CyKor Week1

작성자: 사이버국방학과 2024350213 김진모

1. 대략적인 구현 과정

대략적인 구현 과정에 대해 먼저 설명하려고 한다.

-Push 함수와 Pop 함수

우선 call_stack과 stack_info 각각 두 스택에 대해서 push와 pop 동작을 서로 다른 함수로 구현을 하려고 했다. 그러나 잘 생각을 해 보면 push와 pop을 항상 일괄적으로 진행하기 때문에, 하나의 push와 pop함수로 구현했다.

-함수 프롤로그

만든 push함수와 pop함수를 사용하여 func1, func2, func3 내부에 각 함수들의 함수 프롤로그와 에필로그를 구현했다. 함수 프롤로그는 매개변수를 push(오른쪽부터) → Return Address를 push → 현재 FP를 SFP로 사용하기 위해 push → FP값에 현재 SP값을 저장 → 지역변수를 push(위쪽부터)의 과정으로 구현했다. 이때 지역변수는 실제로는 push되는 것이 아니라, 지역 변수들의 크기에 맞게 SP가 새로 설정된 이후에 지역변수가 저장되게 되므로 이를 반영하기 위해 SP값을 먼저 새로 설정하고, 지역 변수들을 빈 공간에 저장하는 방식으로 수정한 후 두 번째 commit을 진행했다. 그리고 이 글을 쓰는 지금, SP값을 먼저 새로 설정하는 이유가 Stack이 넘쳐서 Segmentation Fault를 발생하지 않게 하기 위함이기 때문에 SP 값을 증가시켰을 때에 이 값이 STACK_SIZE와 같거나 보다 크면 그 즉시 Segmentation Fault임을 출력하고 exit하도록 수정한 후 세 번째 commit을 진행했다.

-함수 에필로그

함수 에필로그는 SP값에 현재 FP값을 저장 → SFP값을 참조하여 FP에 이전 FP값을 복원 → Return Address를 pop(실제로는 해당 주소값으로 돌아간다.) → 나머지 매개변수를 pop의 과정으로 구현했다. C에서의 Calling convention에 따라 caller 함수인 main, func1, func2 함수가 매개변수를 정리하도록 했다.

---

2. Push 함수와 Pop 함수

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

void push(int val, char *desc);
int pop();

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

우선 강의 영상에서는, push, pop 함수와 함수 프롤로그, 함수 에필로그를 각각 함수로 구현하는 것과, 이들 모두를 func1, func2, func3 안에서 구현하는 것 모두 허용된다고 했기 때문에 push와 pop에 대해서만 따로 함수로 구현을 해 주었다.

```
void push(int val, char *desc)
{
    SP += 1;
    call_stack[SP] = val;
    strcpy(stack_info[SP], desc);
}
```

push함수는 call_stack에 push할 값인 변수 val과 변수 val에 대한 설명, 즉 stack_info에 push할 문자열인 desc를 매개변수로 받는다. 이때 call_stack의 높이가 1 증가하므로 SP 값을 먼저 1 늘리고, call_stack[SP]에 val값을 저장해준다. 이후 stack_info[SP]에 이에 대한 설명인 문자열 desc를 copy해준다.

```
int pop()
{
    int a = call_stack[SP];
    call_stack[SP] = 0;
    strcpy(stack_info[SP], "");
    SP -= 1;
    return a;
}
```

pop함수는 따로 매개변수를 받지 않는다. 이후에 pop()했을 때 call_stack에서 나온 변수를 통해 이전 FP의 값을 복원해야 하므로 int형 변수를 리턴하도록 함수를 짰다. call_stack[SP]에 저장된 값을 변수 a에 저장해주었다. 이후 call_stack[SP]의 값을 0으로

초기화하고, stack_info[SP]의 값 역시 공백 문자열을 copy하여 초기화했다. 변수가 하나 나오기 때문에 SP의 값을 1 감소시키고, 이후 pop한 값인 a를 리턴해준다.

---

3. 함수 프롤로그

최종적으로 세 번의 commit을 통해 함수 프롤로그를 수정한 결과로는

매개변수를 push(오른쪽부터) → Return Address를 push → 현재 FP를 SFP로 사용하기 위해 push → FP값에 현재 SP값을 저장 → 지역 변수들의 전체 크기에 맞게 SP값을 새로 설정 → stack의 빈 공간에 지역변수들을 저장(위쪽부터)의 과정으로 구현했다.

```
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg3, "arg3");
    push(arg2, "arg2");
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func1 SFP");
    FP = SP;
    //push(var_1, "var_1");
    SP += 1;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP] = var_1;
    strcpy(stack_info[SP], "var_1");
```

func1은 우선 매개변수가 오른쪽부터 arg3, arg2, arg1의 순서대로 있으므로 stack에 해당 순서대로 push된다. 이후 Return Address를 -1의 값으로 push한다. 현재 FP값을 SFP로 사용해야 하므로 FP값을 main 함수의 frame pointer, 즉 func1의 함수 프롤로그에 저장되는 SFP가 되도록 push한다. 이후 FP값에 SP를 저장하여 갱신하고, 지역 변수들의 크기에 맞게 SP의 크기를 늘린다. func1은 지역 변수가 var_1 한 개이므로 SP의 값을 1 늘렸다. 이때 SP의 값이 STACK_SIZE의 값과 같거나 보다 크면 Segmentation Fault를 출력하면서 프로그램을 종료하도록 했다. 마지막으로 stack의 빈 공간에 지역 변수인 var_1을 push하지 않고 직접 넣었고, stack_info에도 var_1이라는 정보를 직접 넣었다.

```c
void func2(int arg1, int arg2)
{
    int var_2 = 200;

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg2, "arg2");
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func2 SFP");
    FP = SP;
    //push(var_2, "var_2");
    SP += 1;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP] = var_2;
    strcpy(stack_info[SP], "var_2");
```

```c
void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func3 SFP");
    FP = SP;
    // push(var_3, "var_3");
    // push(var_4, "var_4");
    SP += 2;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP - 1] = var_3;
    strcpy(stack_info[SP - 1], "var_3");
    call_stack[SP] = var_4;
    strcpy(stack_info[SP], "var_4");
```

func2와 func3의 함수 프롤로그는 같은 방식으로 구현했다. 매개변수들을 오른쪽부터 call_stack에 push한 후, Return Address를 -1의 값으로 push한다. 이후 FP값을 갱신한 뒤, SP값을 지역 변수들의 크기를 고려하여 증가시키고 Segmentation Fault가 발생하지는 않는 지 검사한다. 마지막으로 call_stack의 빈 공간에 지역 변수들을 위에서부터 넣어주었고, 마찬가지로 설명 역시 copy해 주었다.

---

4. 함수 에필로그

SP값에 현재 FP값을 저장 → SFP값을 참조하여 FP에 이전 FP값을 복원 → Return Address를 pop(실제로는 해당 주소값으로 돌아간다.) → 나머지 매개변수를 pop의 과정으로 구현했다.

```c
print_stack();
func3(77);
// func3의 스택 프레임 제거 (함수 에필로그 + pop)
SP = FP;
FP = pop();
pop();  // Return Address pop
pop();  // 매개변수 77 pop
```

func3의 함수 에필로그 (func2 내부에 구현)

func3이 호출되고 나서 종료되는 과정은 func2 내부에 구현되어 있다. 이때 SP에 FP값을 넣어 SP를 갱신한다. 이후 pop한 값을 FP에 저장해주면 func3의 SFP값, 즉 func2의 frame pointer값이 FP에 저장되므로 func3이 종료된 후 func2로 돌아갈 수 있게 된다. func2의 FP로 돌아왔으므로 Return Address를 통해 원래 코드의 위치로 돌아가면 함수 에필로그가 끝난다. 그러나 이 과제에서는 Return Address의 값을 -1로 저장했으므로, Return Address를 pop하는 것으로 정리한다. 이후 남은 func3의 매개변수를 Calling Convention에 따라 caller함수인 func2 내부에서 pop하는 것으로 정리한다. 이러면 func3 호출 이전의 상태로 call_stack이 복원된다.

```
print_stack();
func2(11, 13);
// func2의 스택 프레임 제거 (함수 에필로그 + pop)
SP = FP;
FP = pop();
pop();  // Return Address pop
pop();  // 매개변수 11 pop
pop();  // 매개변수 13 pop
```

func2의 함수 에필로그 (func1 내부에 구현)

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

func1의 함수 에필로그 (main 내부에 구현)

func2와 func1의 함수 에필로그는 같은 방식으로 구현했다. SP값에 현재 FP값을 저장한 후, 저장된 SFP값을 pop하여 FP에 저장한다. 이후 Return Address를 통해 원래 코드의 주소로 돌아가준다. 과제에서는 pop으로 이 과정을 대체했다. 마지막으로 C에서의 Calling Convention에 따라 caller함수에서 매개변수들을 모두 정리한다.

---

5. 각각의 코드에 대한 구현 이유

push함수와 pop함수는 강의 영상에서 예시로 사용되었던 push함수와 pop함수를 참고하였다. 위에서도 언급했듯이 두 개의 stack이 존재하지만 실제로는 같은 index에 어떤 변수의 값과 해당 변수에 대한 설명을 push하는 것이기에, 하나의 push함수가 두 개의 stack에 작용하도록 구현했다. pop함수 역시 마찬가지로 두 개의 stack에서 모두 pop되고, 값이 저장된 call_stack의 값을 리턴하도록 구현했다.

함수 프롤로그는 강의에서 제시된 과정을 그대로 push함수를 사용하여 구현했다. 이 역시 위에서 언급했듯이, 지역 변수를 저장할 때에 SP가 먼저 지역 변수들의 크기에 맞춰서 증가되고, 이후 지역 변수를 빈 공간에 할당한다는 언급이 있었기 때문에 push로 먼저 구현을 했다가 이에 맞춰서 코드를 수정했다. 이때, SP를 먼저 늘리는 과정의 취지가 Segmentation Fault가 발생하는 지 확인하기 위함이라는 취지를 고려하여 Segmentation Fault를 한 차례 검사하도록 구현했다.

함수 에필로그는 강의에서 제시된 과정을 그대로 구현하되, Return Address의 위치로 코드가 따라가는 과정만 이 값을 pop하는 것으로 대체해 구현했다. Return Address의 주소값에 접근하면 되지만, 이 과제의 틀에서는 Return Address값을 알 수 있는 방법은 있으나 call_stack이 포인터를 저장하지 못하는, int형을 저장하는 배열이기에 -1의 값으로 대체하게 되었다. 따라서 Return Address를 따라가지 못하게 되고, C 코드에서 자연적으로 함수가 종료됨에 따라 원래의 위치로 돌아가기 때문에 Return Address를 pop하여 정리하는 과정을 구현했다.

---

6. 최종 코드

이는 callstack_2024350213_김진모.c 파일로 github에서도 확인할 수 있다.

```
/*  call_stack

   실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 d
   원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제에

   int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
   char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

   ============================call_stack 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
   Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
   반환 주소값을 push할 경우       : -1
   ============================================================



   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우        : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우                : "Return Address"
   ============================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.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 배

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
```

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

void push(int val, char *desc);
int pop();

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

/*
   현재 call_stack 전체를 출력합니다.
   해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
void print_stack()
{
   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");
}

void push(int val, char *desc)
{
    SP += 1;
    call_stack[SP] = val;
    strcpy(stack_info[SP], desc);
}

int pop()
{
    int a = call_stack[SP];
    call_stack[SP] = 0;
    strcpy(stack_info[SP], "");
    SP -= 1;
    return a;
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
     int var_1 = 100;

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg3, "arg3");
    push(arg2, "arg2");
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func1 SFP");
    FP = SP;
    //push(var_1, "var_1");
    SP += 1;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP] = var_1;
    strcpy(stack_info[SP], "var_1");
```

```
    print_stack();
    func2(11, 13);
    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    FP = pop();
    pop();  // Return Address pop
    pop();  // 매개변수 11 pop
    pop();  // 매개변수 13 pop

    print_stack();
}


void func2(int arg1, int arg2)
{
    int var_2 = 200;

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg2, "arg2");
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func2 SFP");
    FP = SP;
    //push(var_2, "var_2");
    SP += 1;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP] = var_2;
    strcpy(stack_info[SP], "var_2");

    print_stack();
    func3(77);
    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    FP = pop();
    pop();  // Return Address pop
    pop();  // 매개변수 77 pop

    print_stack();
```

```
}


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    push(arg1, "arg1");
    push(-1, "Return Address");
    push(FP, "func3 SFP");
    FP = SP;
    // push(var_3, "var_3");
    // push(var_4, "var_4");
    SP += 2;
    if (SP >= STACK_SIZE) {printf("Segmentation Fault"); exit(0);}
    call_stack[SP - 1] = var_3;
    strcpy(stack_info[SP - 1], "var_3");
    call_stack[SP] = var_4;
    strcpy(stack_info[SP], "var_4");

    print_stack();
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main()
{
    func1(1, 2, 3);
    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    SP = FP;
    FP = pop();
    pop();  // Return Address pop
    pop();  // 매개변수 1 pop
    pop();  // 매개변수 2 pop
    pop();  // 매개변수 3 pop
```

```
    print_stack();
    return 0;
}
```