ESTAB에 속한 symbol과 control section들을 형식에 맞게 출력합니다.

|  |  |
| --- | --- |
| input | none |
| return | none |

**int Run()**

: 로딩을 통해 메모리에 올라간 프로그램을 실행합니다.

|  |  |
| --- | --- |
| input | none |
| return | (int) 1: success, -1: failure |

**int AddBreakPoint(int bp)**

: 입력받은 address에 breakpoint를 설정합니다.

|  |  |
| --- | --- |
| input | (int) bp |
| return | (int) 1: success, -1: failure(중복) |

**void ClearBreakPoint()**

: 설정된 breakpoint를 모두 삭제합니다.

|  |  |
| --- | --- |
| input | none |
| return | none |

**void PrintBreakPoint()**

: 설정된 breakpoint를 모두 화면에 출력합니다.

|  |  |
| --- | --- |
| input | none |
| return | none |

**void PrintRegisters()**

: 레지스터들의 값을 모두 화면에 출력합니다.

|  |  |
| --- | --- |
| input | none |
| return | none |

**hash\_pointer FindOpcodebyNum(int opcode)**

: opcode를 통해 OPTAB에서 해당 opcode의 정보를 담고 있는 hash\_pointer를 리턴합니다.

|  |  |
| --- | --- |
| input | (int) opcode |
| return | (hash\_pointer) 해당 opcode를 가리킴 |

**4 전역 변수 정의**

|  |  |
| --- | --- |
| **int prog\_total\_len** | 프로그램의 총 길이 |
| **int end\_addr** | 프로그램이 종료되는 지점의 주소 |
| **int PC** | program counter(PC)가 가리키는 값 |
| **int CC** | 비교하는 operation의 결과를 가리키는 Condition Code  ‘<’는 -1, ‘=’는 0, ‘>’는 1으로 표현한다 |
| **int progaddr** | 프로그램이 로드되는 시작 주소 |
| **int csaddr** | 각 control section이 시작되는 주소 |
| **int execaddr** | 프로그램이 실행되는 시작 주소 |
| **int regi\_arr[6]** | register의 값을 저장 (A,X,L,B,S,T) |
| **int\* break\_point** | breakpoint를 저장하는 배열 포인터 |
| **int bp\_size** | 설정된 breakpoint 개수 |

**5 코드 설명**

**void InsertExternal(char\* s, int addr, int len, char type)**

|  |
| --- |
| int hashcode = HashFunction(s, ESTAB\_SIZE); **//find hashcode**  **//insert into ESTAB**  estab\_pointer tmp = (estab\_pointer)malloc(sizeof(ex\_node));  strcpy(tmp->name, s);  tmp->address = addr;  tmp->length = len;  tmp->type = type;  tmp->link = ESTAB[hashcode];  ESTAB[hashcode] = tmp; |

**estab\_pointer FindExternal(char\* s)**

|  |
| --- |
| int flag = 0; **//탐색 결과를 가리키는 flag**  int hashcode = HashFunction(s, ESTAB\_SIZE);  estab\_pointer ptr = ESTAB[hashcode];  **//search ESTAB**  for(; ptr; ptr=ptr->link){  if(!strcmp(s, ptr->name)){  flag = 1;  break;  }  }  if(flag == 0) **//symbol을 찾지 못함**  return NULL;  else **//해당 symbol을 가리키는 pointer 리턴**  return ptr; |

**void PrintExternal()**

|  |
| --- |
| prog\_total\_len = 0; **//프로그램 총 길이**  estab\_pointer ptr;  printf("control symbol address length\n");  printf("section name\n");  printf("-----------------------------\n");  for(int i=0; i<ESTAB\_SIZE; i++){  ptr = ESTAB[i];  for(; ptr; ptr=ptr->link){  **//control section**  if(ptr->type=='c'){  printf("%s\t\t%04X %04X\n", ptr->name, ptr->address, ptr->length);  prog\_total\_len += ptr->length;  }  **//symbol**  else if(ptr->type=='s'){  printf("\t%s\t%04X\n", ptr->name, ptr->address);  }  }  }  printf("-----------------------------\n");  printf("\t total length %04X\n", prog\_total\_len); |

**void InitLinkingLoader()**

|  |
| --- |
| **//initialize external symbol table**  estab\_pointer next, cur;  for(int i=0; i<ESTAB\_SIZE; i++){  cur = ESTAB[i];  while(cur!=NULL){  next = cur->link;  free(cur);  cur = next;  }  ESTAB[i] = NULL; **//hashtable을 모두 NULL로 초기화**  } |

**int AddBreakPoint(int bp)**

|  |
| --- |
| **//중복체크**  for(int i=0;i<bp\_size;i++){  if(break\_point[i] == bp) return -1; **//이미 breakpoint가 설정되었을 경우**  }  break\_point = (int\*)realloc(break\_point, sizeof(int)\*(bp\_size+1));  break\_point[bp\_size++] = bp;  printf("\t[ok] create breakpoint %04X\n", bp);  return 1; |

**void ClearBreakPoint()**

|  |
| --- |
| **//break\_point 배열, bp\_size 변수 초기화**  bp\_size = 0;  free(break\_point);  break\_point = (int\*)malloc(sizeof(int));  printf("\t[ok] clear all breakpoints\n"); |

**void PrintBreakPoint()**

|  |
| --- |
| **//설정된 breakpoint 목록 출력**  printf("\tbreakpoint\n");  printf("\t----------\n");  for(int i=0;i<bp\_size;i++){  printf("\t%X\n", break\_point[i]);  } |

**void PrintRegisters()**

|  |
| --- |
| **//레지스터 값 출력**  printf("A : %06X X : %06X\n", regi\_arr[0], regi\_arr[1]);  printf("L : %06X PC : %06X\n", regi\_arr[2], PC);  printf("B : %06X S : %06X\n", regi\_arr[3], regi\_arr[4]);  printf("T : %06X\n", regi\_arr[5]); |

**hash\_pointer FindOpcodebyNum(int opcode)**

|  |
| --- |
| hash\_pointer ptr;  **//hashtable 전체 탐색**  for(int i=0;i<HASH\_TABLE\_SIZE;i++){  ptr = hashTable[i];  for(; ptr; ptr=ptr->link){  if(opcode == ptr->opcode)  return ptr;  }  }  return NULL; **//opcode가 없을 경우** |

**int LinkingLoader(char args[][ARG\_MAX\_LEN], int file\_num)**

int cslth; **//control section length**

int addr, text\_len, value, refernum;

int mdf\_len, refaddr;

int byte, line;

char buf1[80], buf2[20], tmp[20]; **//입력 버퍼**

char\* token;

char referArr[10][10]; **//reference symbol array**

char op;

estab\_pointer symbol;

**InitLinkingLoader();** **//initialization**

**//pass1**

csaddr = progaddr; **//control section address 지정**

for(int csnum=0;csnum<file\_num;csnum++){ **//입력받은 object file 다 읽을 때까지 반복**

FILE\* fp = fopen(args[csnum], "r");

if(!fp){ **//file open error**

printf("file open error\n");

return -1;

}

line = 1; **//object file line**

**//read header record**

fscanf(fp, "%s", buf1);

fscanf(fp, "%s", buf2);

**//search estab for control section name**

**if(FindExternal(buf1+1) != NULL)**{ **//external symbol exist**

**//에러메시지 출력 후 종료**

printf("loader 1 error: duplicate external symbol\n");

printf("%s(line: %d)[%s]\n", args[csnum], line, buf1+1);

printf("\*%s\n", buf1);

return -1;

}

cslth = strtol(buf2+6, NULL, 16); **//control section length**

**InsertExternal(buf1+1, csaddr, cslth, 'c');** **//insert control section into ESTAB**

fscanf(fp, "%s", buf1); **//read next input record**

**while(buf1[0]!='E')**{ **//until END record**

line++;

**//external symbol definition**

**if(buf1[0]=='D')**{

strcpy(buf1, buf1+1);

while(1){ **//D record line 입력받아 처리**

fscanf(fp, "%s", buf2);

strncpy(tmp, buf2, 6); **//address를 가리키는 문자열**

tmp[6] = '\0';

addr = csaddr + strtol(tmp, NULL, 16); **//external symbol address**

**//search estab for symbol**

**if(FindExternal(buf1) != NULL)**{

**//이미 symbol이 ESTAB에 존재할 경우, 에러메시지 출력 후 종료**

printf("loader 1 error: duplicate exernal symbol\n");

printf("%s(line: %d)[%s]\n", args[csnum], line, buf1);

return -1;

}

**//insert symbol into ESTAB**

**InsertExternal(buf1, addr, 0, 's');**

if(buf2[6]=='\0') break;

strcpy(buf1, buf2+6);

}

}

fscanf(fp, "%s", buf1); **//read next input record**

}

fclose(fp);

csaddr += cslth; **//start address for next control section**

}

**//pass2**

char c;

csaddr = progaddr; **//control section address 지정**

execaddr = progaddr; **//프로그램 실행 시작주소**

for(int csnum=0; csnum<file\_num; csnum++){

FILE\* fp = fopen(args[csnum], "r");

line = 1;

**//read header record**

fscanf(fp, "%s", buf1);

fscanf(fp, "%s\n", buf2);

strcpy(referArr[1], buf1+1); **//control section에 reference number 1 배정**

cslth = strtol(buf2+6, NULL, 16);

c = fgetc(fp); **//line의 첫글자 read**

**while(c!='E')**{

line++;

**//external symbol reference**

**if(c=='R')**{

fgets(buf1, 80, fp);

buf1[strlen(buf1)-1] = '\0';

token = strtok(buf1, " ");

**//referArr에 reference number에 맞는 symbol 저장**

while(token != NULL){

strncpy(tmp, token, 2);

tmp[2] = '\0';

refernum = strtol(tmp, NULL, 16);

strcpy(referArr[refernum], token+2);

token = strtok(NULL, " ");

}

}

**//text record**

**else if(c=='T')**{

fgets(buf1, 80, fp);

**//get start address**

strncpy(tmp, buf1, 6);

tmp[6] = '\0';

addr = csaddr + strtol(tmp, NULL, 16);

**//get length**

strncpy(tmp, buf1+6, 2);

tmp[2] = '\0';

text\_len = strtol(tmp, NULL, 16);

**//move to memory location**

strcpy(buf1, buf1+8);

for(int loc=0; loc<text\_len; loc++){

strncpy(tmp, buf1+loc\*2, 2);

tmp[2] = '\0';

value = strtol(tmp, NULL, 16);

**Edit(addr+loc, value); //load**

}

}

**//modification record**

**else if(c=='M'){**

fgets(buf1, 80, fp);

sscanf(buf1, "%6X%2X%1c%2X", &addr, &mdf\_len, &op, &refernum);

addr += csaddr; **//modify할 시작 address**

**//get external symbol address**

**symbol = FindExternal(referArr[refernum]);**

**//symbol이 ESTAB에 존재하지 않을 경우 에러메시지 출력 후 종료**

if(symbol == NULL){

printf("loader 2 error: undefined external symbol\n");

printf("%s(line: %d)[%s]\n", args[csnum], line, referArr[refernum]);

return -1;

}

refaddr = symbol->address;

byte = (op=='+'? refaddr : -refaddr); **//address에서 modify되는 값**

if(mdf\_len==6){ **//address length to be modified = 6**

byte = byte+(memory[addr]<<16)+(memory[addr+1]<<8)+memory[addr+2]; **//get actual address**

**//overflow, underflow가 일어났을 경우 값 수정**

if(byte>0xFFFFFF) byte -= 0x1000000;

if(byte<0x000000) byte += 0x1000000;

**//로딩된 메모리주소값 수정**

**Edit(addr+2, byte&0x0000FF);**

byte = byte>>8;

**Edit(addr+1, byte&0x00FF);**

byte = byte>>8;

**Edit(addr, byte);**

}

else if(mdf\_len==5){ **//length 5**

byte = byte+((memory[addr]&0x0F)<<16)+(memory[addr+1]<<8)+memory[addr+2]; **//get actual address**

**//overflow, underflow가 일어났을 경우 값 수정**

if(byte>0xFFFFF) byte -= 0x100000;

if(byte<0x00000) byte += 0x100000;

**//로딩된 메모리주소값 수정**

**Edit(addr+2, byte&0x0000FF);**

byte = byte >> 8;

**Edit(addr+1, byte&0x00FF);**

byte = (byte >> 8) + (memory[addr]&0xF0);

**Edit(addr, byte);**

}

}

else{ **//other input**

fgets(buf1, 80, fp);

}

c = fgetc(fp);

}

fclose(fp);

csaddr += cslth; **//start address of next control section**

}

**PrintExternal();**

**//initialize for run**

memset(regi\_arr, 0, sizeof(regi\_arr));

regi\_arr[2] = prog\_total\_len; **//L register 프로그램 길이로 초기화**

PC = progaddr;

end\_addr = progaddr + prog\_total\_len;

return 1;

**int Run()**

int target\_addr, target\_value, disp, opcode, e, nextPC;

int regi1, regi2;

hash\_pointer op\_ptr;

while(PC < end\_addr){

**//find opcode**

opcode = memory[PC] >> 2 << 2;

op\_ptr = **FindOpcodebyNum(opcode);**

if(op\_ptr == NULL){ **//opcode를 찾을 수 없으면 에러처리하고 종료**

printf("error: unknown opcode [PC:%X]\n", PC);

return -1;

}

**//format**

if(op\_ptr->format[0] == '1'){ **//format 1**

nextPC = PC + 1;

}

else if(op\_ptr->format[0] == '2'){ **//format 2**

nextPC = PC + 2;

regi1 = memory[PC+1]>>4;

regi2 = memory[PC+1]&0x0F;

}

else{ **//format 3 or 4**

**e** = (memory[PC+1]>>4) & 1;

if(e == 0){ **//format 3**

nextPC = PC + 3;

**//displacement를 통해 target address 구하기**

disp = ((memory[PC+1]&0x0F)<<8) + memory[PC+2];

if(((memory[PC+1]>>5)&1) == 1){ **//PC relative**

if(disp>2047) disp -= 0xFFF;

target\_addr = nextPC + disp;

}

else if(((memory[PC+1]>>6)&1) == 1){ **//base relative**

target\_addr = regi\_arr[3] + disp;

}

else{ **//direct addressing**

target\_addr = disp;

}

}

else{ **//format 4**

nextPC = PC + 4;

target\_addr = ((memory[PC+1]&0x0F)<<16) + (memory[PC+2]<<8) + (memory[PC+3]);

}

if((memory[PC]&3) == 2){ **//indirect**

target\_addr = (memory[target\_addr]<<16) + (memory[target\_addr+1]<<8) + (memory[target\_addr+2]);

}

if((memory[PC+1]>>7) == 1){ **//indexed X**

target\_addr += regi\_arr[1];

}

**//target value 구하기**

if((memory[PC]&3) == 1){ **//immediate addressing**

target\_value = target\_addr;

}

else{ **//이외 일반적인 경우**

target\_value = (memory[target\_addr]<<16) + (memory[target\_addr+1]<<8) + memory[target\_addr+2];

}

}

**//opcode에 맞는 명령 수행**

switch(opcode){

case 0x00: **//LDA**

regi\_arr[0] = target\_value;

break;

case 0x68: **//LDB**

regi\_arr[3] = target\_value;

break;

case 0x74: **//LDT**

regi\_arr[5] = target\_value;

break;

case 0x50: **//LDCH**

regi\_arr[0] = regi\_arr[0]>>8<<8;

regi\_arr[0] = target\_value>>16;

break;

case 0x0C: **//STA**

memory[target\_addr] = regi\_arr[0]>>16;

memory[target\_addr+1] = regi\_arr[0]>>8;

memory[target\_addr+2] = regi\_arr[0];

break;

case 0x14: **//STL**

memory[target\_addr] = regi\_arr[2]>>16;

memory[target\_addr+1] = regi\_arr[2]>>8;

memory[target\_addr+2] = regi\_arr[2];

break;

case 0x10: **//STX**

memory[target\_addr] = regi\_arr[1]>>16;

memory[target\_addr+1] = regi\_arr[1]>>8;

memory[target\_addr+2] = regi\_arr[1];

break;

case 0x54: **//STCH**

memory[target\_addr] = regi\_arr[0]&0x0000FF;

break;

case 0x48: **//JSUB**

regi\_arr[2] = nextPC; **//store return address in L**

nextPC = target\_addr;

break;

case 0x4C: **//RSUB**

nextPC = regi\_arr[2];

break;

case 0x3C: **//J**

nextPC = target\_addr;

break;

case 0x30: **//JEQ**

if(CC==0) nextPC = target\_addr;

break;

case 0x38: **//JLT**

if(CC<0) nextPC = target\_addr;

break;

case 0x28: **//COMP**

**//CC는**

CC = regi\_arr[0] - target\_value;

break;

case 0xA0: **//COMPR**

CC = regi\_arr[regi1]-regi\_arr[regi2];

break;

case 0xB8: **//TIXR**

CC = ++regi\_arr[1] - regi\_arr[regi1];

break;

case 0xB4: **//CLEAR**

regi\_arr[regi1] = 0;

break;

case 0xE0: **//TD**

CC = -1; **//CC를 ‘<’으로 변경**

break;

case 0xD8: **//RD**

break;

case 0xDC: **//WD**

break;

}

PC = nextPC; **//PC 증가**

**//breakpoint check**

for(int i=0; i<bp\_size; i++){

if(break\_point[i] == PC){

**PrintRegisters();**

printf("\tStop at checkpoint[%X]\n", PC);

return 1;

}

}

}

**PrintRegisters();**

printf("\tEnd Program\n");

return 1;