callstack_2025350038_박준우.pdf (보고서)

목차

초반부 헤더파일, 전역변수, 함수 선언 등

base.c

```c
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int      call_stack[STACK_SIZE];          // Call Stack을 저장하는 배열
char     stack_info[STACK_SIZE][20];      // Call Stack 요소에 대한 설명을 저장하는 배열
```

callstack_2025350038_박준우.c

```c
#include <stdio.h>
#define STACK_SIZE 50 // 최대 스택 크기

int      call_stack[STACK_SIZE];          // Call Stack을 저장하는 배열
char*    stack_info[STACK_SIZE];      // Call Stack 요소에 대한 설명을 저장하는 배열
```

편의상 stack_info를 char 형 STACK_SIZE by 20 크기의 2차원 배열에서 char* 형 STACK_SIZE 크기의 1차원 배열로 바꿨다.

사용 예시:

```c
stack_info[SP] = "Return Address";
```

기존의 형태(char 형 2차원 배열)를 그대로 유지하고 위와 같은 기능을 하는 코드를 짜려면 strcpy 함수(string.h 헤더파일에 있는 문자열 복사 함수)를 사용해야 하는 등 번거로움이 있어서 형태를 위와 같이(char* 형 1차원 배열) 바꾸었다.

---

base.c / callstack_2025350038_박준우.c

```c
int SP = -1;
int FP = -1;

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);
```

(변경사항 없음)

push

base.c

```

```

callstack_2025350038_박준우.c

```
void push(int value, char* info)
{
    SP++;
    call_stack[SP] = value;
    stack_info[SP] = info;
}
```

스택의 원리를 그대로 구현한 코드로, SP를 1만큼 더하여 저장 공간을 한 칸 만들어준 다음에 그 곳에 값을 저장한다.

사용 예시:

```
push(num, "num");
```

push_at

base.c

```
```

callstack_2025350038_박준우.c

```
void push_at(int idx_from_sp, int value, char* info)
// 함수 프롤로그에서 지역변수를 저장할 때에만 사용한다.
{
    int index = SP + idx_from_sp;
    call_stack[index] = value;
    stack_info[index] = info;
}
```

함수 프롤로그에서 지역변수를 저장할 때에만 사용한다. call stack에 어떤 함수의 지역변수를 저장할 때, 컴파일 과정에서 지역변수의 수를 미리 계산한 다음 call stack에 미리 그 만큼의 공간을 할당한다. 이를 C언어 코드로 구현하면 다음과 같다.

```
SP += LOCAL_VAR_COUNT;
```

이 상태에서 지역변수의 값을 저장해야 하는데, 이를 push 함수로 구현하기는 어렵다고 판단하여 push_at 함수를 추가로 구현하였다.
물론 다음과 같이 push 함수로도 구현을 할 수는 있다.

```
SP += LOCAL_VAR_COUNT; // 미리 계산된 만큼 공간 할당

SP -= LOCAL_VAR_COUNT; // SP를 원래 위치로 돌려놓기 (?)
int i;
for (i = 0; i <= LOCAL_VAR_COUNT; i++)
{
    push(local_vars[i], "description");
}
```

그러나 이 경우 컴파일러가 지역변수의 수를 미리 계산한 것이 의미가 없어진다.

사용 예시:

```
push_at(2, num, "num"); // SP가 가리키는 곳보다 두 칸 앞에, 즉 call_stack[SP+2]에 저장한다.
```

```
// 지역변수 저장
SP += LOCAL_VAR_COUNT;
int i;
for (i = 0; i <= LOCAL_VAR_COUNT; i++)
{
    push_at(-i, local_vars[i], "description");
}
```

pop

base.c

```

```

callstack_2025350038_박준우.c

```
int pop()
{
    SP--;
    return call_stack[SP + 1];
}
```

스택 최상단의 값을 반환하고 SP에서 1만큼 뺀다. 스택 최상단은 이제 사용하지는 않지만, 그 값이 지워지진 않는다.

print_stack

이 단락의 모든 코드는 지금부터 print_stack 함수의 내부이다.
참고: print_stack 선언부

```
void print_stack();
```

base.c

```
if (SP == -1)
{
    printf("Stack is empty.\n");
    return;
}

printf("====== Current Call Stack ======\n");

for (int i = SP; i >= 0; i--)
{
    if (call_stack[i] != -1)
        printf("%d : %s = %d", i ,stack_info[i], call_stack[i]);
    else
        printf("%d : %s", i, stack_info[i]);

    if (i == SP)
        printf("     <=== [esp]\n");
    else if (i == FP)
        printf("     <=== [ebp]\n");
    else
        printf("\n");
}
printf("================================\n\n");
```

callstack_2025350038_박준우.c

```
if (SP == -1)
{
    // 참고: main 함수의 stack frame을 구현했으므로 call stack이 비는 경우는 없음.
    printf("Stack is empty.\n");
    return;
}

printf("====== Current Call Stack ======\n");

for (int i = SP; i >= 0; i--)
```

```
{
    if (call_stack[i] != -1)
        printf("%d : %s = %d", i, stack_info[i], call_stack[i]);
    else
        printf("%d : %s", i, stack_info[i]);

    if (i == SP)
        printf("      <=== [esp]\n");
    else if (i == FP)
        printf("      <=== [ebp]\n");
    else
        printf("\n");
}
printf("================================\n\n");
```

(주석 제외 변경사항 없음)

main 함수의 stack frame을 구현한 이유는 후술한다.

stack_info 형식

종류	stack_info에 저장될 설명
매개변수	Param ‘{변수명}’ of Func ‘{함수명}’
지역변수	Local Var ‘{변수명}’ of Func ‘{함수명}’
반환주소값	Return Address
SFP	SFP of Func ‘{함수명}’

함수 호출 구조

main -> func1 -> func2 -> func3

func1

이 단락의 모든 코드는 지금부터 func1 함수의 내부이다.
참고: func1 선언부

```
void func1(int arg1, int arg2, int arg3);
```

base.c

```
int var_1 = 100;

// func1의 스택 프레임 형성 (함수 프롤로그 + push)
print_stack();
```

callstack_2025350038_박준우.c

```
    int var_1 = 100;

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
#define FUNC1_LOCAL_VAR_COUNT 1 // 컴파일 과정에서 계산될 지역변수의 수
    push(arg1, "Param \'arg1\' of Func \'func1\'");
    push(arg2, "Param \'arg2\' of Func \'func1\'");
    push(arg3, "Param \'arg3\' of Func \'func1\'");

    push(-1, "Return Address");
    push(FP, "SFP of Func \'main\'");
    FP = SP;
    SP += FUNC1_LOCAL_VAR_COUNT;

    push_at(0, var_1, "Local Var \'var_1\' of Func \'func1\'");

    print_stack();
    /*************************************/
```

func1의 지역변수의 수는 컴파일 과정에서 계산되지만, 여기선 메크로 상수로 대체하였다. (Line 4)

함수 프롤로그 과정:
매개변수 (Line 5~7) -> 반환주소값 (Line 9) -> 현재 FP (Line 10) -> FP를 SP 위치에 갱신 (Line 11) -> 지역변수를 저장하도록 SP 갱신 (Line 12) -> 지역변수 저장 (Line 14)
Line 16의 print_stack(); 실행 결과:

```
====== Current Call Stack ======
6 : Local Var 'var_1' of Func 'func1' = 100     <=== [esp]
```

```
5 : SFP of Func 'main' = 0     <=== [ebp]
4 : Return Address
3 : Param 'arg3' of Func 'func1' = 3
2 : Param 'arg2' of Func 'func1' = 2
1 : Param 'arg1' of Func 'func1' = 1
0 : Stack Frame of Func 'main'
================================
```

---

base.c

```
func2(11, 13);
// func2의 스택 프레임 제거 (함수 에필로그 + pop)
print_stack();
```

callstack_2025350038_박준우.c

```
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
#define FUNC2_PARAM_COUNT 2
    SP = FP;
    FP = pop();
    SP--; // Return Address
    SP -= FUNC2_PARAM_COUNT;
    print_stack();
    /************************************/
```

함수 에필로그 과정:

SP를 FP 위치로 갱신 (Line 4) -> FP 복원 (Line 5) -> 주소값 복원 (호출한 함수로 되돌아감. 여기선 Return Address를 pop하는 것으로 대체함.) (Line 6) -> 매개변수 정리 (Line 7)

Line 8의 print_stack(); 실행 결과:

```
====== Current Call Stack ======
6 : Local Var 'var_1' of Func 'func1' = 100     <=== [esp]
5 : SFP of Func 'main' = 0     <=== [ebp]
4 : Return Address
3 : Param 'arg3' of Func 'func1' = 3
2 : Param 'arg2' of Func 'func1' = 2
1 : Param 'arg1' of Func 'func1' = 1
0 : Stack Frame of Func 'main'
================================
```

func2

이 단락의 모든 코드는 지금부터 func2 함수의 내부이다.
참고: func2 선언부

```
void func2(int arg1, int arg2);
```

base.c

```
int var_2 = 200;

// func2의 스택 프레임 형성 (함수 프롤로그 + push)
print_stack();
```

callstack_2025350038_박준우.c

```
    int var_2 = 200;

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
#define FUNC2_LOCAL_VAR_COUNT 1
    push(arg1, "Param \'arg1\' of Func \'func2\'");
    push(arg2, "Param \'arg2\' of Func \'func2\'");

    push(-1, "Return Address");
    push(FP, "SFP of Func \'func1\'");
    FP = SP;
    SP += FUNC2_LOCAL_VAR_COUNT;

    push_at(0, var_2, "Local Var \'var_2\' of Func \'func2\'");

    print_stack();
    /************************************/
```

함수 프롤로그 과정:
매개변수 (Line 5~6) -> 반환주소값 (Line 8) -> 현재 FP (Line 9) -> FP를 SP 위치에 갱신 (Line 10) -> 지역변수를 저장하도록 SP 갱신 (Line 11) -> 지역변수 저장 (Line 13)
Line 15의 print_stack(); 실행 결과:

```
====== Current Call Stack ======
11 : Local Var 'var_2' of Func 'func2' = 200     <=== [esp]
10 : SFP of Func 'func1' = 5     <=== [ebp]
9 : Return Address
8 : Param 'arg2' of Func 'func2' = 13
```

```
7 : Param 'arg1' of Func 'func2' = 11
6 : Local Var 'var_1' of Func 'func1' = 100
5 : SFP of Func 'main' = 0
4 : Return Address
3 : Param 'arg3' of Func 'func1' = 3
2 : Param 'arg2' of Func 'func1' = 2
1 : Param 'arg1' of Func 'func1' = 1
0 : Stack Frame of Func 'main'
================================
```

---

base.c

```
func3(77);
// func3의 스택 프레임 제거 (함수 에필로그 + pop)
print_stack();
```

callstack_2025350038_박준우.c

```
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
#define FUNC3_PARAM_COUNT 1
    SP = FP;
    FP = pop();
    SP--; // Return Address
    SP -= FUNC3_PARAM_COUNT;
    print_stack();
    /************************************/
```

함수 에필로그 과정:

SP를 FP 위치로 갱신 (Line 4) -> FP 복원 (Line 5) -> 주소값 복원 (호출한 함수로 되돌아감. 여기선 Return Address를 pop하는 것으로 대체함.) (Line 6) -> 매개변수 정리 (Line 7)

Line 8의 print_stack(); 실행 결과:

```
====== Current Call Stack ======
11 : Local Var 'var_2' of Func 'func2' = 200     <=== [esp]
10 : SFP of Func 'func1' = 5     <=== [ebp]
9 : Return Address
8 : Param 'arg2' of Func 'func2' = 13
7 : Param 'arg1' of Func 'func2' = 11
6 : Local Var 'var_1' of Func 'func1' = 100
5 : SFP of Func 'main' = 0
```

```
4 : Return Address
3 : Param 'arg3' of Func 'func1' = 3
2 : Param 'arg2' of Func 'func1' = 2
1 : Param 'arg1' of Func 'func1' = 1
0 : Stack Frame of Func 'main'
================================
```

func3

이 단락의 모든 코드는 지금부터 func3 함수의 내부이다.
참고: func3 선언부

```
void func3(int arg1);
```

base.c

```
int var_3 = 300;
int var_4 = 400;

// func3의 스택 프레임 형성 (함수 프롤로그 + push)
print_stack();
```

callstack_2025350038_박준우.c

```
    int var_3 = 300;
    int var_4 = 400;

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
#define FUNC3_LOCAL_VAR_COUNT 2
    push(arg1, "Param \'arg1\' of Func \'func3\'");

    push(-1, "Return Address");
    push(FP, "SFP of Func \'func2\'");
    FP = SP;
    SP += FUNC3_LOCAL_VAR_COUNT;

    push_at(0, var_3, "Local Var \'var_3\' of Func \'func3\'");
    push_at(-1, var_4, "Local Var \'var_4\' of Func \'func3\'");

    print_stack();
    /*************************************/
```

Line 15의 print_stack(); 실행 결과:

```
====== Current Call Stack ======
16 : Local Var 'var_3' of Func 'func3' = 300     <=== [esp]
15 : Local Var 'var_4' of Func 'func3' = 400
14 : SFP of Func 'func2' = 10     <=== [ebp]
13 : Return Address
12 : Param 'arg1' of Func 'func3' = 77
11 : Local Var 'var_2' of Func 'func2' = 200
10 : SFP of Func 'func1' = 5
9 : Return Address
```

```
8 : Param 'arg2' of Func 'func2' = 13
7 : Param 'arg1' of Func 'func2' = 11
6 : Local Var 'var_1' of Func 'func1' = 100
5 : SFP of Func 'main' = 0
4 : Return Address
3 : Param 'arg3' of Func 'func1' = 3
2 : Param 'arg2' of Func 'func1' = 2
1 : Param 'arg1' of Func 'func1' = 1
0 : Stack Frame of Func 'main'
================================
```

main

이 단락의 모든 코드는 지금부터 main 함수의 내부이다.

base.c

```
```

callstack_2025350038_박준우.c

```
// FP == -1(기본값)일 때 출력이 안 되는 오류가 있어서 main 함수의 stack frame을 구현함.
push(-1, "Stack Frame of Func \'main\'");
FP = 0;
```

main 함수를 실행중일 때, FP의 값은 기본값인 -1이다. 그러나 이 때 다른 함수를 호출하여 SFP를 저장할 때 -1이라는 값이 저장되고, print_stack 함수는 이를 하나의 값이 아닌 일종의 메타데이터로 취급하기 때문에 출력하지 않는 오류가 있다.
예상되는 출력 결과:

```
...
5 : SFP of Func 'main' = -1
...
```

실제 출력 결과(오류):

```
...
5 : SFP of Func 'main'
...
```

따라서 main 함수의 stack frame을 구현하고 이를 인덱스 0 위치에 고정시킴으로써 FP의 기본값을 -1이 아닌 0으로 하여 이 문제를 해결하였다.

---

base.c

```
func1(1, 2, 3);
// func1의 스택 프레임 제거 (함수 에필로그 + pop)
print_stack();
return 0;
```

callstack_2025350038_박준우.c

```
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
#define FUNC1_PARAM_COUNT 3
    SP = FP;
    SP--; // Return Address
```

```
    SP -= FUNC1_PARAM_COUNT;
    print_stack();
    /************************************/

    return 0;
```

실행 결과:

```
====== Current Call Stack ======
0 : Stack Frame of Func 'main'     <=== [esp]
================================
```

참고: esp와 ebp가 같은 경우 콘솔 창에는 esp만 표시된다.