Stack Debugging

Young W. Lim

2017-09-02

Outline

1. Based on
2. Introduction
   - Compiling to IA32 Assembly
   - Stack Frame
3. Workings of Stack Frames
   - Stack Frame Pointers
   - Basic Function Calls and Stack Frame Usage
4. Calling Conventions and Stack Frames
   - function call and return
   - function prologue and epilogue
   - enter and leave Instructions
   - arguments and return value
   - caller and callee's stack frames
   - x86 calling convention example codes
5. Function Call Examples
   - analyzing main()
   - analyzing func1()

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
  - 32-bit compiling (IA-32)

Stack Frame

- procedure calls
  - passing procedure arguments
  - receiving return informations
  - saving registers
- stack frame:
  - the portion of the stack
    - allocated for single procedure call
    - frame pointer (%ebp)
    - stack pointer (%esp)

Caller's Viewpoint

——— H.I.G.H. A.D.D.R.E.S.S. ———

- frame pointer (%ebp)
- saved registers
- local variables
- temporaries
- arguments for a funcion call to the callee
- return address
- stack pointer (%esp)

——— L.O.W. A.D.D.R.E.S.S. ———

local variables > function arguments > return address

Callee's Viewpoint

... .

- %ebp+c: *argument 2* from the caller
- %ebp+8: *argument 1* from the caller
- %ebp+4: *return address* of the caller

——————— H.I.G.H. A.D.D.R.E.S.S. ———————

- %ebp : caller's %ebp stored
- saved registers of the callee
- local variables of the callee
- temporaries of the callee

... .

——————— L.O.W. A.D.D.R.E.S.S. ———————

function arguments > return address > caller's %ebp > local variables

stack frame pointers (1)

- Full Downward Stack

```
.......<--BP_old
  |
  |   old                              pushl   %ebp
  |   stack                            movl    %esp, %ebp
  |   frame                            subl    $16, %esp
  V
.......<--SP_old  .......<-- BP_new - pushl %ebp
                    |                - the old BP must be saved on the stack
                    |        new     - stack grows downward
                    |        stack   - decreasing new SP
                    |        frame     for local variabls
                    V
                  .......<-- SP_new
```

stack frame pointers (2)

```
 ...........<-- BP_old
     |
     |          old
     |          stack
     |          frame
     V
...BP_old...<-- SP_old    ...........<-- BP_new
                              |
                              |          new
                              |          stack
                              |          frame
                              V
                            xxxxx
                          ...........<-- SP_new
```

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

c example 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
```

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
```

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
```

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 ) {
  // prepare space for
  // local variables (16 bytes)
  // var i is stored at %ebp-4
  int i = 99;
}
```

main() analysis

```
main:
.LFB1:
        ----------------------
        --function prologue---
        ----------------------
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        ----------------------
        movl    $3, -4(%ebp)
        pushl   -4(%ebp)
        call    func1
        addl    $4, %esp
        movl    $0, %eax
        ----------------------
        --function epilogue---
        ----------------------
        leave
        ret
        ----------------------
.LFE1:
```

```
int main(void) {
  // prepare space for
  // local variables (16 bytes)
  // var i is storead at %ebp-4
  int i = 3;

  // the argument i in func1(i)
  // the value $3 is at %ebp-4
  // pushed on the stack
  // before calling func1()
  // [%ebp-4] at %ebp-20 (20=16+4)
  func1( i );

  // return value $0
  // is stored in %eax
  return 0;
}
```

Omitting frame pointer compiler option

```
-fomit-frame-pointer
```

```
-fno-omit-frame-pointer
```

Function Call

1. push the return address (store %eip at %ebp+4)
2. jump to the start of the called function

- return address
  - the address of the instruction
  - immediately following the call instruction

```
after a call is executed             after executing a function prologue
    +--------------+                 +--------------+
    |              | <-- old %ebp    |              | <-- old %ebp
    :              :                 :              :
    |              |                 |              |
    +--------------+                 +--------------+
    | return addr  | <-- %esp        | return addr  |
    +--------------+                 +--------------+
    |              |                 | old %ebp     | <-- new %ebp, %esp
    +--------------+                 +--------------+
```

Function Return

1. pops the return address from the stack
2. jump to the return address location

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

```
after executing a function epilogue  after a call is executed
    +--------------+                     +--------------+
    |              | <-- %bsp (restored) |              | <-- %ebp (restored)
    :              :                     :              :
    |              |                     |              | <-- %esp
    +--------------+                     +--------------+
    | return addr  | <-- %esp            |              |
    +--------------+                     +--------------+
    |              |                     |              |
    +--------------+                     +--------------+
```

Function Prologue

1. push %ebp
   - stores the BP (base pointer) of the previous frame on the stack
   - decrements SP by 4 bytes (long: 32bits)
2. mov %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. sub $16 %esp
   - to allocate local variables
   - stack grows downward
   - SP points 4 bytes lower address
   - 16-byte alignment

Function Epilogue

1. mov %ebp %esp
    - deallocates locals and clear the current stack frame
    - increments the SP to the original SP
    - the original SP was stored in the BP
2. pop %ebp
    - restores the original BP
    - the current BP points where
    - the orginal BP was stored on the stack
3. ret
    - returns to the caller
    - pops the return address (the last value from the stack)
    - jumps to this return address

Comparison

enter instruction does:

- backup the old %ebp
- initialize

```
creates a new stack frame
----------------------
--function prologue---
----------------------
pushl   %ebp
movl    %esp, %ebp

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

leave instruction does:

- deallocate
- restore old %ebp

```
delete the current stack frame
----------------------
--function epilogue---
----------------------
movl    %ebp, %esp
popl    %ebp

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

enter instruction

- prepares the stack frame after entering to the callee
- equivalent instructions as follows

```
- push the current %ebp
- new %ebp points where this old %ebp is saved
- the last stack content is this savd %ebp
- the 2nd last stack content is the return address (^)

push %ebp        ;; push the old %ebp
mov  %esp %ebp   ;; new %ebp = the old %esp


before a call             after an enter
|..........|<--%ebp0      |..........|
|..........|              |..........|
|..........|              |..........|
|..........|<--%esp0      |.Ret Addr.|
                          |..%ebp0...|<--%esp <-- %ebp
                          |          |
                          |          |
```

leave instruction

- prepares the stack frame for returning to the caller
- equivalent instructions as follows

```
- current %esp points where the old %ebp was saved
- restore %ebp by this saved old %ebp
- the last stack content will be the return address (^)

movl %ebp, %esp  ;; restored %esp1 = the current %ebp
pop %ebp         ;; restored %ebp1 = the saved old %ebp0


before a leave             after a leave
|..........|<-- %ebp0      |..........|<--%ebp
|..........|               |..........|
|..........|               |..........|
|.Ret Addr.|               |.Ret Addr.|<--%esp
|   %ebp0  |<-- %ebp       |  %ebp0   |
|          |
|          |
```

pushing arguments on to the stack frame

- before calling a function
- storing arguments onto the current stack frame
- from the right to the left (reverse order)
- the first argument is save at the last (lower address)

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

push D; push C; push B; push A

- just before calling      - just after entering
  |..D (arg 4)..|            |..D (arg 4)..|
  |..C (arg 3). |            |..C (arg 3). |
  |..B (arg 2). |            |..B (arg 2). |
  |..A (arg 1)..|            |..A (arg 1)..|
  | Return addr |<-%esp      | Return addr |
                             | %ebp old    | <-%ebp <-%esp
```

storing return value to %eax regsiter

- store the return value in the %eax register

```
                              stack grows downward
                              %ebp   : BP
                              %ebp-4 : 4 bytes downward from BP
movl   -4(%ebp), %eax         the word at %ebp-4 is moved to %eax
leave
ret                           a local stack variable is stored here
                              its value is the return value

                              the old BP is stored where %ebp points to
```

caller and callee's stack frames

```
- CALLER's Stack           - CALLEE's Stack                - CALLEE's Stack
- just before calling      - just after entering           - local varaibles

  |.. (arg n) ..|            |.. (arg n) ..|                 |.. (arg n) ..|
  |..         ..|            |..         ..|                 |..         ..|
  |.. (arg 2) ..|            |.. (arg 2) ..|                 |.. (arg 2) ..|
  |.. (arg 1) ..|            |.. (arg 1) ..|                 |.. (arg 1) ..|
  | Return addr |<-%esp      | Return addr |                 | Return addr |
                             | %ebp old     | <-%ebp, %esp    | %ebp old     |
                                                             |..         ..|
                                                             |.. local    ..|
                                                             |.. var's    ..|
                                                             |..         ..|
```

TBD

- TBD

example C codes

```
--------------------------------
        main()  func1()  func2()
--------------------------------
local:  val     val      val
init :  0       0        i+j+k
call :  func1   func2
args :  (A)     (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;
}
```

generated assembly listing (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     ...
```

generated assembly listing (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:
```

generated assembly listing (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:
```

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           ;; %esp -= 16, allocate local variables
        movl    $0, -4(%ebp)        ;; local var val=0 at (%ebp-4)
        .......................
        pushl   $1                  ;; allocate the argument 1
        .......................
        call    func1               ;; func1(A)
        addl    $4, %esp            ;; deallocate the argument space
        movl    %eax, -4(%ebp)      ;; stpre the returned val at (%ebp-4)
        ...........................
        movl    -4(%ebp), %eax      ;; prepare the main's return value
-----------------------------------
        leave
        ret
-----------------------------------
.LFE2:
```

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:
```

```
- just after calling 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 returning 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() - (4)

```
main:
.LFB2:
-----------------------------------
        pushl   %ebp                 - after returning from func1()
        movl    %esp, %ebp             decrease stack by 4 bytes :
-----------------------------------    deallocate the arguments space
        subl    $16, %esp
        movl    $0, -4(%ebp)         - the return value from func2 is in %eax
        .......................      - val is updated with this return value
        pushl   $1
        .......................      - this value is stored to %eax
        call    func1                  for the main's return value
        addl    $4, %esp
        movl    %eax, -4(%ebp)
        ...........................  %ebp - 4 : XX   ;; val = func1(A);
        movl    -4(%ebp), %eax       %ebp - 8 :
-----------------------------------  %ebp - c :
        leave                        %ebp -10 :    <-- %esp
        ret
-----------------------------------
.LFE2:
```

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 most argument
- to the leftmost 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:
-----------------------------------
        pushl   %ebp                 ;; Function
        movl    %esp, %ebp           ;; Prologue
-----------------------------------
        subl    $16, %esp            ;; %esp -= 16, allocate local variables
        movl    $0, -4(%ebp)         ;; local var i=0 at (%ep-4)
        .......................
        pushl   $4                   ;; push arg D
        pushl   $3                   ;; push arg C
        pushl   $2                   ;; push arg B
        .......................
        call    func2                ;; func2(B,C,D)
        addl    $12, %esp            ;; deallocate arguments space (4*3=12)
        movl    %eax, -4(%ebp)       ;; update the local var val at (%ebp-4)
        ...........................
        movl    -4(%ebp), %eax       ;; prepare the func1's return value
-----------------------------------
        leave
        ret
-----------------------------------
.LFE1:
```

analyzing func1() - (3)

```
func1:
.LFB1:                              - just after calling func2
----------------------------------- - call instruction pushes the
        pushl   %ebp                  return address on the stack
        movl    %esp, %ebp
----------------------------------- %ebp - 4 : $0 (init val)  %esp+1c
        subl    $16, %esp           .................................
        movl    $0, -4(%ebp)        %ebp -14 : $4             %esp+c
        .......................     %ebp -18 : $3             %esp+8
        pushl   $4                  %ebp -1c : $2             %esp+4
        pushl   $3                  %ebp +20 : [ret addr] <-- %esp
        pushl   $2
        .......................
        call    func2               - just after returning from func2
        addl    $12, %esp           - ret instruction pops the
        movl    %eax, -4(%ebp)        return address from the stack
        ...........................
        movl    -4(%ebp), %eax      %ebp - 4 : $0 (init val)  %esp+18
----------------------------------- .................................
        leave                       %ebp -14 : $4             %esp+8
        ret                         %ebp -18 : $3             %esp+4
----------------------------------- %ebp -1c : $2             %esp
.LFE1:
```

analyzing func1() - (4)

```
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 :
  deallocate teh arguments space

- the return value from func3 is in %eax
- val is updated with this return value

- this value is stored to %eax
  for the func1's return value
```

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

%ebp - 4 : $0 (init val)  %esp+18
.................................
%ebp -10 :           <-- %esp
%ebp -14 : $4
%ebp -18 : $3
%ebp -1c : $2
%ebp +20 :
```

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;
}
```

- just after executing the function prologue

```
%ebp +14 : $0
%ebp +10 : $4
%ebp + c : $3
%ebp + 8 : $2
%ebp + 4 : [Ret addr]
%ebp - 0 : [old %ebp] <-- %ebp <= %esp’
%ebp - 4 :
%ebp - 8 :
%ebp - c :
%ebp -10 :           <-- %esp <= %esp’-16
```

analyzing func2() - (2)

```
func2:
.LFB0:
-----------------------------------
        pushl   %ebp                 ;; Function
        movl    %esp, %ebp           ;; Prologue
-----------------------------------
        subl    $16, %esp            ;; %esp -= 16, allocate local variables
        movl    8(%ebp), %edx        ;; %edx = [%ebp+8]   (=$2)
        movl    12(%ebp), %eax       ;; %eax = [%ebp+12] (=$3)
        addl    %eax, %edx           ;; %edx += %eax
        movl    16(%ebp), %eax       ;; %eax = [%ebp+16] (=$4)
        addl    %edx, %eax           ;; %eax += %edx
        movl    %eax, -4(%ebp)       ;; [%ebp-4] = %eax, local var val
        ...........................
        movl    -4(%ebp), %eax       ;; %eax = [%ebp-4], return val
-----------------------------------
        leave
        ret
-----------------------------------
```

analyzing func2() - (3)

```
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
-----------------------------------
```

```
- just before leave
%ebp +14 : $0
%ebp +10 : $4
%ebp + c : $3
%ebp + 8 : $2
%ebp + 4 : [Ret addr]
%ebp - 0 : [old %ebp] <-- %ebp <= %esp'
%ebp - 4 : val
%ebp - 8 :
%ebp - c :
%ebp -10 :             <-- %esp <= %esp'-16
```

example code overview

- main( ). . . . . {local: var=3}
    - func1(var)
- func1(int m). . . . . . . {local: str[]="Hello, world!"}
    - func2(str)
- func2(char *s). . . . . .{local: i=1}
    - func3(&i) - call by reference
    - print i
- func3(int *a). . . . . . .{local: c='\0'}
    - print pid & Press <Enter>
    - c=fgetc(stdin)
    - print c
    - *a=9

example c code

```
#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;
}
```

```
-------------------------------------
pid = 3534; Press <Enter>

c=

i = 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;
}
```

generated assembly structure

```
        .file   "t.c"
        .section        .rodata
.LC0:
        .string "pid = %d; "
.LC1:
        .string "Press <Enter> "
.LC2:
        .string "c=%c\n"
        .text
        .globl  func3
        .type   func3, @function
func3:  ============================
        .size   func3, .-func3
        .section        .rodata
```

```
.LC3:
        .string "i = %d \n"
        .text
        .globl  func2
        .type   func2, @function
func2:  ============================
        .size   func2, .-func2
        .globl  func1
        .type   func1, @function
func1:  ============================
        .size   func1, .-func1
        .globl  main
        .type   main, @function
main:   ============================
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~
        .section        .note.GNU-stack,"",@p
```

printf string constants

```
.LC0:
        .string "pid = %d; "
.LC1:
        .string "Press <Enter> "
.LC2:
        .string "c=%c\n"
.LC3:
        .string "i = %d \n"


char str[] = "Hello, world!";

$1819043144 0x6C6C6548 lleH
$1998597231 0x77202C6F w ,o
$1684828783 0x646C726F dlro
$33                    !
```

```
void func3( int *a ) {
  ...
  printf("pid = %d; ", getpid());
  printf("Press <Enter> \n");
  ...
  printf("c=%c\n", c);
  ...
}

void func2( char *s ) {
  ...
  printf("i = %d \n", i);
}
```

main c code

```
int main(void) {
  int var = 3;

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

call by reference : main (1)-a

```
main:
.LFB3:
        leal    4(%esp), %ecx             ;; %ecx= %esp+4 : %ecx-4=%esp
        andl    $-16, %esp                ;; & 0xFFFFFFF0
        pushl   -4(%ecx)                  ;; push [%ecx-4]        (%esp -= 4)
        pushl   %ebp ...................  ;; push %ebp...........(%esp -= 4)
        movl    %esp, %ebp .............  ;; %ebp= %esp.......................
        pushl   %ecx                      ;; push %ecx            (%esp -= 4)
        subl    $20, %esp                 ;; alloc local var's    (%esp -=20)
        movl    $3, -12(%ebp)             ;; [%ebp-12] = 3 ;; int var = 3;
        subl    $12, %esp                 ;; alloc argments       (%esp -=12)
        pushl   -12(%ebp)                 ;; push [%ebp-12]
        ................................
        call    func1                     ;; func1( var );
        addl    $16, %esp                 ;; dealloc (16=12+4)    (%esp +=16)
        movl    $0, %eax                  ;; %eax= 0  ;; return 0;
        movl    -4(%ebp), %ecx            ;; %ecx= [%ebp-4]
        leave   ........................  ;; %esp= %ebp, pop %ebp.(%esp += 4)
        leal    -4(%ecx), %esp            ;; %esp= %ecx-4
        ret     ........................  ;; pop return address...(%esp -= 4)
.LFE3:
```

call by reference : main (1)-b

```
main:
.LFB3:
%ecx= %esp+4 : %ecx-4=%esp          %ecx= init_%esp+4
& 0xFFFFFFF0                        16-byte alignment of %esp
push [%ecx-4]        (%esp -= 4)    push [init_%esp]  : content of init_%esp
push %ebp............(%esp -= 4)....push  init_%ebp   : address init_%ebp
%ebp= %esp..........................new_%ebp = %esp (aligned %esp)
push %ecx            (%esp -= 4)    push  init_%esp+4 : address init_%esp+4
alloc local var's    (%esp -=20)    enlarge %esp by 20
[%ebp-12] = 3 ;; int var = 3;       (%ebp-12) = 3     : initialize var
alloc argments       (%esp -=12)    enlarge %esp by 12
push [%ebp-12]                      push arg (=var)
................................    ........................................
func1( var );                       function call
dealloc (16=12+4)    (%esp +=16)    contrict %esp by 16 %esp += 16
%eax= 0  ;; return 0;               return value of main() = 0
%ecx= [%ebp-4]                      %ecx = saved_%ecx = %esp0+4
%esp= %ebp, pop %ebp.(%esp += 4)....%esp = init %ebp
%esp= %ecx-4                        %esp = saved_%esp = (%esp0+4)-4
pop return address...(%esp -= 4)....the next instruction to be executed
.LFE3:
```

call by reference : main (2)-a

```
|..........|<--%ebp0    |..........|
|..........|            |..........|
|...Z3Z2Z1.|<--%ecx     |...Z3Z2Z1.| <--%esp0+4
|.Z0Y3Y2Y1.|<--%esp0    |.Z0Y3Y2Y1.|
|.Y0.......|            |.Y3Y2Y1Y0.|
                        |..%ebp0...| <-- %ebp
                        |..%ecx....|
                       0|..........| -----------------------------------
                       1|.. $3 ....|             20 bytes     var=3
                       2|..........|             for local
                       3|..........|             variables
                       4|..........| -----------------------------------
                       0|..........|             12 bytes
                       1|..........|             for func1()
                       2|..........|             arguments
                        |.. $3 ....| <-- %esp                 func(var)
                        |..........|
                        |..........|
                        |..........|
```

call by reference : main (2)-b

```
4 bytes alignment examples

|..........|<--%ebp0    |..........|
|..........|            |..........|
|.....Z3Z2.|<--%ecx     |.....Z3Z2.| <--%esp0+4
|.Z1Z0Y3Y2.|<--%esp0    |.Z1Z0Y3Y2.|
|.Y1Y0.....|            |.Y3Y2Y1Y0.|
                        |..%ebp0...| <-- %ebp

|..........|<--%ebp0    |..........|
|..........|            |..........|
|...Z3Z2Z1.|<--%ecx     |...Z3Z2Z1.| <--%esp0+4
|.Z0Y3Y2Y1.|<--%esp0    |.Z0Y3Y2Y1.|
|.Y0.......|            |.Y3Y2Y1Y0.|
                        |..%ebp0...| <-- %ebp

|..........|<--%ebp0    |..........|
|..........|            |..........|
|.Z3Z2Z1Z0.|<--%ecx     |.Z3Z2Z1Z0.| <--%esp0+4
|.Y3Y2Y1Y0.|<--%esp0    |.Y3Y2Y1Y0.|
|.Y3Y2Y1Y0.|            |.Y3Y2Y1Y0.|
                        |..%ebp0...| <-- %ebp
```

func1 c code

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

  func2( str );
}
```

call by reference : func1 (1)-a

```
func1:
.LFB2:

        pushl   %ebp ..................  ;; push %ebp.............(%esp -= 4)
        movl    %esp, %ebp ............  ;; %ebp = %esp.....................
        subl    $40, %esp                ;;                      (%esp -=40)
        movl    %gs:20, %eax             ;; %eax = [%gs:20]
        movl    %eax, -12(%ebp)          ;; [%ebp-12] = %eax
        xorl    %eax, %eax               ;; %eax = 0
        movl    $1819043144, -26(%ebp)   ;; [%ebp-26] = "lleH"
        movl    $1998597231, -22(%ebp)   ;; [%ebp-22] = "w ,o"
        movl    $1684828783, -18(%ebp)   ;; [%ebp-18] = "dlro"
        movw    $33, -14(%ebp)           ;; [%ebp-14] = '!'
        subl    $12, %esp                ;;                      (%esp -=12)
        leal    -26(%ebp), %eax          ;; %eax = %ebp-26
        pushl   %eax                     ;; push %eax            (%esp -= 4)
```

call by reference : func1 (1)-b

```
func1:
.LFB2:
push %ebp ............(%esp -= 4)...  | function prologue
%ebp = %esp ........................  |
                      (%esp -=40)     | allocate 40 bytes for local variables
%eax = [%gs:20]                       . for checking the stack contamination
[%ebp-12] = %eax                      . save [%gs:20] at [%ebp-12]
%eax = 0                              .
[%ebp-26] = "lleH"                    | the string "Hello, world!" (13-byte)
[%ebp-22] = "w ,o"                    | from [%ebp-14]         to [%ebp-26]
[%ebp-18] = "dlro"                    | High add                    Low add
[%ebp-14] = ’!’                       |  !  d  l  r  o  w  ,  o  l  l  e  H
                      (%esp -=12)       allocate space for the function call
%eax = %ebp-26                          address of the start of the string
push %eax             (%esp -= 4)       the 1st argument on the stack
```

call by reference : func1 (2)-a

```
        ................................  ;; ...............................
        call    func2                     ;; func2( str );
        addl    $16, %esp                 ;;                      (%esp +=16)
        nop                               ;; nop
        movl    -12(%ebp), %eax           ;; %eax = [%ebp-12]
        xorl    %gs:20, %eax              ;; %eax ^= %gs:20
        je      .L5                       ;; if !%eax, jump
        ................................  ;; ................................
        call    __stack_chk_fail          ;; call stack check fail
.L5:                                      ;; .L5:
        leave...........................  ;; %esp= %ebp, pop %ebp.(%esp += 4)
        ret.............................  ;; pop return address...(%ESP += 4)
.LFE2:


----------------------------------------------------------------
char str[] = "Hello, world!";

$1819043144 0x6C6C6548 lleH
$1998597231 0x77202C6F w ,o
$1684828783 0x646C726F dlro
$33                    !
```

call by reference : func1 (2)-b

```
................................   ................................
func2( str );                      function call
                     (%esp +=16)   deallocates argument space for func2()
nop                                no operation
%eax = [%ebp-12]                   recover the stored [gs:20]
%eax ^= %gs:20                     xor with the current [gs:20]
if !%eax, jump                     if the same content, then jump to .L5
................................   ................................
call stack check fail              otherwise, handle the contaminated stack
.L5:                               .L5: label
%esp= %ebp, pop %ebp.(%esp += 4)...recover %esp and %ebp
pop return address...(%esp += 4)...jump to the saved return address
```

call by reference : func1 (3)-a

```
 |...%ebp0..|   <-- %ebp              3   2   1   0
0|..........|4                       -1  -2  -3  -4
1|..........|8                       -5  -6  -7  -8
2|.[%gs:20].|12                      -9 -10 -11 -12
3|...!.d.l..|16                     -13 -14 -15 -16
4|.r.o.w. ..|20                     -17 -18 -19 -20
5|.,.o.l.l..|24                     -21 -22 -23 -24
6|.e.H......|28 <-- %ebp-26         -25 -26 -27 -28
7|..........|32                     -29 -30 -31 -32
8|..........|36                     -33 -34 -35 -36
9|..........|40 <-- %esp1 <-- %esp4 -37 -38 -39 -40
0|..........|44                     -41 -42 -43 -44
1|..........|48                     -45 -46 -47 -48
2|..........|52 <-- %esp2           -49 -50 -51 -52
 |.%ebp-26..|56 <-- %esp3           -53 -54 -55 -56
 |..........|                         .   .   .   .
 |..........|                         .   .   .   .
 |..........|                         .   .   .   .
 |..........|                         .   .   .   .
```

func2 c code

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

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

```
 |...%ebp0..|   <-- %ebp
0|..........|4
1|..........|8
2|.[%gs:20].|12* stack contamination check
3|..i = 1...|16* local variable i
4|....... ..|20
5|..........|24
6|.(%ebp+8).|28*  1st arg : address of
7|..........|32  "Hello, world!"
8|..........|36
9|..........|40
0|..........|44
1|..........|48
2|..........|52 <-- %esp2
 |.(%ebp-16)|56 <-- %esp3 : func3's 1st arg
 |..........|        : address of local var i
 |..........|
 |..........|
 |..........|
```

func2 calls func3 by reference

```
 |...%ebp0..|    <-- %ebp
0|..........|4                       int i = 1;
1|..........|8                       func3( &i );
2|..........|12
3|..i = 1...|16* <-- %ebp-16         movl    $1, 16(%ebp)
4|....... ..|20
5|..........|24                      leal   -16(%ebp), %eax
6|..........|28                      pushl  %eax
7|..........|32
8|..........|36
9|..........|40
0|..........|44                      func3's first argument
1|..........|48                      : address of the local variable i
2|..........|52                      : %ebp-16 is storead on the stack
 |.(%ebp-16)|56* 1st arg ------- <-- %ebp3+8
 |.ret addr |
 |...%ebp...|------------------- <-- %ebp3 of func3
 |..........|

 func3 access this argument by %ebp3+8
 %ebp3 is the base pointer of func3's stack frame
```

call by reference : func2 (1)-a

```
func2:
.LFB1:
        pushl   %ebp................;; push %ebp..............(%esp -= 4)
        movl    %esp, %ebp..........;; %ebp = %esp.......................
        subl    $40, %esp           ;;                       (%esp -=40)
        movl    8(%ebp), %eax       ;; %eax = [%ebp+8]
        movl    %eax, -28(%ebp)     ;; [%ebp-28] = %eax
        movl    %gs:20, %eax        ;; %eax = %gs:20
        movl    %eax, -12(%ebp)     ;; [%ebp-12] = %eax
        xorl    %eax, %eax          ;; %eax = 0
        movl    $1, -16(%ebp)       ;; [%ebp-16] = 1  ;; int i = 1;
        subl    $12, %esp           ;;                       (%esp -=12)
        leal    -16(%ebp), %eax     ;; %eax = %ebp-16
        pushl   %eax                ;; push %eax             (%esp -= 4)
        ..........................
```

call by reference : func2 (1)-b

```
func2:
.LFB1:
push %ebp..............(%esp -= 4)..| function prologue
%ebp = %esp.........................|
                       (%esp -=40)  | allocate 40 bytes for local variables
%eax = [%ebp+8]                       1st arg - the address of "Hello, world!"
[%ebp-28] = %eax                      store 1st arg as a local var - never used
%eax = %gs:20                       . for checking the stack contamination
[%ebp-12] = %eax                    . save [%gs:20] at [%ebp-12]
%eax = 0                            .
[%ebp-16] = 1 ;; int i = 1;           initialize the local variable i
                       (%esp -=12)    allocate the func2 argument space
%eax = %ebp-16                        the address of the local variable i
push %eax              (%esp -= 4)    the 1st argument (&i) on the stack
..........................   (1)      push %eax (=&i)
```

call by reference : func2 (2)-a

```
        call    func3               ;; func3( &i );
        addl    $16, %esp           ;;                     (%esp +=16)
        movl    -16(%ebp), %eax     ;; %eax = [%ebp-16]
        subl    $8, %esp            ;; %esp -= 8
        pushl   %eax                ;; push %eax           (%esp -= 4)
        pushl   $.LC3               ;; push addr(.LC3),    (%esp -= 4)
        ..........................  ;; ................................
        call    printf              ;; printf("i = %d \n", i);
        addl    $16, %esp           ;;                     (%esp +=16)
        nop                         ;; nop
        movl    -12(%ebp), %eax     ;; %eax = [%ebp-12]
        xorl    %gs:20, %eax        ;; %eax ^= %gs:20
        je      .L3                 ;; if !%eax, jump
        ..........................  ;; ................................
        call    __stack_chk_fail    ;; call stack check fail
.L3:                                ;; .L3;
        leave.....................  ;; %esp= %ebp, pop %ebp.(%esp += 4)
        ret.......................  ;; pop return address...(%esp += 4)
.LFE1:
```

call by reference : func2 (2)-b

```
func3( &i );                      func3();
                    (%esp +=16)   deallocate local variable space
%eax = [%ebp-16]                  local variable in terms of %ebp
                    (%esp -= 8)   allocate printf argument space
push %eax           (%esp -= 4)   the 1st argument : local variable i
push addr(.LC3),    (%esp -= 4)   the 2nd argument : format string
..............................    ..............................
printf("i = %d \n", i);           printf();
(16=8+4+4)          (%esp +=16)   deallocate the printf argument space
nop                               nop
%eax = [%ebp-12]                  recover the stored [gs:20]
%eax ^= %gs:20                    xor with the current [gs:20]
if !%eax, jump                    if the same content, then jump to .L3
..............................    ..............................
call stack check fail             otherwise, handle the contaminated stack
.L3;                              L3: label
%esp= %ebp, pop %ebp.(%esp += 4)...recover %esp and %ebp
pop return address...(%esp += 4)...jump to the saved return address
```

func3 c code

```
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;
}
```

```
  |...%ebp0..|   <-- %ebp
 0|..........|4
 1|..........|8
 2|.int c= 0.|12* local var c=0
 3|..........|16
 4|....... ..|20
 5|..........|24
```

```
0|..........|28
1|..........|32
2|..%eax....|36* arg2: ret val of getpid()
3|..(.LC0)..|40* arg1: addr of "pid = %d\n"

0|..........|28
1|..........|32
2|..........|36
3|..(.LC1)..|40* arg1: address of
                   "Press <Enter> \n"
0|..........|28
1|..........|32
2|..........|36
3|..stdio...|40  arg1: stdio

0|..........|28
1|..........|32
2|.(%ebp-12)|36  arg2: Paddress of c
3|..(.LC2)..|40  arg1: address of
                       "c=%c\n"
```

func3 stack frame access

```
  | (%ebp0-16) |   <--%ebp+8 : func2's local var i's address
  | ret addr   |   <--%ebp+4 : func2's return address
  |...%ebp0....|   <--%ebp   : func2's saved %ebp0
 0|............|4
 1|............|8
 2|.int c= 0...|12*            func3's local var c=0
 3|............|16
 4|............|20
 5|............|24
```

```
movl 8(%ebp), %eax   ;; %eax = [%ebp+8] = %ebp0-16 which is the
                        address of the local variable i of func2
movl $9, (%eax)      ;; [%eax]= [[%ebp+8]] = [%ebp0-16] = 9
                        store 9 at the address of i of func2
```

call by reference : func3 (1)-a

```
func3:
.LFB0:
        pushl   %ebp.....................;; push %ebp...........(%esp -= 4)
        movl    %esp, %ebp...............;; %ebp = %esp...................
        subl    $24, %esp                ;;                     (%esp -=24)
        movl    $0, -12(%ebp)            ;; [%ebp-12] = 0  ;; int c = '\0';
        .......................          ;; ..............................
        call    getpid                   ;; getpid()
        subl    $8, %esp                 ;;                     (%esp -= 8)
        pushl   %eax                     ;; push %eax           (%esp -= 4)
        pushl   $.LC0                    ;; push addr(.LC0)     (%esp -= 4)
        .......................          ;; ..............................
        call    printf                   ;; printf("pid = %d; ", getpid());
        addl    $16, %esp                ;; (8+4+4)             (%esp +=16)
        subl    $12, %esp                ;;                     (%esp -=12)
        pushl   $.LC1                    ;; push addr(.LC1)     (%esp -= 4)
        .......................          ;; ..............................
```

call by reference : func3 (1)-b

```
func3:
.LFB0:
push %ebp.............(%esp -= 4).....| function prologue
%ebp = %esp...........................|
                      (%esp -=24)     | allocate 40 bytes for local variables
[%ebp-12] = 0                           int c = ’\0’;
.......................                 .....................................
getpid()                                getpid();
                      (%esp -= 8)       allocate the printf argument space
push %eax             (%esp -= 4)       2nd arg: getpid return value in %eax
push addr(.LC0)       (%esp -= 4)       1st arg: format string
.......................                 .....................................
printf("pid = %d; ", getpid());         printf();
(8+4+4)               (%esp +=16)       deallocate the printf argument space
                      (%esp -=12)       allocate the printf argument space
push addr(.LC1)       (%esp -= 4)       1st arg: format string
.......................                 .....................................
```

call by reference : func3 (2)-a

```
        call    puts                     ;; printf("Press <Enter> \n");
        addl    $16, %esp                ;; (12+4)                (%esp +=16)
        movl    stdin, %eax              ;; %eax = stdin
        subl    $12, %esp                ;;                       (%esp -=12)
        pushl   %eax                     ;; push %eax, %esp -= 4
        .......................          ;; .................................
        call    fgetc                    ;; c = fgetc( stdin );
        addl    $16, %esp                ;; (12+4)                (%esp +=16)
        movl    %eax, -12(%ebp)          ;; [%ebp-12] = %eax
        subl    $8, %esp                 ;;                       (%esp -= 8)
        pushl   -12(%ebp)                ;; push [%ebp-12]        (%esp -= 4)
        pushl   $.LC2                    ;; push addr(.LC2),      (%esp -= 4)
        .......................          ;; .................................
        call    printf                   ;; printf("c=%c\n", c);
        addl    $16, %esp                ;; (8+4+4)               (%esp +=16)
        movl    8(%ebp), %eax            ;; %eax = [%ebp+8]  ;; *a = 9;
        movl    $9, (%eax)               ;; [%eax] = 9
        nop                              ;; nop
        leave..........................  ;; %esp= %ebp, pop %ebp..(%esp += 4)
        ret............................  ;; pop return address....(%esp +=16)
.LFE0:
```

call by reference : func3 (2)-b

```
printf("Press <Enter> \n");        printf();
(12+4)                (%esp +=16)   deallocate the printf argument space
%eax = stdin                        allocate the fgetc argument space
                      (%esp -=12)   'stdin' in %eax is the 1st argpush %eax
.................................   ......................................
c = fgetc( stdin );                 fgetc();
%esp += 16 (12+4)                   deallocate the fgetc argument space
[%ebp-12] = %eax                    fgetc return val in %eax
                      (%esp -= 8)   allocate the printf argument space
push [%ebp-12],       (%esp -= 4)   2nd arg: int c value at [%ebp-12]
push addr(.LC2)       (%esp -= 4)   1st arg: format string
.................................   ......................................
printf("c=%c\n", c);                printf();
(8+4+4)               (%esp +=16)   deallocate the printf argument space
%eax = [%ebp+8]  ;; *a = 9;         (1) 1st arg (&i) (2) return address
[%eax] = 9                          store 9 at the (%ebp+8)
nop                                 no operation
%esp= %ebp, pop %ebp..(%esp += 4)...recover %esp and %ebp
pop return address....(%esp +=16)...jump to the saved return address
.LFE0:
```

Linux Stack Protector %gs:20 (1) [1]

Stack protector works by putting predefined pattern at the start of the stack frame and verifying that it hasn't been overwritten when returning from the function. The pattern is called stack canary and unfortunately gcc requires it to be at a fixed offset from %gs. On x86_64, the offset is 40 bytes and on x86_32 20 bytes. x86_64 and x86_32 use segment registers differently and thus handles this requirement differently.

On x86_64, %gs is shared by percpu area and stack canary. All percpu symbols are zero based and %gs points to the base of percpu area. The first occupant of the percpu area is always irq_stack_union which contains stack_canary at offset 40. Userland %gs is always saved and restored on kernel entry and exit using swapgs, so stack protector doesn't add any complexity there.

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

Linux Stack Protector %gs:20 (2) [1]

On x86_32, it's slightly more complicated. As in x86_64, %gs is used for userland TLS. Unfortunately, some processors are much slower at loading segment registers with different value when entering and leaving the kernel, so the kernel uses %fs for percpu area and manages %gs lazily so that %gs is switched only when necessary, usually during task switch.

As gcc requires the stack canary at %gs:20, %gs can't be managed lazily if stack protector is enabled, so the kernel saves and restores userland %gs on kernel entry and exit. This behavior is controlled by CONFIG_X86_32_LAZY_GS and accessors are defined in system.h to hide the details.

Linux Stack Protector %gs:20 (3) [1]

```
- putting predefined pattern at the start of the stack frame
- verifying that it hasn't been overwritten when returning

- stack canary
- a fixed offset from %gs (gcc)
  - 40 bytes for x86_64
  - 20 bytes for x86
- segment reg %gs
```

stack frame address range

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

stack frame in the Linux Mint

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

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);

}
```

results of displaying /proc/<pid>/maps

```
 &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]
```

fields in /proc/<pid>/maps

(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)
```

perms & inode fields

- 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)

pathname field

- 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.

stack elements in /proc/<pid>/maps file

```
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
```

```
int main(int argc, char *argv[])
             <-----------
push *argv[]
push argc
```

code for checking the main stack frame (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];

  ...

}
```

code for checking the main stack frame (2)

```
int main(int argc, char *argv[]) {

  int var = 5;
  char cmd[64];

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

code for checking the main stack frame (3)

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

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

results of checking the main stack frame (1)

```
2. environ   = 0xffb7e274 *environ  = 0xffb7f38b
3. argv      = 0xffb7e264 *argv     = 0xffb7f37d
3. argv[0]   = 0xffb7f37d *argv[0]  = .
3. &argv[0]  = 0xffb7e264 argv[0]   = 0xffb7f37d
3. &argv[1]  = 0xffb7e268 argv[1]   = 0xffb7f385
3. &argv[2]  = 0xffb7e26c argv[2]   = 0xffb7f388
.........................................................
2. environ   = 0xffb7e274 *environ  = 0xffb7f38b
3. argv      = 0xffb7e264 *argv     = 0xffb7f37d
4. &argc     = 0xffb7e1d0 argc      = 3
   &var      = 0xffb7e168 var       = 5
```

results of checking the main stack frame (2)

```
----------------------------------------------------------
2. environ   = 0xffb7e274 *environ  = LC_PAPER=ko_KR.UTF-8
3. argv      = 0xffb7e264 *argv     = ./a.out
3. argv[0]   = 0xffb7f37d *argv[0]  = .
3. &argv[0]  = 0xffb7e264 argv[0]   = ./a.out
3. &argv[1]  = 0xffb7e268 argv[1]   = aa
3. &argv[2]  = 0xffb7e26c argv[2]   = bb
.........................................................
2. environ   = 0xffb7e274 *environ  = 0xffb7f38b
3. argv      = 0xffb7e264 *argv     = 0xffb7f37d
4. &argc     = 0xffb7e1d0 argc      = 3
   &var      = 0xffb7e168 var       = 5
```

results of checking the main stack frame (3)

```
         +-----------+
&argv[2] |  argv[2]  |
         +-----------+
&argv[1] |  argv[1]  |
         +-----------+
&argv[0] |  argv[0]  |
         +-----------+


         +-----------+
         |           |
 argv[2] |   "bb"    |
         |           |
         +-----------+
         |           |
 argv[1] |   "aa"    |
         |           |
         +-----------+
         |           |
 argv[0] | "./a.out" |
         |           |
         +-----------+
```

```
0xffb7e26c | 0xffb7f388 |
0xffb7e268 | 0xffb7f385 |
0xffb7e264 | 0xffb7f37d |

0xffb7f38a | \0         |
0xffb7f389 | b          |
0xffb7f388 | b          |<-argv[2]
0xffb7f387 | \0         |
0xffb7f386 | a          |
0xffb7f385 | a          |<-argv[1]
0xffb7f374 | \0         |
0xffb7f373 | t          |
0xffb7f372 | u          |
0xffb7f371 | o          |
0xffb7f370 | .          |
0xffb7f37f | a          |
0xffb7f37e | /          |
0xffb7f37d | .          |<-argv[0]
```

results of checking the main stack frame (4)

```
08048000-08049000 r-xp 00000000 08:51 396707     /home/young/a.out
08049000-0804a000 r--p 00000000 08:51 396707     /home/young/a.out
0804a000-0804b000 rw-p 00001000 08:51 396707     /home/young/a.out
0895c000-0897d000 rw-p 00000000 00:00 0          [heap]
f7532000-f76e2000 r-xp 00000000 08:51 559546     /lib/i386-linux-gnu/libc-2.23.so
f76e2000-f76e4000 r--p 001af000 08:51 559546     /lib/i386-linux-gnu/libc-2.23.so
f76e4000-f76e5000 rw-p 001b1000 08:51 559546     /lib/i386-linux-gnu/libc-2.23.so
f76e5000-f76e8000 rw-p 00000000 00:00 0
f7708000-f770a000 rw-p 00000000 00:00 0
f770a000-f770c000 r--p 00000000 00:00 0          [vvar]
f770c000-f770e000 r-xp 00000000 00:00 0          [vdso]
f770e000-f7730000 r-xp 00000000 08:51 559518     /lib/i386-linux-gnu/ld-2.23.so
f7730000-f7731000 rw-p 00000000 00:00 0
f7731000-f7732000 r--p 00022000 08:51 559518     /lib/i386-linux-gnu/ld-2.23.so
f7732000-f7733000 rw-p 00023000 08:51 559518     /lib/i386-linux-gnu/ld-2.23.so
ffb5f000-ffb80000 rw-p 00000000 00:00 0          [stack]
```

using gdb to check stack frames

```
use gdb
break func3
run
backtrace

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

example code

```
#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;
}
```

gdb commands

```
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)
```

break func3, run, backtrace

gdb stack frame numbering

[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
```

Some gdb command (1) [2]

- print $eax
  print contents of register %eax in *decimal*.

- print /x $eax
  print contents of register %eax in *hex*.

- print /t $eax
  print contents of register %eax in *binary*.

- print /x ($eax + 8)
  print contents of memory address %eax + 8 in *hex*.

- print *(int *) (($eax + 8))
  print contents of memory address %eax + 8 *as an integer*

[2] https://cs61.seas.harvard.edu/wiki/Useful_GDB_commands

Some gdb command (2) [2]

- x ADDRESS
  print the value at a memory address.
  - x/d ADDRESS will print the value as an *integer*;
  - x/i ADDRESS as an *instruction*;
  - x/s ADDRESS as a string.
  - x/8xw ADDRESS will print 8 *four-byte words* in *hexadecimal* format.
    Note that ADDRESS can be written as a formula, e.g. $esp + 4.
    Try help x for more information.
- info registers
  print the values in the registers.
- info frame
  print information about the current stack frame.

Examining the Stack (1)

```
ddd a.out

break main
break func2

run
step


graph display ‘x /16wx $esp‘
```

```
(gdb) x /16wx $esp
0xffffd010:     0x00000000      0x00000001      0xf7ffd918      0x00f0b0ff
0xffffd020:     0xffffd05e      0x00000001      0x000000c2      0xf7e8d6bb
0xffffd030:     0xffffd05e      0xffffd15c      0xffffd078      0x0804860e
0xffffd040:     0xffffd05e      0xf7ffd918      0xffffd060      0x080482ba
```

Examining the Stack (2)

```
Dump of assembler code for function func2:
   0x08048581 <+0>:     push   %ebp
   0x08048582 <+1>:     mov    %esp,%ebp
   0x08048584 <+3>:     sub    $0x28,%esp
=> 0x08048587 <+6>:     mov    0x8(%ebp),%eax
   0x0804858a <+9>:     mov    %eax,-0x1c(%ebp)
   0x0804858d <+12>:    mov    %gs:0x14,%eax
   0x08048593 <+18>:    mov    %eax,-0xc(%ebp)
   0x08048596 <+21>:    xor    %eax,%eax
   0x08048598 <+23>:    movl   $0x1,-0x10(%ebp)
   0x0804859f <+30>:    sub    $0xc,%esp
   0x080485a2 <+33>:    lea    -0x10(%ebp),%eax
   0x080485a5 <+36>:    push   %eax
   0x080485a6 <+37>:    call   0x804851b <func3>
```

Examining the Stack (3)

```
Dump of assembler code for
   0x080485ab <+42>:    add    $0x10,%esp
   0x080485ae <+45>:    mov    -0x10(%ebp),%eax
   0x080485b1 <+48>:    sub    $0x8,%esp
   0x080485b4 <+51>:    push   %eax
   0x080485b5 <+52>:    push   $0x8048700
   0x080485ba <+57>:    call   0x80483b0 <printf@plt>
   0x080485bf <+62>:    add    $0x10,%esp
   0x080485c2 <+65>:    nop
   0x080485c3 <+66>:    mov    -0xc(%ebp),%eax
   0x080485c6 <+69>:    xor    %gs:0x14,%eax
   0x080485cd <+76>:    je     0x80485d4 <func2+83>
   0x080485cf <+78>:    call   0x80483c0 <__stack_chk_fail@plt>
   0x080485d4 <+83>:    leave
   0x080485d5 <+84>:    ret
End of assembler dump.
```