Stack Debugging

Young W. Lim

2017-07-27 Wed

Outline

1. Introduction
   - References
   - Compiling to IA32 Assembly
   - Checking /proc/<pid>/maps file
   - Checking Stack Frames
   - Calling Convention (Procedure Call Standard)

Based on

"Self-service Linux: Mastering the Art of Problem Determination",
Mark Wilding
"Computer Architecture: A Programmer's Perspective",
Bryant & O'Hallaron

preparation : for Linux Mint

- install packages
  - lib32gcc1
  - lib32gcc-dbg
  - gcc-multilib
- gcc -m32 t.c

stack frame address range

- stack frame starts from 0xc0000000
- stack bottom : 0xc0000000
- stack address range
  - 0xbfffe000 (low address)
  - 0xc0000000 (high address)
- stack grows toward lower address
  - from HIGH address (0xc0000000)
  - to LOW address (0xbfffe000)

stack frame in the Linux Mint

- stack address range
  - 0xc0000000 (from HIGH address)
  - 0xbfffe000 (to LOW address)
  - 0x00002000
- stack address range in the Linux Mint
  - 0xfffc9000 (from HIGH address)
  - 0xfffa8000 (to LOW address)
  - 0x00021000
  - these addresses are not fixed

code for displaying /proc/<pid>/maps

- local variables (stack variables) : var
- local variable address : 0xff9e00e8

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int var = 5;
  char cmd[64];

  printf(" &var       = %p\n",  &var    );

  sprintf(cmd, "cat /proc/%d/maps", getpid());
  system(cmd);

}
```

result for displaying /proc/<pid>/maps (1)

```
 &var        = 0xff9e00e8
08048000-08049000 r-xp 00000000 08:51 424080     /home/young/a.out
08049000-0804a000 r--p 00000000 08:51 424080     /home/young/a.out
0804a000-0804b000 rw-p 00001000 08:51 424080     /home/young/a.out
0851b000-0853c000 rw-p 00000000 00:00 0          [heap]
f75fb000-f75fc000 rw-p 00000000 00:00 0
f75fc000-f77a9000 r-xp 00000000 08:51 1475199    /lib32/libc-2.23.so
f77a9000-f77aa000 ---p 001ad000 08:51 1475199    /lib32/libc-2.23.so
f77aa000-f77ac000 r--p 001ad000 08:51 1475199    /lib32/libc-2.23.so
f77ac000-f77ad000 rw-p 001af000 08:51 1475199    /lib32/libc-2.23.so
f77ad000-f77b1000 rw-p 00000000 00:00 0
f77ce000-f77d0000 r--p 00000000 00:00 0          [vvar]
f77d0000-f77d2000 r-xp 00000000 00:00 0          [vdso]
f77d2000-f77f4000 r-xp 00000000 08:51 1475178    /lib32/ld-2.23.so
f77f4000-f77f5000 rw-p 00000000 00:00 0
f77f5000-f77f6000 r--p 00022000 08:51 1475178    /lib32/ld-2.23.so
f77f6000-f77f7000 rw-p 00023000 08:51 1475178    /lib32/ld-2.23.so
ff9c1000-ff9e2000 rw-p 00000000 00:00 0          [stack]
```

result for displaying /proc/<pid>/maps (2)

(from [1])

- address: the address space in the process that it occupies
- perms: a set of permissions
- offset: the offset into the mapping
- dev: the device (major:minor)
- inode: the inode on that device.
- pathname: shows the name associated file for this mapping

```
address           perms offset   dev   inode     pathname
fffa8000-fffc9000 rw-p  00000000 00:00 0         [stack]   (1)
ff9c1000-ff9e2000 rw-p  00000000 00:00 0         [stack]   (2)
```

[1] https://stackoverflow.com/questions/19379793/how-to-identify-stack-and-heap-segments-in-proc-pid-maps-file

result for displaying /proc/<pid>/maps (3)

- perms: a set of permissions
  - r = read
  - w = write
  - x = execute
  - s = shared
  - p = private (copy on write)
- inode: the inode on that device.
  - 0 indicates that no inode is associated with the memory region
  - like BSS (uninitialized data)

result for displaying /proc/<pid>/maps (4)

- pathname
  - If the mapping is associated with a file:
    - the name of the associated file for this mapping
  - If the mapping is not associated with a file:
    - [heap] = the heap of the program
    - [stack] = the stack of the main process
    - [stack:1001] = the stack of the thread with tid 1001
    - [vdso] = the "virtual dynamic shared object", the kernel system call handler
    - empty = the mapping is anonymous.

result for displaying /proc/<pid>/maps (5)

```
addr(stack_var) = 0xff9200e8

address           perms offset   dev   inode     pathname
ff9c1000-ff9e2000 rw-p  00000000 00:00 0         [stack]
```

```
[from HIGH] ff9e2000
       |
       |
       V    ff9e00e8 :  &var
       |
       V
[to   LOW ] ff9c1000
```

ELF Stack Contents

[from HIGH] ff9e2000 - stack bottom

1. path name specified in exec()
2. environment variables
3. argv strings
4. argc
5. aux vectors

```
[from HIGH] ff9e2000
       |    1, 2, 3, 4, 5
       |
       V    ff9e00e8 :  &var
       |
       |
[to   LOW ] ff9c1000
```

checking /proc/<pid>/maps file (1)

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

extern char **environ;

int main(int argc, char *argv[]) {
  int var = 5;
  char cmd[64];

  printf("   *environ  = %p\n", *environ  );
  printf("1. argv[0]   = %p\n", argv[0]   );
  printf("2. environ   = %p\n", environ   );
  printf("3. argv      = %p\n", argv      );
  printf("4. &argc     = %p\n", &argc     );
  printf("   &var      = %p\n", &var);
```

checking /proc/<pid>/maps file (2)

```
  printf("   *environ  = %s\n", *environ  );
  printf("1. argv[0]   = %s\n", argv[0]   );
  printf("2. *environ  = %p\n", *environ );
  printf("3. *argv     = %p\n", *argv    );
  printf("4. argc      = %d\n", argc     );
  printf("   var       = %d\n", var);


  sprintf(cmd, "cat /proc/%d/maps", getpid());
  system(cmd);
  return 0;
}
```

checking /proc/<pid>/maps results (1)

```
.............. 0xffcc9000.....[from HIGH]

   *environ   = 0xffce938b             |
1. argv[0]    = 0xffce9383             |
2. environ    = 0xffce866c             |
3. argv       = 0xffce8664             |
4. &argc      = 0xffce85d0             |
   &var       = 0xffce8568             V

.............. 0xffcea000.....[to   LOW ]

   *environ   = LC_PAPER=ko_KR.UTF-8
1. argv[0]    = ./a.out
2. *environ   = 0xffce938b
3. *argv      = 0xffce9383
4. argc       = 1
   var        = 5
```

checking /proc/<pid>/maps results (2)

```
08048000-08049000 r-xp 00000000 08:51 424080   /home/young/a.out
08049000-0804a000 r--p 00000000 08:51 424080   /home/young/a.out
0804a000-0804b000 rw-p 00001000 08:51 424080   /home/young/a.out
08ca3000-08cc4000 rw-p 00000000 00:00 0        [heap]
f75fd000-f75fe000 rw-p 00000000 00:00 0
f75fe000-f77ab000 r-xp 00000000 08:51 1475199  /lib32/libc-2.23.so
f77ab000-f77ac000 ---p 001ad000 08:51 1475199  /lib32/libc-2.23.so
f77ac000-f77ae000 r--p 001ad000 08:51 1475199  /lib32/libc-2.23.so
f77ae000-f77af000 rw-p 001af000 08:51 1475199  /lib32/libc-2.23.so
f77af000-f77b3000 rw-p 00000000 00:00 0
f77d0000-f77d2000 r--p 00000000 00:00 0        [vvar]
f77d2000-f77d4000 r-xp 00000000 00:00 0        [vdso]
f77d4000-f77f6000 r-xp 00000000 08:51 1475178  /lib32/ld-2.23.so
f77f6000-f77f7000 rw-p 00000000 00:00 0
f77f7000-f77f8000 r--p 00022000 08:51 1475178  /lib32/ld-2.23.so
f77f8000-f77f9000 rw-p 00023000 08:51 1475178  /lib32/ld-2.23.so
ffcc9000-ffcea000 rw-p 00000000 00:00 0        [stack]
```

checking stack frames (1)

```
use gdb
break func3
run
backtrace

-------     -----------------     -------     ------
main()    -> func1()           -> func2() -> func3()
var           str                 i           c
3             "Hello, world!"     1           '\0'
-------     -----------------     -------     ------
#3            #2                  #1          #0
```

checking stack frames (2)

```
#include <stdio.h>
#include <unistd.h>

void func3( int *a ) {
  int c = '\0';

  printf("pid = %d; ", getpid());
  printf("Press <Enter> \n");

  c = fgetc( stdin );

  printf("c=%c\n", c);

  *a = 9;
}
```

```
void func2( char *s ) {
  int i = 1;

  func3( &i );
  printf("i = %d \n", i);
}

void func1( int m ) {
  char str[] = "Hello, world!";

  func2( str );
}

int main(void) {
  int var = 3;

  func1( var );
  return 0;
}
```

checking stack frames results (1)

```
gdb a.out --->
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
...
(gdb) break func3
Punto de interrupción 1 at 0x8048521: file t.c, line 7.
(gdb) run
Starting program: /home/young/a.out

Breakpoint 1, func3 (a=0xffffd068) at t.c:7
7           int c = '\0';
(gdb) backtrace
#0  func3 (a=0xffffd068) at t.c:7
#1  0x080485ab in func2 (s=0xffffd09e "Hello, world!") at t.c:19
#2  0x0804860e in func1 (m=3) at t.c:26
#3  0x08048648 in main () at t.c:32
(gdb)
```

checking stack frames results (2)

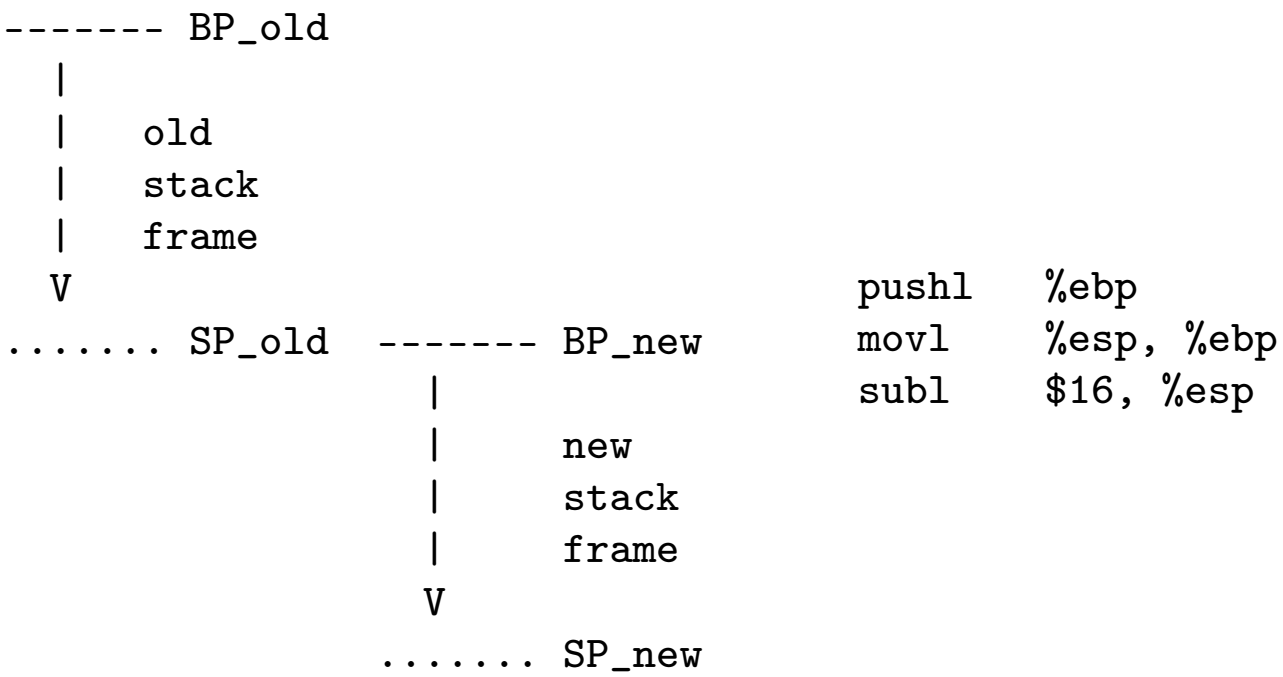
[from HIGH] - stack bottom

1. main() - gdb #3
2. func1() - gdb #2
3. func2() - gdb #1
4. func3() - gdb #0 — break point

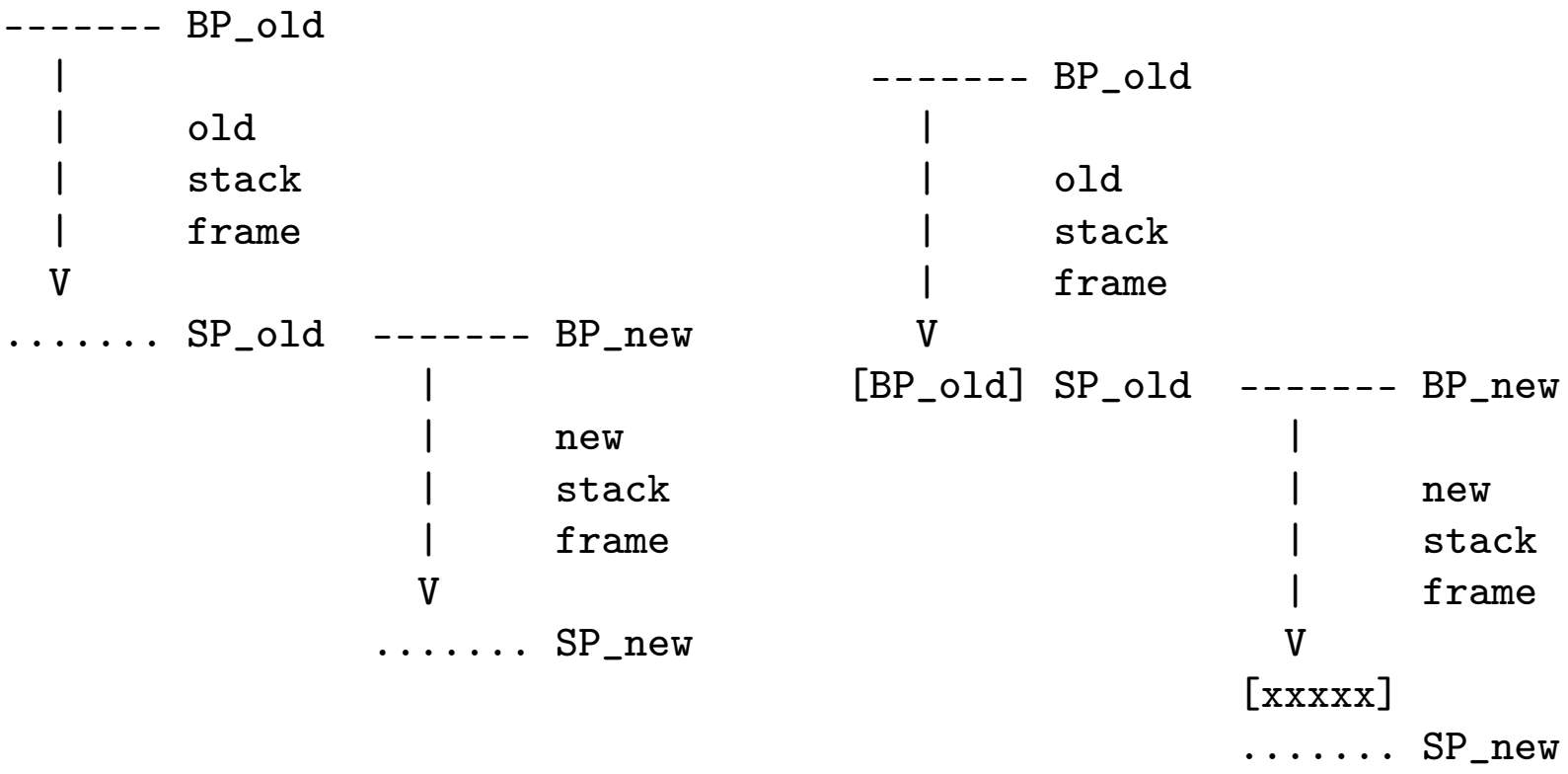
```
#0  func3 (a=0xffffd068) at t.c:7
#1  0x080485ab in func2 (s=0xffffd09e "Hello, world!") at t.c:19
#2  0x0804860e in func1 (m=3) at t.c:26
#3  0x08048648 in main () at t.c:32
```

Stack frame pointers (1)

- Full Downward Stack



Stack frame pointers (2)



- old BP must be saved
  on the stack
- pushl %ebp
- stack grows downward
- decreasing new SP
  for manual push operation

```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
```

functions and stack frames - c source code

```
#include <stdio.h>

void func1( int m ) {
  int i = 99;
}

int main(void) {
  int i = 3;

  func1( i );
  return 0;
}
```

```
gcc -m32 -S -Wall t.c
t.c -> t.s

t.c
```

functions and stack frames - generated assembly (1)

```
.file   "t.c"
.text
.globl  func1
.type   func1, @function
```

```
func1:
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    $99, -4(%ebp)
        nop
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   func1, .-func1
        .globl  main
        .type   main, @function
```

functions and stack frames - generated assembly (2)

```
main:
.LFB1:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    $3, -4(%ebp)
        pushl   -4(%ebp)
        call    func1
        addl    $4, %esp
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1:
```

```
.size   main, .-main
.ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4)
.section        .note.GNU-stack,"",@progbits
```

functions and stack frames - func1() analysis

```
func1:
.LFB0:
        ----------------------
        --function prologue---
        ----------------------
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        ----------------------
        movl    $99, -4(%ebp)
        nop
        ----------------------
        --function epilogue---
        ----------------------
        leave
        ret
        ----------------------
.LFE0:
```

```
void func1( int m ) {
  // the local stack variable
  // stored at %ebp-4
  int i = 99;
}
```

functions and stack frames - main() analysis

```
main:
.LFB1:
        ----------------------
        --function prologue---      int main(void) {
        ----------------------        // local stack variable
        pushl   %ebp                  // at %ebp-4
        movl    %esp, %ebp            int i = 3;
        subl    $16, %esp
        ----------------------        // the argument $3
        movl    $3, -4(%ebp)          // the value at %ebp-4
        pushl   -4(%ebp)              // pushed on the stack
        call    func1                 // before calling func1()
        addl    $4, %esp              func1( i );
        movl    $0, %eax
        ----------------------        // return value $0
        --function epilogue---        // is stored in %eax
        ----------------------        return 0;
        leave                       }
        ret
        ----------------------
.LFE1:
```

Function Prologue

1. pushl %ebp
   - stores the BP (base pointer) of the previous frame on the stack
   - decrements SP by 4 bytes (long: 32bits)
2. movl %esp, %ebp
   - updates the BP (base pointer) with the SP (stack pointer)
   - BP always points to the current stack bottom (higher address)
   - SP always points to the current stack top (lower address)
   - SP always points to full (non-empty) word
3. subl $16, %esp
   - to push the local variable i (int, 4 bytes)
   - stack grows downward
   - SP points 4 bytes lower address
   - 16-byte alignment

Function Epilogue

1. leave
   - movl %ebp, %esp
   - popl %ebp
2. ret

```
----------------------
--function prologue---
----------------------
pushl   %ebp
movl    %esp, %ebp

old BP stored on the stack
old SP becomes new BP
new BP points to the stored old BP
```

```
----------------------
--function epilogue---
----------------------
movl   %ebp, %esp
popl   %ebp

current BP points to the stored old BP
this BP becomes new SP
take the stored old BP
```

Omitting frame pointer compiler option

```
-fomit-frame-pointer

-fno-omit-frame-pointer
```

call instruction

- push the return address
- jump to the start of the called function
- return address
  - the address of the instruction
  - immediately following the call instruction

```
- Direct call : call label
- Indirect call : call *operand
```

```
operand :
  - offset Imm
  - a base register Eb
  - an index register Ei
  - a scalefactor s
```

ret instruction

- pops the return address from the stack
- jump to the return address location
- the correct return address must be stored
  - where the SP points to
- leave instruction does this stack preparation

leave instruction

- prepares the stack for returning to the caller
- preforms the following equivalent instructions

```
leave
```

```
%esp points to where %ebp points to
%ebp points to
   the beginning of the stack
   where the old %ebp is stored
```

```
movl %ebp, %esp
pop %ebp
```

```
this saved old %ebp is restored
%esp is adjusted to the end of the caller's s
```

example C codes for checking the X86 calling convention

```
main()   --> func1()    --> func2()
val          val             val
0            0               i+j+k
func1(A)     func2(B,C,D)
.................................
```

```
#include <stdio.h>
#define A 1
#define B 2
#define C 3
#define D 4
```

```
int func2( int i, int j, int k ) {
  int val = i+j+k;

  return val;
}

int func1( int m ) {
  int val = 0;

  val = func2( B, C, D );
  return val;
}

int main(void) {
  int val = 0;

  val = func1( A );
  return val;
}
```

checking calling convention - generated assembly (1)

```
        .file   "t.c"
        .text
.....................................
        .globl  func2                   func2: [AAA]
        .type   func2, @function        .LFB0:
func2:  [AAA]                                [[ func2 ]]
        .size   func2, .-func2          .LFE0:
.....................................
        .globl  func1                   func1: [BBB]
        .type   func1, @function        .LFB1:
func1:  [BBB]                                [[ func1 ]]
        .size   func1, .-func1          .LFE1:
.....................................
        .globl  main                    main: [CCC]
        .type   main, @function         .LFB2:
main:   [CCC]                                [[ main ]]
        .size   main, .-main            .LFE2:
.....................................
        .ident  "GCC:...
        .section    ...
```

checking calling convention - generated assembly (2)

```
func2: [AAA]
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %eax, %edx
        movl    16(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0
```

```
func1: [BBB]
.LFB1:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    $0, -4(%ebp)
        pushl   $4
        pushl   $3
        pushl   $2
        call    func2
        addl    $12, %esp
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1:
```

checking calling convention - generated assembly (3)

```
main:
.LFB2: [CCC]
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        movl    $0, -4(%ebp)
        pushl   $1
        call    func1
        addl    $4, %esp
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE2:
```

pushing arguments on to the stack frame

- storing arguments onto the stack
- from the right to the left
- in the reverse order
- the first argument is save at the last

```
func(A, B, C, D);
     <---------
```

```
push D -> push C -> push B -> push A
```

storing return value to %eax regsiter

- store the return value in the %eax register

```
movl    -4(%ebp), %eax
leave
ret
```

```
stack grows downward
%ebp    : BP
%ebp-4 : 4 bytes downward from BP
the word at %ebp-4 is moved to %eax

a local stack variable is stored here
its value is the return value

the old BP is stored at %ebp
```

Analyzing func2() - (1)

```
func2:
.LFB0:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %eax, %edx
        movl    16(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        ..........................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
```

```
int func2( int i, int j, int k ) {
  int val = i+j+k;

  return val;
}
```

main() -> func1() -> func2()

- just after func2() is called
- %ebp & %esp of the caller func1()
- [old %ebp] %ebp of main()

```
%ebp - 0 : [old %ebp]    <-- %ebp
%ebp - 4 : $0
%ebp - 8 : $4
%ebp - c : $3
%ebp -10 : $2
%ebp -14 : [Ret address] <-- %esp
```

Analyzing func2() - (2)

```
func2:
.LFB0:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %eax, %edx
        movl    16(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        ..........................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
```

```
- after function prolog is executed
- [old %ebp] %ebp of func1()
%ebp - 4 : $0
%ebp - 8 : $4                          %ebp+10
%ebp - c : $3                          %ebp+c
%ebp -10 : $2                          %ebp+8
%ebp -14 : [Ret address]               %ebp+4
%ebp -18 : [old %ebp]  <-- %esp <-- %ebp

- just before leave
%ebp +14 : $0
%ebp +10 : $4
%ebp + c : $3
%ebp + 8 : $2
%ebp + 4 : [Ret address]
%ebp     : [old %ebp] <-- %ebp
%ebp - 4 : [val]
 ...
%ebp -10 :            <-- %esp
```

Analyzing func1() - (1)

```
func1:
.LFB1:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    $0, -4(%ebp)
        ........................
        pushl   $4
        pushl   $3
        pushl   $2
        ........................
        call    func2
        addl    $12, %esp
        movl    %eax, -4(%ebp)
        ............................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
.LFE1:
```

```
#define B 2
#define C 3
#define D 4

int func1( int m ) {
  int val = 0;
  val = func2( B, C, D );
  return val;
}

- push arguements on the stack
- from the right argument
- to the left argument

- just after executing the function prologue

%ebp - 0 : [old %ebp] <-- %ebp <= %esp
%ebp - 4 :
%ebp - 8 :
%ebp - c :
%ebp -10 :            <-- %esp <= %esp-16
```

Analyzing func1() - (2)

```
func1:
.LFB1:
-----------------------------------  - just after call func2
        pushl   %ebp                 - call instruction pushes the
        movl    %esp, %ebp             return address on the stack
-----------------------------------
        subl    $16, %esp            %ebp - 4 : $0 (init val)  %esp+10
        movl    $0, -4(%ebp)         %ebp - 8 : $4             %esp+c
        ........................     %ebp - c : $3             %esp+8
        pushl   $4                   %ebp -10 : $2             %esp+4
        pushl   $3                   %ebp +14 : [ret addr] <-- %esp
        pushl   $2
        ........................
        call    func2                - just after ret from func2
        addl    $12, %esp            - ret instruction pops the
        movl    %eax, -4(%ebp)         return address from the stack
        ............................
        movl    -4(%ebp), %eax       %ebp - 4 : [updated val]  %esp+c
-----------------------------------  %ebp - 8 : $4             %esp+8
        leave                        %ebp - c : $3             %esp+4
        ret                          %ebp -10 : $2         <-- %esp
-----------------------------------
.LFE1:
```

Analyzing func1() - (3)

```
func1:
.LFB1:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    $0, -4(%ebp)
        ........................
        pushl   $4
        pushl   $3
        pushl   $2
        ........................
        call    func2
        addl    $12, %esp
        movl    %eax, -4(%ebp)
        ............................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
.LFE1:
```

- after returning from func2()
  decrease stack by 12 bytes
- the SP now points
  to the stack variable val
- the return value from func2 in %eax
- val is updated with this return value
- this value is stored to %eax
  the return value of func1()

```
%ebp - 4 : XX <-- %esp
%ebp - 8 : $4
%ebp - c : $3
%ebp -10 : $2
```

Analyzing main() - (1)

```
main:
.LFB2:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    $0, -4(%ebp)
        ......................
        pushl   $1
        ......................
        call    func1
        addl    $4, %esp
        movl    %eax, -4(%ebp)
        ......................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
.LFE2:
```

```
#define A 1

int main(void) {
  int val = 0;

  val = func1( A );
  return val;
}
```

- just after executing the function prologue

```
%ebp - 0 : [old %ebp] <-- %ebp <= %esp
%ebp - 4 :
%ebp - 8 :
%ebp - c :
%ebp -10 :            <-- %esp <= %esp-16
```

Analyzing main() - (2)

```
main:
.LFB2:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    $0, -4(%ebp)
        ........................
        pushl   $1
        ........................
        call    func1
        addl    $4, %esp
        movl    %eax, -4(%ebp)
        ............................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
.LFE2:
```

- just after call func1
- call instruction pushes the return address on the stack

```
%ebp - 4 : $0 (init val)  %esp+14
%ebp - 8 :                %esp+10
%ebp - c :                %esp+c
%ebp -10 :                %esp+8
%ebp +14 : $1             %esp+4
%ebp +18 : [ret addr] <-- %esp
```

- just after ret from func1
- ret instruction pops the return address from the stack

```
%ebp - 4 : $0 (init val)  %esp+10
%ebp - 8 :                %esp+c
%ebp - c :                %esp+8
%ebp -10 :                %esp+4
%ebp +14 : $1        <--  %esp
```

}

Analyzing main() - (3)

```
main:
.LFB2:
-----------------------------------
        pushl   %ebp
        movl    %esp, %ebp
-----------------------------------
        subl    $16, %esp
        movl    $0, -4(%ebp)
        ......................
        pushl   $1
        ......................
        call    func1
        addl    $4, %esp
        movl    %eax, -4(%ebp)
        ..........................
        movl    -4(%ebp), %eax
-----------------------------------
        leave
        ret
-----------------------------------
.LFE2:
```

- after returning from func1()
  decrease stack by 4 bytes

- the return value from func2 in %eax
- val is updated with this return value

- this value is stored to %eax
  the return value of main

```
%ebp - 4 : XX
%ebp - 8 :
%ebp - c :
%ebp -10 :    <-- %esp

}
```

References

[1] https://stackoverflow.com/questions/19379793/how-to-identify-stack-and-heap-se