# Lecture Note 4.
# Process Structure

October 1, 2025

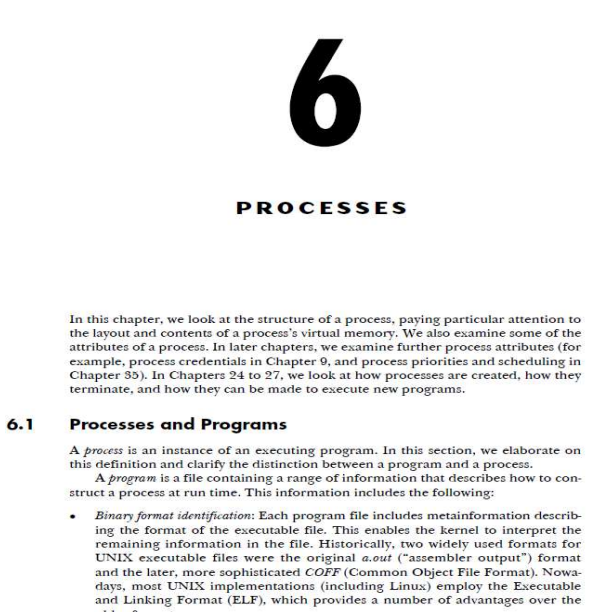
Jongmoo Choi
Dept. of Software
Dankook University
http://embedded.dankook.ac.kr/~choijm

(본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작 되었습니다.)

# Objectives

- Understand the definition of a process
- Explore the process structure
- Discuss the relation between program and process structure
- Grasp the details of stack

- Refer to Chapter 6 in the LPI and Chapter 8 in the CSAPP



6

PROCESSES

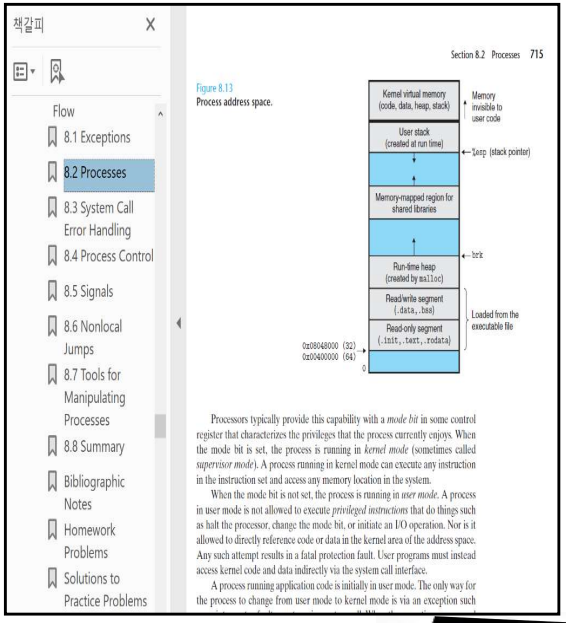
In this chapter, we look at the structure of a process, paying particular attention to the layout and contents of a process's virtual memory. We also examine some of the attributes of a process. In later chapters, we examine further process attributes (for example, process credentials in Chapter 9, and process priorities and scheduling in Chapter 35). In Chapters 24 to 27, we look at how processes are created, how they terminate, and how they can be made to execute new programs.

## 6.1 Processes and Programs

A *process* is an instance of an executing program. In this section, we elaborate on this definition and clarify the distinction between a program and a process.

A *program* is a file containing a range of information that describes how to construct a process at run time. This information includes the following:

- *Binary format identification*: Each program file includes metainformation describing the format of the executable file. This enables the kernel to interpret the remaining information in the file. Historically, two widely used formats for UNIX executable files were the original *a.out* ("assembler output") format and the later, more sophisticated *COFF* (Common Object File Format). Nowadays, most UNIX implementations (including Linux) employ the Executable and Linking Format (ELF), which provides a number of advantages over the



책갈피

Flow

8.1 Exceptions

8.2 Processes

8.3 System Call Error Handling

8.4 Process Control

8.5 Signals

8.6 Nonlocal Jumps

8.7 Tools for Manipulating Processes

8.8 Summary

Bibliographic Notes

Homework Problems

Solutions to Practice Problems

Section 8.2 Processes 715

Figure 8.13
Process address space.

Kernel virtual memory (code, data, heap, stack)
Memory invisible to user code
User stack (created at run time)
%esp (stack pointer)
Memory-mapped region for shared libraries
brk
Run-time heap (created by malloc)
Read/write segment (.data, .bss)
Read-only segment (.init, .text, .rodata)
Loaded from the executable file
0x08048000 (32)
0x00400000 (64)
0

Processors typically provide this capability with a *mode bit* in some control register that characterizes the privileges that the process currently enjoys. When the mode bit is set, the process is running in *kernel mode* (sometimes called *supervisor mode*). A process running in kernel mode can execute any instruction in the instruction set and access any memory location in the system.

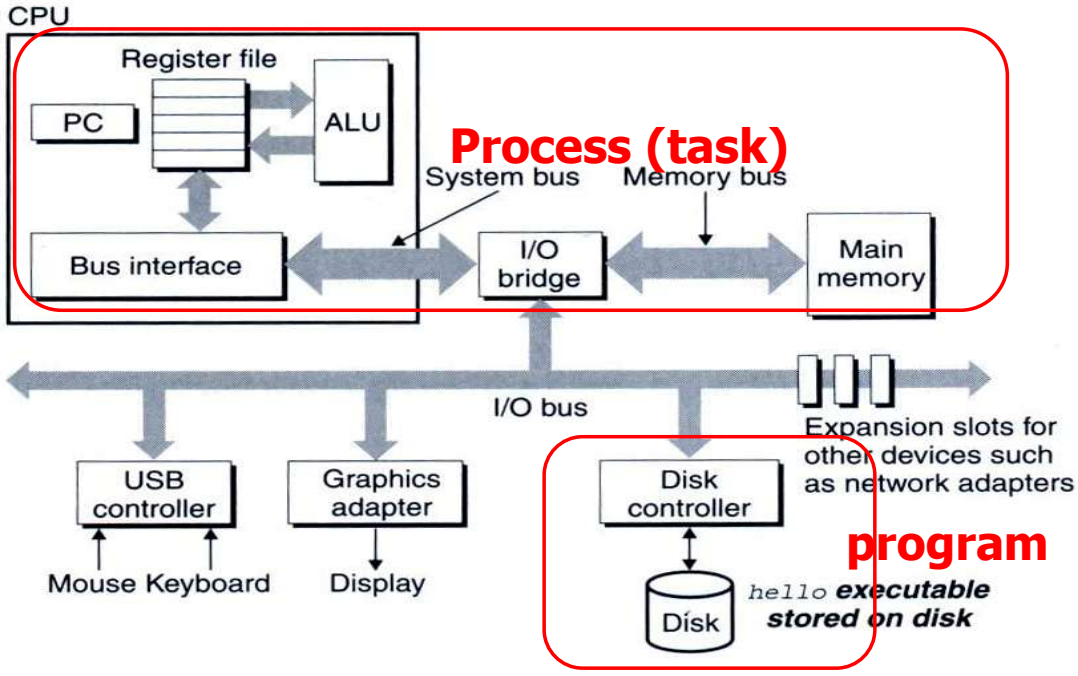
When the mode bit is not set, the process is running in *user mode*. A process in user mode is not allowed to execute *privileged instructions* that do things such as halt the processor, change the mode bit, or initiate an I/O operation. Nor is it allowed to directly reference code or data in the kernel area of the address space. Any such attempt results in a fatal protection fault. User programs must instead access kernel code and data indirectly via the system call interface.

A process running application code is initially in user mode. The only way for the process to change from user mode to kernel mode is via an exception such

# Process Definition (1/2)

- What is a process (also called as task)?
  - ✓ Program in execution
  - ✓ Having its own memory space and CPU registers
  - ✓ Scheduling entity
  - ✓ Conflict each other for resource allocation
  - ✓ Parent-child relation (family)

Figure 1.4
**Hardware organization of a typical system.** CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program Counter, USB: Universal Serial Bus.




(Source: CSAPP)

# Process Definition (2/2)

- Related terminology
  - ✓ Load
    - from disk into main memory
    - carried out by OS (e.g. page fault mechanism)
      - disk: file system (LN 3)
      - main memory: virtual memory (CSAPP 9, OS Course)
  - ✓ Fetch
    - from memory into CPU
    - carried out by hardware
      - Transparent to OS
      - instruction fetch and data fetch (LN 7)

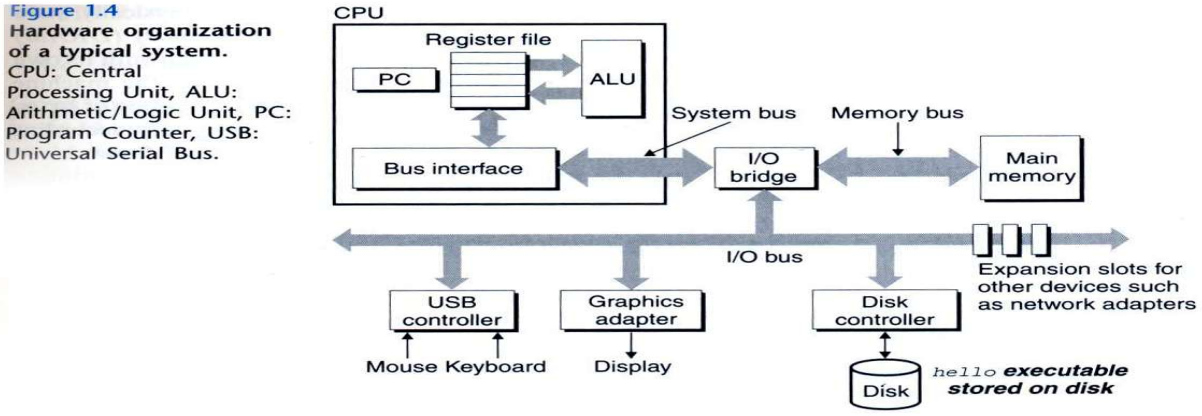



Figure 1.4 **Hardware organization of a typical system.** CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program Counter, USB: Universal Serial Bus.

# Process Structure (1/6)

- Conceptual structure
  - ✓ text, data, heap, stack

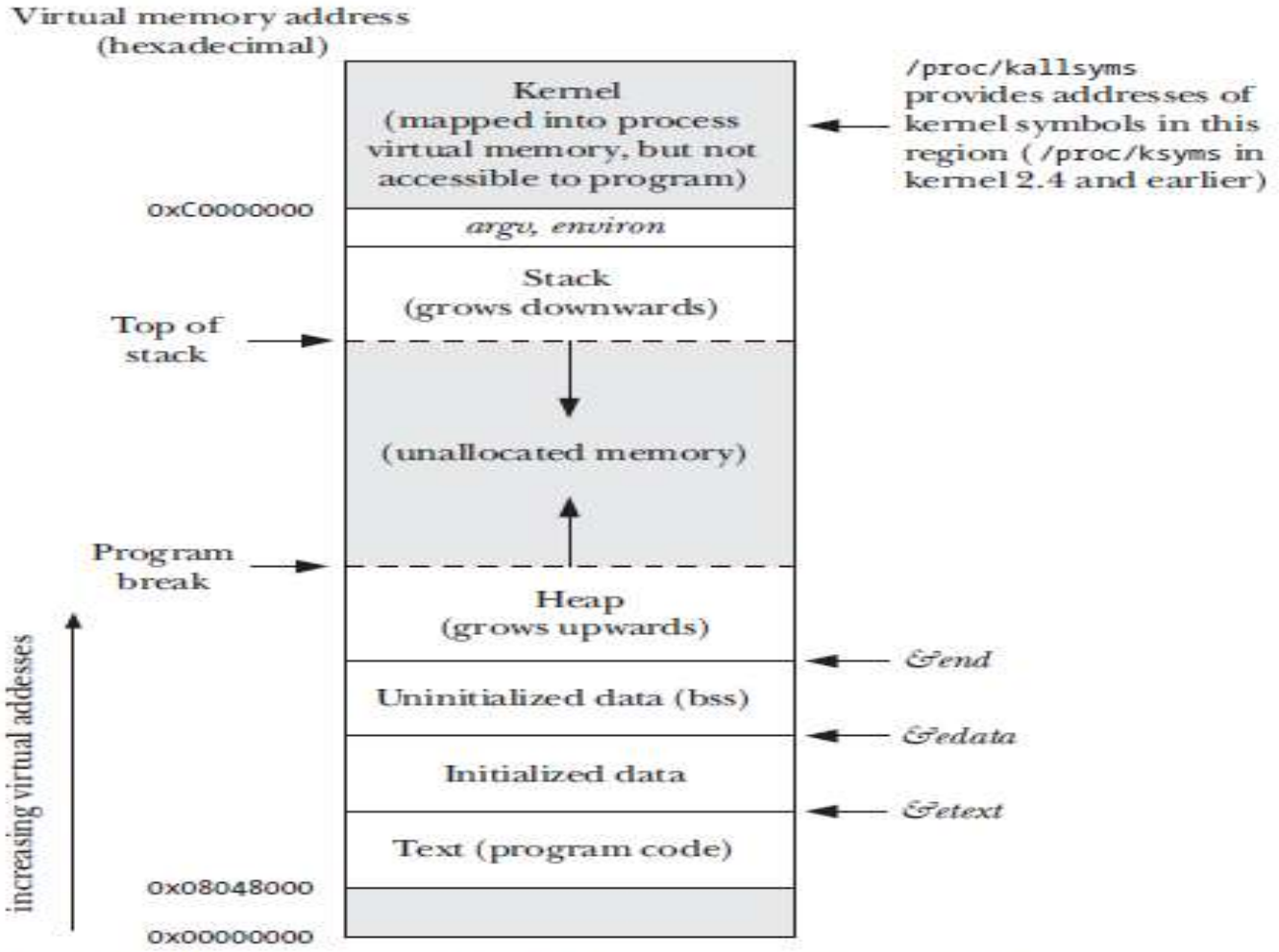



**Figure 6-1:** Typical memory layout of a process on Linux/x86-32

**(Source: LPI)**

# Process Structure (2/6)

- Process structure in C program: function pointer

```
/* f_pointer.c: for function pointer exercise, by choijm, choijm@dku.edu */
#include <stdio.h>

int a = 10;

int func1(int arg1)
{
   printf("In func1: arg1 = %d\n", arg1);
}

int main()
{
   int *pa;
   int (*func_ptr)(int);

   pa = &a;
   printf("pa = %p, *pa = %d\n", pa, *pa);
   func1(3);

   func_ptr = func1;
   func_ptr(5);

   printf("Bye..^^\n");
}
```

# Process Structure (3/6)

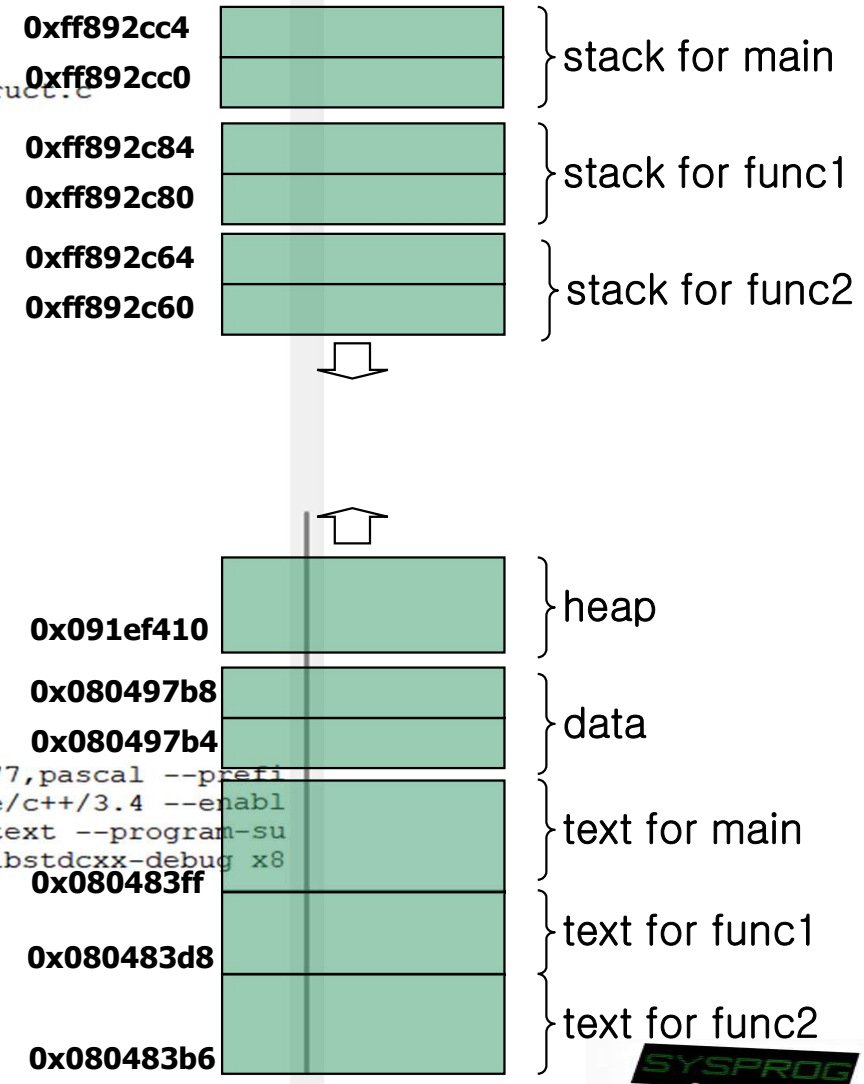
- Process structure in C program: address printing

```
/* task_struct.c: display addresses of variables and functions, choijm@dku.edu  */
#include <stdlib.h>
#include <stdio.h>

int glob1, glob2;

int func2() {
   int f2_local1, f2_local2;

   printf("func2 local: \n\t%p, \n\t%p\n", &f2_local1, &f2_local2);
}

int func1() {
   int f1_local1, f1_local2;

   printf("func1 local: \n\t%p, \n\t%p\n", &f1_local1, &f1_local2);
   func2();
}

int main(){
   int m_local1, m_local2; int *dynamic_addr;

   printf("main local: \n\t%p, \n\t%p\n", &m_local1, &m_local2);
   func1();

   dynamic_addr = malloc(16);
   printf("dynamic: \n\t%p\n", dynamic_addr);
   printf("global: \n\t%p, \n\t%p\n", &glob2, &glob1);
   printf("functions: \n\t%p, \n\t%p, \n\t%p\n", main, func1, func2);
}
```

# Process Structure (4/6)

- Process structure in C program: address printing

```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ ls
f_pointer  f_pointer.c  task_struct  task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o task_struct -m32 task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./task_struct
main local:
        0xff892cc4,
        0xff892cc0
func1 local:
        0xff892c84,
        0xff892c80
func2 local:
        0xff892c64,
        0xff892c60
dynamic:
        0x91ef410
global:
        0x80497b8,
        0x80497b4
functions:
        0x80483ff,
        0x80483d8,
        0x80483b6
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -v
Using built-in specs.
Configured with: ../src/configure -v --enable-languages=c,c++,f77,pascal --prefi
x=/usr --libexecdir=/usr/lib --with-gxx-include-dir=/usr/include/c++/3.4 --enabl
e-shared --with-system-zlib --enable-nls --without-included-gettext --program-su
ffix=-3.4 --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdcxx-debug x8
6_64-linux-gnu
Thread model: posix
gcc version 3.4.6 (Ubuntu 3.4.6-6ubuntu5)
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ whoami
choijm
choijm@system02:~/syspro/chap4$
```




☞ Addresses can be different based on Compiler, OS and CPU (32bit vs. 64bit)

# Process Structure (5/6)

- Summary
  - ✓ Process: consist of four regions, text, data, stack and heap

**Also called as segment or vm_object**

  - ✓ Text
    - Program code (assembly language)
    - Go up to the higher address according to coding order
  - ✓ Data
    - Global variable
    - Initialized and uninitialized data are managed separately (for the performance reason)
  - ✓ Stack
    - Local variable, argument, return address
    - Go down to the lower address as functions invoked
  - ✓ Heap
    - Dynamic allocation area (malloc(), calloc(), …)
    - Go up to the higher address as allocated

# Process Structure (6/6)

- Relation btw program and process

```
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ls
f_pointer  f_pointer.c  task_struct  task_struct.c  test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.c
#include <stdio.h>

int a = 10;
int b = 20;
int c;

int main()
{
        c = a + b;
        printf("C = %d\n", c);
}

choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$
```

```
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -S -m32 -mno-accumulate-outgoing-args test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.s
        .file   "test.c"
.globl a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
.globl b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    b, %eax
        addl    a, %eax
        movl    %eax, c
        subl    $8, %esp
        pushl   c
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Ubuntu 3.4.6-6ubuntu5)"
choijm@system02:~/syspro/chap4$
```

data

text

stack

# Process Structure in CSAPP (Optional)

- Another viewpoint for process structure
  - ✓ text, data, heap, stack + shared region, kernel

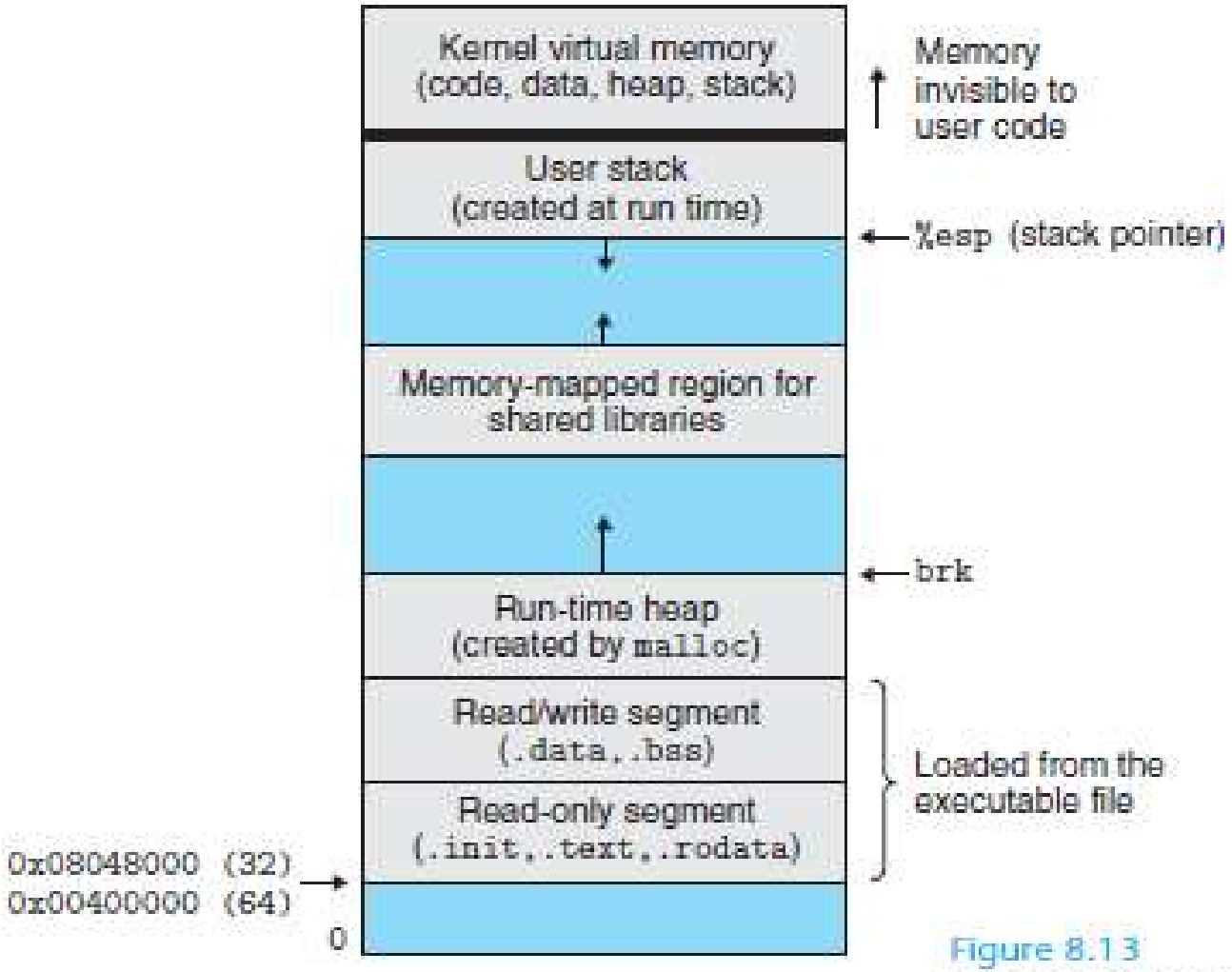



Figure 8.13
**Process address space.**

**(Source: CSAPP)**

# Stack Details (1/6)

- What is Stack?
  - ✓ A contiguous array of memory locations with LIFO property
    - Stack operation: push and pop
    - Stack management: base (bottom) and top (e.g. Stack Segment and ESP in intel)

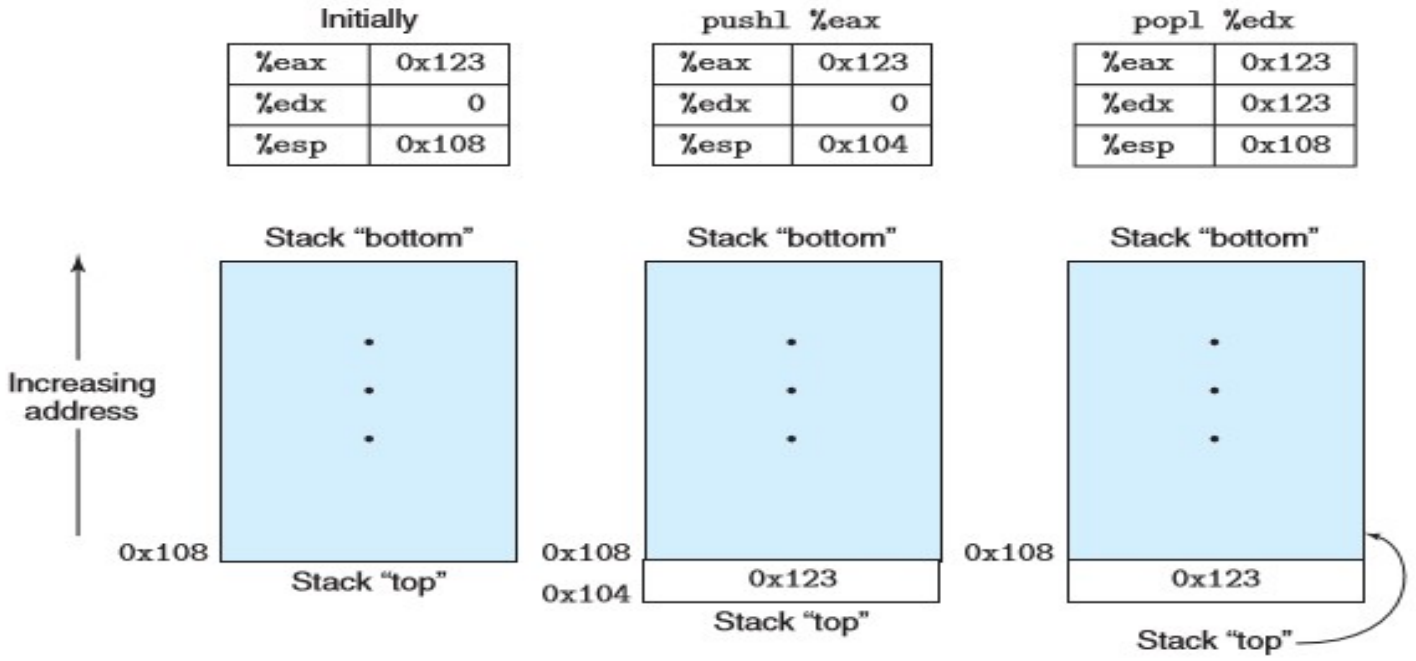
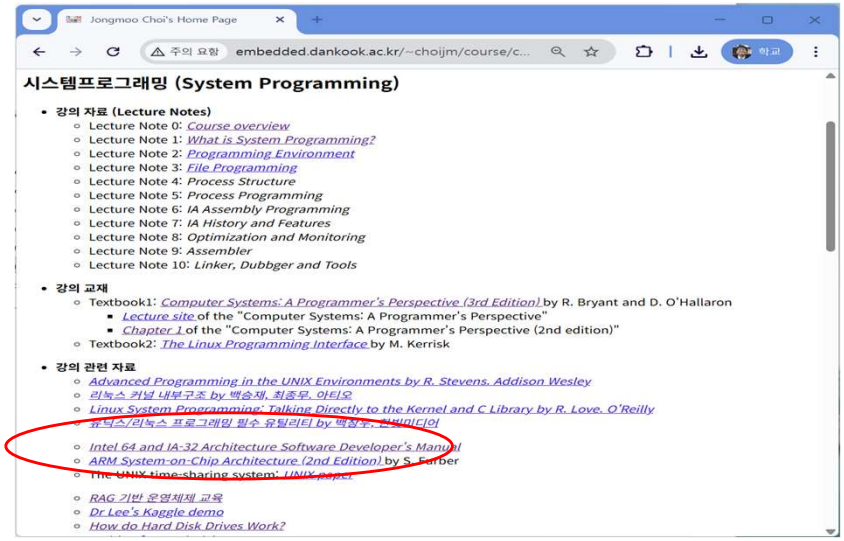


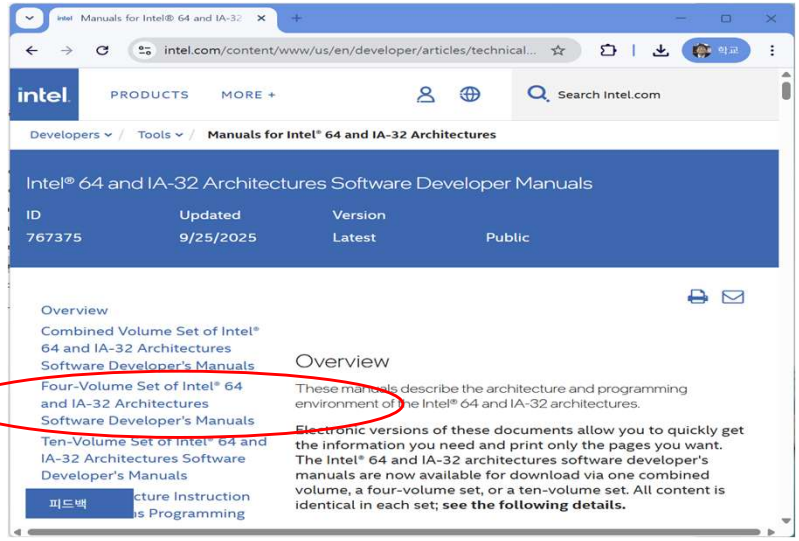

Figure 3.5 **Illustration of stack operation.** By convention, we draw stacks upside down, so that the "top" of the stack is shown at the bottom. IA32 stacks grow toward lower addresses, so pushing involves decrementing the stack pointer (register %esp) and storing to memory, while popping involves reading from memory and incrementing the stack pointer.

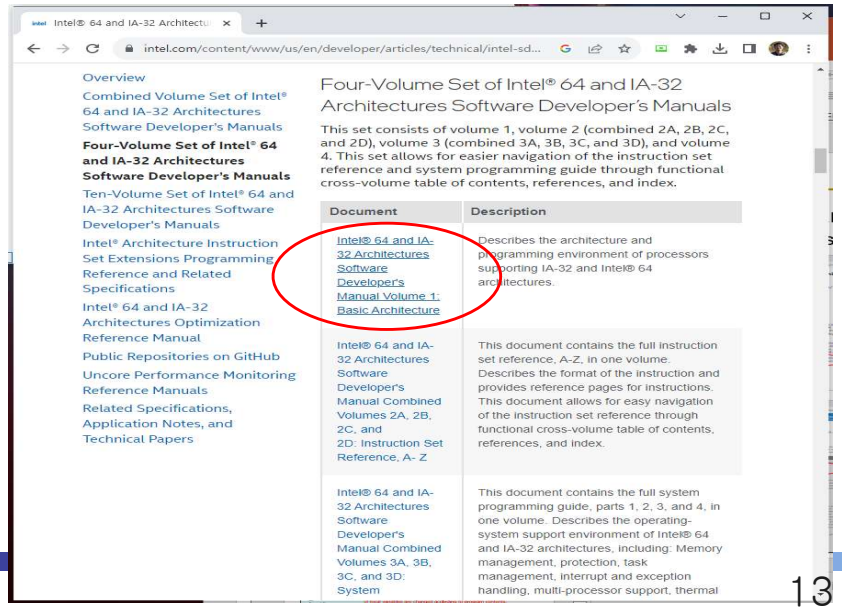
**(Source: CSAPP)**

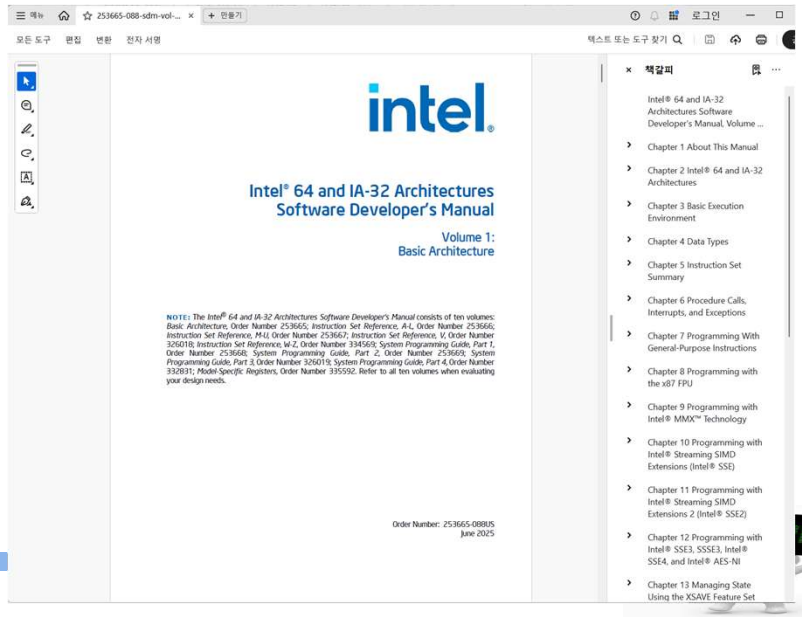
# Stack Details (2/6)

- Stack in Intel architecture
  - ✓ How to access Intel manual?



# Stack Details (3/6)

- Stack in Intel architecture
  - ✓ Real manipulation of push and pop
    - ESP (Extended Stack Pointer): pointing the top position
    - push: decrement the ESP and write data at the top of stack (down)
    - pop: read data from the top and increment the ESP (up)
  - ✓ What are in the stack?
    - 1) argument (parameters), 2) return address, 3) local variable, …
    - Return address: an address that returns after finishing a function (usually an address of an instruction after “call”)

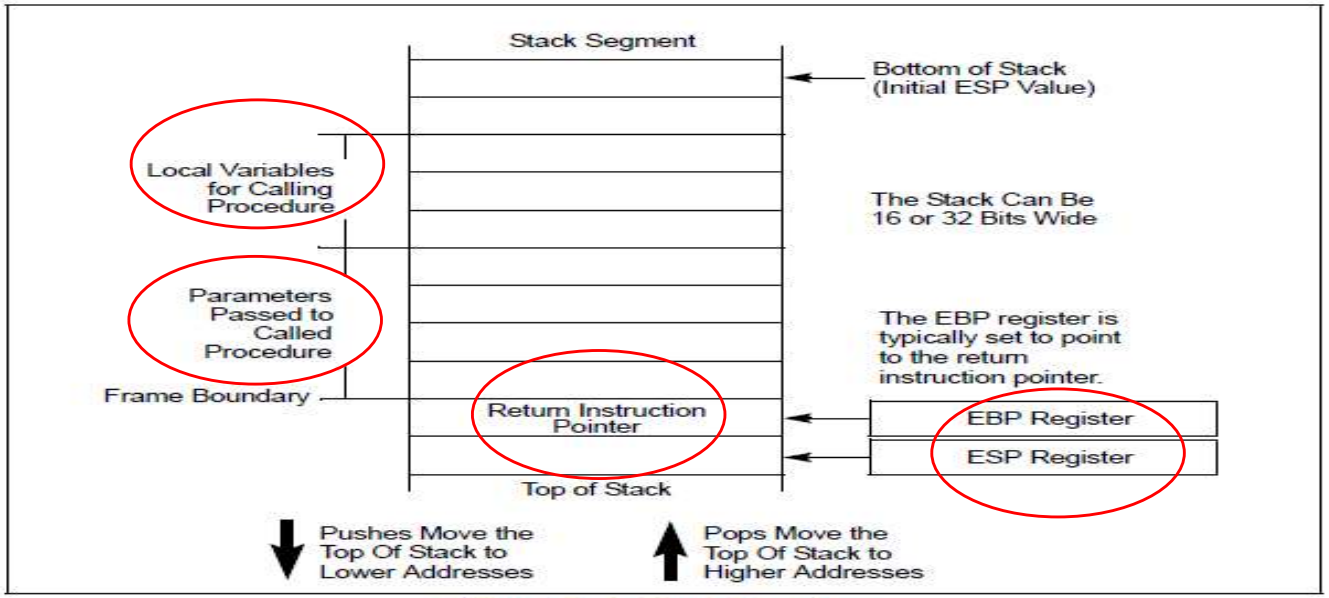


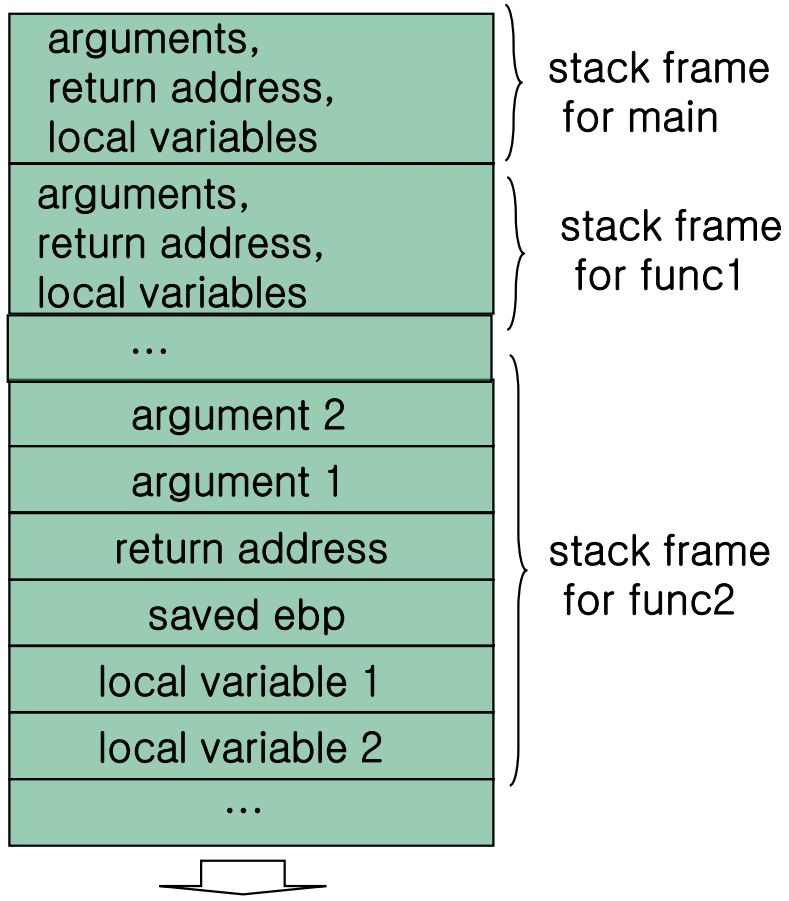

Figure 6-1. Stack Structure

(Source: Intel 64 and IA-32 Architectures Software Developer’s Manual)

# Stack Details (4/6)

## Stack in Linux

```
int func2(int x, int y) {
   int f2_local1 = 21, f2_local2 = 22;
   int *pointer, i;

   ...
}

void func1()
{
   int ret_val;
   int f1_local1 = 11, f1_local2 = 12;

   ...
   ret_val = func2(111, 112);
   f1_local++;
   ...
}

int main()
{
   ...
   func1();
   ...
}
```




- Compiler (and version) dependent (see **Appendix**)
- Especially, recent compiler makes use of obfuscation, where the locations of local variables are changed according to program contents.
- But, gcc 3.* version comply with the Intel's suggestion (like this figure)
  For lecturing purpose, gcc 3.* is more effective (Use 3.4 in this lecture note)

# Stack Details (5/6)

- Stack example 1

```
/* stack_struct.c: stack structure analysis, by choijm. choijm@dku.edu  */
#include <stdio.h>

int func2(int x, int y) {
   int f2_local1 = 21, f2_local2 = 22;
   int *pointer;

   printf("func2 local: \t%p, \t%p, \t%p\n", &f2_local1, &f2_local2, &pointer);
   pointer = &f2_local1;

   printf("\t%p  \t%d\n", (pointer), *(pointer));
   printf("\t%p  \t%d\n", (pointer-1), *(pointer-1));
   printf("\t%p  \t%d\n", (pointer+3), *(pointer+3));

   *(pointer+4) = 333;
   printf("\ty = %d\n", y);
   return 222;
}

void func1() {
   int ret_val, f1_local1 = 11, f1_local2 = 12;

   ret_val = func2(111, 112);
}

int main() {
   func1();
}
```

# Stack Details (6/6)

## Stack example 2

```
/* stack_destroy.c: 스택 구조 분석 2, 9월 19일, choijm@dku.edu  */
#include <stdio.h>

void f1() {
   int i;
   printf("In func1\n");
}

void f2() {
   int j, *ptr;
   printf("f2 local: \t%p, \t%p\n", &j, &ptr);
   printf("In func2 \n");

   ptr = &j;
   *(ptr+2) = f1;
}

void f3() {
   printf("Before invoke f2()\n");
   f2();
   printf("After invoke f2()\n");
}

main() {
   f3();
}
```

```
choijm@system02: ~/syspro/chap4
}

void f2() {
    int j, *ptr;
    printf("f2 local: \t%p, \t%p\n", &j, &ptr);
    printf("In func2 \n");

//    ptr = &j;
//    *(ptr+2) = f1;


}

void f3() {
    printf("Before invoke f2()\n");
    f2();
    printf("After invoke f2()\n");
}

main() {
    f3();
}
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xff9c5304,     0xff9c5300
In func2
After invoke f2()
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi stack_destroy.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:14: warning: assignment makes integer from pointer without a cast
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xffa46274,     0xffa46270
In func2
In func1
Segmentation fault (core dumped)
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$
```

# Summary

- Understand the differences between process and program
- Discuss the differences among text, data, heap and stack
- Find out the details of stack structure
  - ✓ Argument passing, Return address, Local variables
  - ✓ Stack overflow

☞ **Homework 4: Make a program of the stack example 2 and examine its results.**

**1.1 Requirements**

**- shows student's ID and date (using whoami and date)**

**- discuss why the segmentation fault occurs in this program**

**1.2 Bonus: overcome the segmentation fault problem**

**1.3 Deadline: Next week (same time)**

**1.4 How to submit? Send 1) report and 2) source code to google form**



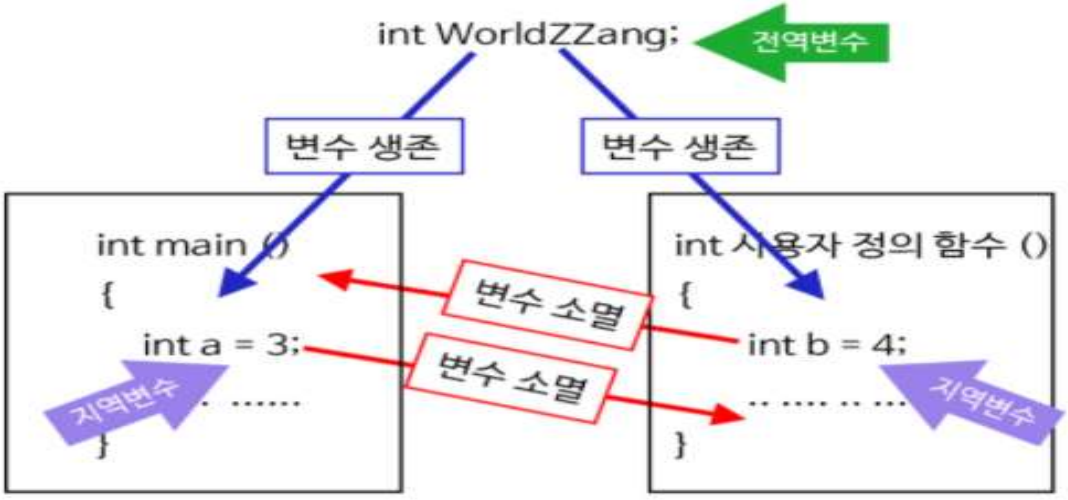
# Homework 4: Snapshot example

choijm@system02: ~/syspro/chap4

```
    printf("Before invoke f2()\n");
    f2();
    printf("After invoke f2()\n");
}

main() {
    f3();
}
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:        0xff9c5304,     0xff9c5300
In func2
After invoke f2()
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi stack_destroy.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:14: warning: assignment makes integer from pointer without a cast
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:        0xffa46274,     0xffa46270
In func2
In func1
Segmentation fault (core dumped)
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi stack_destroy.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:15: warning: assignment makes integer from pointer without a cast
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:        0xffc95bf4,     0xffc95bf0
In func2
In func1
After invoke f2()
choijm@system02:~/syspro/chap4$ whoami
choijm
choijm@system02:~/syspro/chap4$
```


# Quiz for this Lecture

- Quiz
  - ✓ 1. Explain the differences among 1) high-level program, 2) binary program, and 3) process.
  - ✓ 2. In C language, the scope of local variables and global variables are different. Discuss the reason of the differences using the process structure.
  - ✓ 3. Discuss the differences between stack and queue.
  - ✓ 4. Describe what are in the stack? (three key components)




(Source: https://dasima.xyz/c-local-global-variables/)

# Appendix 1

- Assembly differences with or without –m32
  - ✓ Task structure

```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ ls
f_pointer  f_pointer.c  task_struct  task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o task_struct -m32 task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./task_struct
main local:
        0xff892cc4,
        0xff892cc0
func1 local:
        0xff892c84,
        0xff892c80
func2 local:
        0xff892c64,
        0xff892c60
dynamic:
        0x91ef410
global:
        0x80497b8,
        0x80497b4
functions:
        0x80483ff,
        0x80483d8,
        0x80483b6
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -v
Using built-in specs.
Configured with: ../src/configure -v --enable-languages=c,c++,f77,pascal --prefi
x=/usr --libexecdir=/usr/lib --with-gxx-include-dir=/usr/include/c++/3.4 --enabl
e-shared --with-system-zlib --enable-nls --without-included-gettext --program-su
ffix=-3.4 --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdcxx-debug x8
6_64-linux-gnu
Thread model: posix
gcc version 3.4.6 (Ubuntu 3.4.6-6ubuntu5)
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ whoami
choijm
choijm@system02:~/syspro/chap4$
```

(with –m32 option)

```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ ls
f_pointer  f_pointer.c  task_struct  task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ vi task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -o task_struct task_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./task_struct
main local:
        0x7ffcd5da4e6c,
        0x7ffcd5da4e68
func1 local:
        0x7ffcd5da4e4c,
        0x7ffcd5da4e48
func2 local:
        0x7ffcd5da4e2c,
        0x7ffcd5da4e28
dynamic:
        0x66b420
global:
        0x600ab8,
        0x600ab4
functions:
        0x400511,
        0x4004e6,
        0x4004c5
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -v
Using built-in specs.
Configured with: ../src/configure -v --enable-languages=c,c++,f77,pascal --prefi
x=/usr --libexecdir=/usr/lib --with-gxx-include-dir=/usr/include/c++/3.4 --enabl
e-shared --with-system-zlib --enable-nls --without-included-gettext --program-su
ffix=-3.4 --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdcxx-debug x8
6_64-linux-gnu
Thread model: posix
gcc version 3.4.6 (Ubuntu 3.4.6-6ubuntu5)
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ whoami
choijm
choijm@system02:~/syspro/chap4$
```

(without –m32 option)

# Appendix 1

- Assembly differences with or without –m32
  - ✓ Assembly code

```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ more test.c
#include <stdio.h>

int a = 10;
int b = 20;
int c;

int main()
{
        c = a + b;
        printf("C = %d\n", c);
}

choijm@system02:~/syspro/chap4$ gcc -S -m32 test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.s
        .file   "test.c"
.globl a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
.globl b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    b, %eax
        addl    a, %eax
        movl    %eax, c
        movl    c, %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Ubuntu 3.4.6-6ubuntu5)"
choijm@system02:~/syspro/chap4$
```

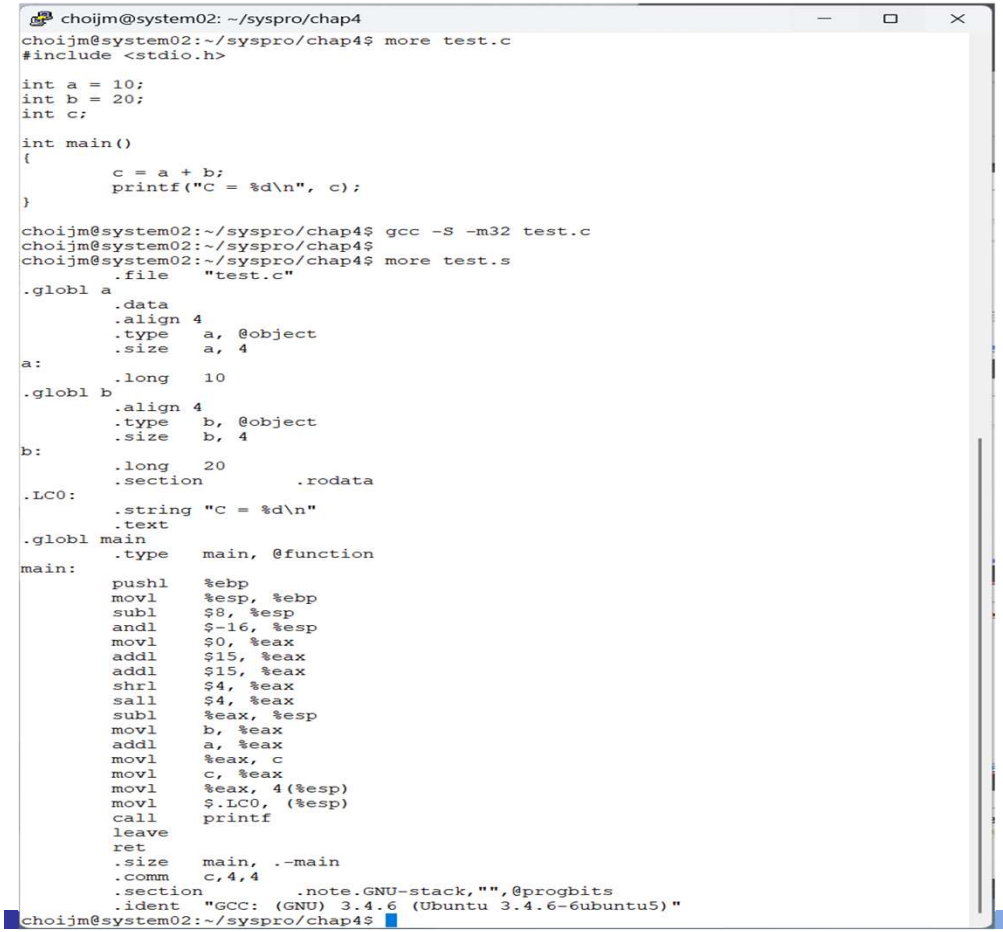
(with –m32 option)

```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ gcc -S test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.s
        .file   "test.c"
.globl a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
.globl b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
.globl main
        .type   main, @function
main:
.LFB2:
        pushq   %rbp
.LCFI0:
        movq    %rsp, %rbp
.LCFI1:
        movl    b(%rip), %eax
        addl    a(%rip), %eax
        movl    %eax, c(%rip)
        movl    c(%rip), %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        leave
        ret
.LFE2:
        .size   main, .-main
        .comm   c,4,4
        .section        .eh_frame,"a",@progbits
.Lframe1:
        .long   .LECIE1-.LSCIE1
.LSCIE1:
        .long   0x0
        .byte   0x1
        .string ""
        .uleb128 0x1
        .sleb128 -8
        .byte   0x10
        .byte   0xc
        .uleb128 0x7
        .uleb128 0x8
        .byte   0x90
        .uleb128 0x1
        .align 8
.LECIE1:
.LSFDE1:
        .long   .LEFDE1-.LASFDE1
.LASFDE1:
```

(without –m32 option)

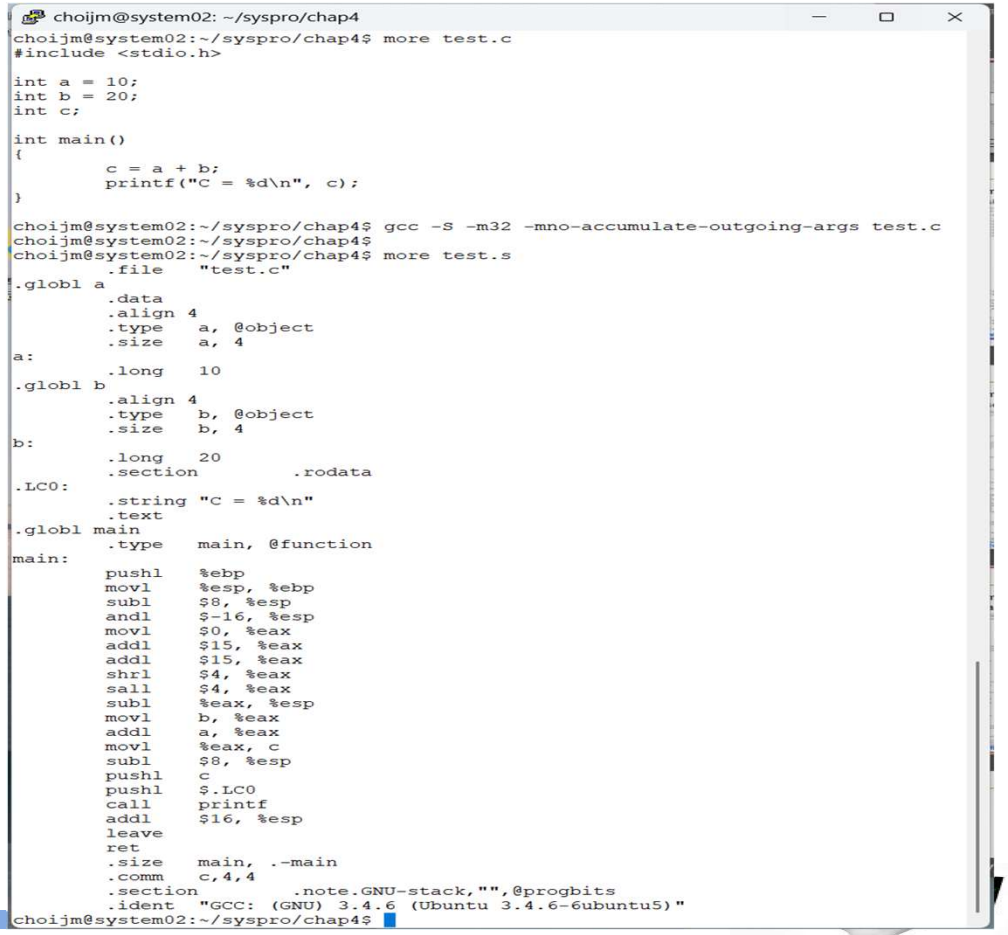
# Appendix 1

- Assembly differences with or without –mpush-args and –mno-accumulate-outgoing-args
  - ✓ Default: accumulate-outgoing-args ➔ allocate stack using a sub instruction ➔ good for performance (no sp adjustment per each argument due to reduced dependencies (enhanced pipeline



```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ more test.c
#include <stdio.h>

int a = 10;
int b = 20;
int c;

int main()
{
        c = a + b;
        printf("C = %d\n", c);
}

choijm@system02:~/syspro/chap4$ gcc -S -m32 test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.s
        .file   "test.c"
.globl a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
.globl b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    b, %eax
        addl    a, %eax
        movl    %eax, c
        movl    c, %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Ubuntu 3.4.6-6ubuntu5)"
choijm@system02:~/syspro/chap4$
```

(without args option)



```
choijm@system02: ~/syspro/chap4
choijm@system02:~/syspro/chap4$ more test.c
#include <stdio.h>

int a = 10;
int b = 20;
int c;

int main()
{
        c = a + b;
        printf("C = %d\n", c);
}

choijm@system02:~/syspro/chap4$ gcc -S -m32 -mno-accumulate-outgoing-args test.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ more test.s
        .file   "test.c"
.globl a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
.globl b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    b, %eax
        addl    a, %eax
        movl    %eax, c
        subl    $8, %esp
        pushl   c
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Ubuntu 3.4.6-6ubuntu5)"
choijm@system02:~/syspro/chap4$
```

(with args option)

# Appendix 2

- Stack difference between different compilers
  - ✓ Stack structure

choijm@system02: ~/syspro/chap4

```
    printf("func2 local: \t%p, \t%p, \t%p\n", &f2_local1, &f2_local2, &pointer);
    pointer = &f2_local1;

    printf("\t%p  \t%d\n", (pointer), *(pointer));
    printf("\t%p  \t%d\n", (pointer-1), *(pointer-1));
    printf("\t%p  \t%d\n", (pointer+3), *(pointer+3));

    *(pointer+4) = 333;
    printf("\ty = %d\n", y);
    return 222;
}

void func1() {
    int ret_val, f1_local1 = 11, f1_local2 = 12;

    ret_val = func2(111, 112);
}

int main() {
    func1();
}
choijm@system02:~/syspro/chap4$ gcc -o stack_struct -m32 stack_struct.c
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ ./stack_struct
func2 local:    0xffb2bcb4,     0xffb2bcb0,     0xffb2bcac
        0xffb2bcb4      21
        0xffb2bcb0      22
        0xffb2bcc0      111
        y = 333
choijm@system02:~/syspro/chap4$
choijm@system02:~/syspro/chap4$ gcc -v
Using built-in specs.
Configured with: ../src/configure -v --enable-languages=c,c++,f77,pascal --prefix=
/usr --libexecdir=/usr/lib --with-gxx-include-dir=/usr/include/c++/3.4 --enable-sh
ared --with-system-zlib --enable-nls --without-included-gettext --program-suffix=-
3.4 --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdcxx-debug x86_64-lin
ux-gnu
Thread model: posix
gcc version 3.4.6 (Ubuntu 3.4.6-6ubuntu5)
choijm@system02:~/syspro/chap4$
```

(gcc 3.4.6)

choijm@embedded: ~/syspro/chap4

```
    ret_val = func2(111, 112);
}

int main() {
    func1();
}
choijm@embedded:~/syspro/chap4$ gcc -o stack_struct -m32 stack_struct.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_struct
func2 local:    0xff861e20,     0xff861e24,     0xff861e28
        0xff861e20      21
        0xff861e1c      -7987672
        0xff861e2c      -1448266752
        y = 112
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-linux-gnu/13/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgcn-amdhsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 13.3.0-6ubuntu2~2
4.04' --with-bugurl=file:///usr/share/doc/gcc-13/README.Bugs --enable-languages=
c,ada,c++,go,d,fortran,objc,obj-c++,m2 --prefix=/usr --with-gcc-major-version-on
ly --program-suffix=-13 --program-prefix=x86_64-linux-gnu- --enable-shared --ena
ble-linker-build-id --libexecdir=/usr/libexec --without-included-gettext --enabl
e-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-cloca
le=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libst
dcxx-abi=new --enable-libstdcxx-backtrace --enable-gnu-unique-object --disable-v
table-verify --enable-plugin --enable-default-pie --with-system-zlib --enable-li
bphobos-checking=release --with-target-system-zlib=auto --enable-objc-gc=auto --
enable-multiarch --disable-werror --enable-cet --with-arch-32=i686 --with-abi=m6
4 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enab
le-offload-targets=nvptx-none=/build/gcc-13-fG75Ri/gcc-13-13.3.0/debian/tmp-nvpt
x/usr,amdgcn-amdhsa=/build/gcc-13-fG75Ri/gcc-13-13.3.0/debian/tmp-gcn/usr --enab
le-offload-defaulted --without-cuda-driver --enable-checking=release --build=x86
_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu --with-build-con
fig=bootstrap-lto-lean --enable-link-serialization=2
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)
choijm@embedded:~/syspro/chap4$
```

(gcc 13.3.0)

# Appendix 2

- Stack difference between different compilers
  - ✓ Assembly code: stack guard and obfuscation

choijm@system02: ~/syspro/chap4

```
choijm@system02:~/syspro/chap4$ gcc -S -m32 -mno-accumulate-outgoing-args stack_st
ruct.c
choijm@system02:~/syspro/chap4$ more stack_struct.s
        .file   "stack_struct.c"
        .section        .rodata
.LC0:
        .string "func2 local: \t%p, \t%p, \t%p\n"
.LC1:
        .string "\t%p  \t%d\n"
.LC2:
        .string "\ty = %d\n"
        .text
.globl func2
        .type   func2, @function
func2:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $24, %esp
        movl    $21, -4(%ebp)
        movl    $22, -8(%ebp)
        leal    -12(%ebp), %eax
        pushl   %eax
        leal    -8(%ebp), %eax
        pushl   %eax
        leal    -4(%ebp), %eax
        pushl   %eax
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        leal    -4(%ebp), %eax
        movl    %eax, -12(%ebp)
        subl    $4, %esp
        movl    -12(%ebp), %eax
        pushl   (%eax)
        pushl   -12(%ebp)
        pushl   $.LC1
        call    printf
        addl    $16, %esp
        subl    $4, %esp
        movl    -12(%ebp), %eax
        subl    $4, %eax
        pushl   (%eax)
        movl    -12(%ebp), %eax
        subl    $4, %eax
        pushl   %eax
```

(gcc 3.4.6)

choijm@embedded: ~/syspro/chap4

```
choijm@embedded:~/syspro/chap4$ gcc -S -m32 -mno-accumulate-outgoing-args stack_
struct.c
choijm@embedded:~/syspro/chap4$ more stack_struct.s
        .file   "stack_struct.c"
        .text
        .section        .rodata
.LC0:
        .string "func2 local: \t%p, \t%p, \t%p\n"
.LC1:
        .string "\t%p  \t%d\n"
.LC2:
        .string "\ty = %d\n"
        .text
        .globl  func2
        .type   func2, @function
func2:
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        pushl   %ebx
        subl    $20, %esp
        .cfi_offset 3, -12
        call    __x86.get_pc_thunk.bx
        addl    $_GLOBAL_OFFSET_TABLE_, %ebx
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    $21, -24(%ebp)
        movl    $22, -20(%ebp)
        leal    -16(%ebp), %eax
        pushl   %eax
        leal    -20(%ebp), %eax
        pushl   %eax
        leal    -24(%ebp), %eax
        pushl   %eax
        leal    .LC0@GOTOFF(%ebx), %eax
        pushl   %eax
        call    printf@PLT
        addl    $16, %esp
        leal    -24(%ebp), %eax
        movl    %eax, -16(%ebp)
```

(gcc 13.3.0)

# Appendix 2

- Stack destroy in gcc 13.3.0
  - ✓ We can make use of options such as –fno-stack-protector, -fno-omit-frame-pointer, -fno-pie, -no-pie, and –mpreferred-stack-boundary=2

```
choijm@embedded: ~/syspro/chap4
choijm@embedded:~/syspro/chap4$ vi stack_destroy.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xff972b74,     0xff972b78
In func2
After invoke f2()
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ vi stack_destroy.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_destroy -m32 stack_destroy.c
stack_destroy.c: In function 'f2':
stack_destroy.c:14:14: warning: assignment to 'int' from 'void (*)()' makes inte
ger from pointer without a cast [-Wint-conversion]
   14 |      *(ptr+2) = f1;
      |              ^
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xfffbcf74,     0xfffbcf78
In func2
*** stack smashing detected ***: terminated
Aborted (core dumped)
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-linux-gnu/13/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgcn-amdhsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 13.3.0-6ubuntu2~2
4.04' --with-bugurl=file:///usr/share/doc/gcc-13/README.Bugs --enable-languages=
c,ada,c++,go,d,fortran,objc,obj-c++,m2 --prefix=/usr --with-gcc-major-version-on
ly --program-suffix=-13 --program-prefix=x86_64-linux-gnu- --enable-shared --ena
ble-linker-build-id --libexecdir=/usr/libexec --without-included-gettext --enabl
e-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-cloca
le=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libst
dcxx-abi=new --enable-libstdcxx-backtrace --enable-gnu-unique-object --disable-v
table-verify --enable-plugin --enable-default-pie --with-system-zlib --enable-li
bphobos-checking=release --with-target-system-zlib=auto --enable-objc-gc=auto --
enable-multiarch --disable-werror --enable-cet --with-arch-32=i686 --with-abi=m6
4 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enab
le-offload-targets=nvptx-none=/build/gcc-13-fG75Ri/gcc-13-13.3.0/debian/tmp-nvpt
x/usr,amdgcn-amdhsa=/build/gcc-13-fG75Ri/gcc-13-13.3.0/debian/tmp-gcn/usr --enab
le-offload-defaulted --without-cuda-driver --enable-checking=release --build=x86
_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu --with-build-con
fig=bootstrap-lto-lean --enable-link-serialization=2
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04)
choijm@embedded:~/syspro/chap4$
```

(default in gcc 13.3.0)

```
choijm@embedded: ~/syspro/chap4
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ vi stack_destroy.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_destroy -m32 -fno-stack-protector -
fno-omit-frame-pointer -fno-pie -no-pie -mpreferred-stack-boundary=2 stack_destr
oy.c
stack_destroy.c: In function 'f2':
stack_destroy.c:14:14: warning: assignment to 'int' from 'void (*)()' makes inte
ger from pointer without a cast [-Wint-conversion]
   14 |      *(ptr+2) = f1;
      |              ^
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xff80db84,     0xff80db80
In func2
In func1
Segmentation fault (core dumped)
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ vi stack_destroy.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_destroy -m32 -fno-stack-protector -
fno-omit-frame-pointer -fno-pie -no-pie -mpreferred-stack-boundary=2 stack_destr
oy.c
stack_destroy.c: In function 'f2':
stack_destroy.c:15:14: warning: assignment to 'int' from 'void (*)()' makes inte
ger from pointer without a cast [-Wint-conversion]
   15 |      *(ptr+2) = f1;
      |              ^
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xffe1ec54,     0xffe1ec50
In func2
In func1
After invoke f2()
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$
```

(make use of options in gcc 13.3.0)

# Appendix 2

- Stack structure and assembly in gcc 13.3.0
  - ✓ We can make use of options such as –fno-stack-protector, -fno-omit-frame-pointer, -fno-pie, -no-pie, and –mpreferred-stack-boundary=2

```
choijm@embedded: ~/syspro/chap4
choijm@embedded:~/syspro/chap4$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ vi stack_struct.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_struct -m32 stack_struct.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_struct
func2 local:    0xffe90ab0,     0xffe90ab4,     0xffe90ab8
        0xffe90ab0      21
        0xffe90aac      -1504584
        0xffe90abc      1917765888
        y = 112
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ gcc -o stack_struct -m32 -fno-stack-protector -fno-omi
t-frame-pointer -fno-pie -no-pie -mpreferred-stack-boundary=2 stack_struct.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ ./stack_struct
func2 local:    0xffe41fe0,     0xffe41fdc,     0xffe41fd8
        0xffe41fe0      21
        0xffe41fdc      22
        0xffe41fec      111
        y = 333
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ cat stack_struct.c
/* stack_struct.c: stack structure analysis, by choijm. choijm@dku.edu  */
#include <stdio.h>

int func2(int x, int y) {
    int f2_local1 = 21, f2_local2 = 22;
    int *pointer;

    printf("func2 local: \t%p, \t%p, \t%p\n", &f2_local1, &f2_local2, &pointer);
    pointer = &f2_local1;

    printf("\t%p  \t%d\n", (pointer), *(pointer));
    printf("\t%p  \t%d\n", (pointer-1), *(pointer-1));
    printf("\t%p  \t%d\n", (pointer+3), *(pointer+3));

    *(pointer+4) = 333;
```

(revisit stack structure with options)

```
choijm@embedded: ~/syspro/chap4
choijm@embedded:~/syspro/chap4$ gcc -S -fno-stack-protector -fno-omit-frame-pointer -f
no-pie -no-pie -mpreferred-stack-boundary=3 test.c
choijm@embedded:~/syspro/chap4$
choijm@embedded:~/syspro/chap4$ cat test.s
        .file   "test.c"
        .text
        .globl  a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   10
        .globl  b
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   20
        .globl  c
        .bss
        .align 4
        .type   c, @object
        .size   c, 4
c:
        .zero   4
        .section        .rodata
.LC0:
        .string "C = %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    a(%rip), %edx
        movl    b(%rip), %eax
        addl    %edx, %eax
        movl    %eax, c(%rip)
        movl    c(%rip), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0"
        .section        .note.GNU-stack,"",@progbits
```

(revisit assembly code with options)

# 사사

- 본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 ‘SW중심대학’의 결과물이라는 출처를 밝혀야 합니다.

IITP 정보통신기획평가원

디지털인재양성단 SW인재팀