# Lecture Note 4.
# Process Structure

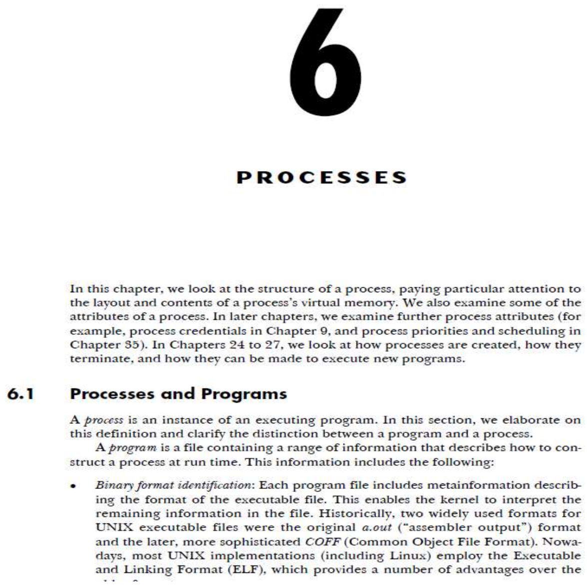
October 9, 2023

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

# 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

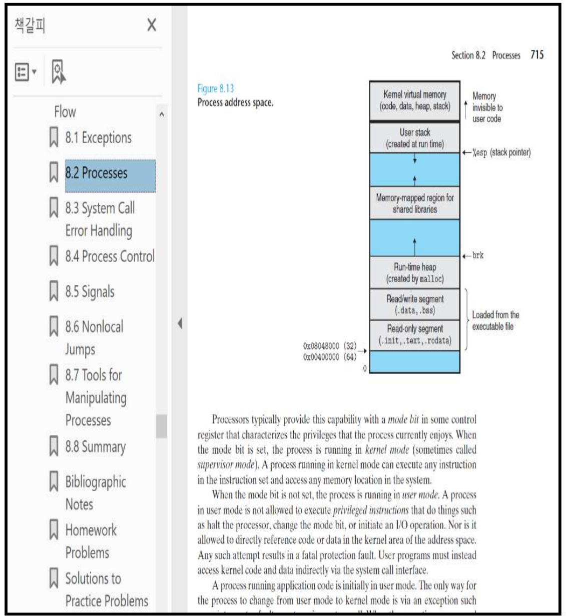
In this chapter, we look at the structure of a process, paying particular attention to the layout and contents of a process's virtual memory. We also examine some of the attributes of a process. In later chapters, we examine further process attributes (for example, process credentials in Chapter 9, and 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
X
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
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

Section 8.2 Processes 715

Figure 8.13
Process address space.

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.

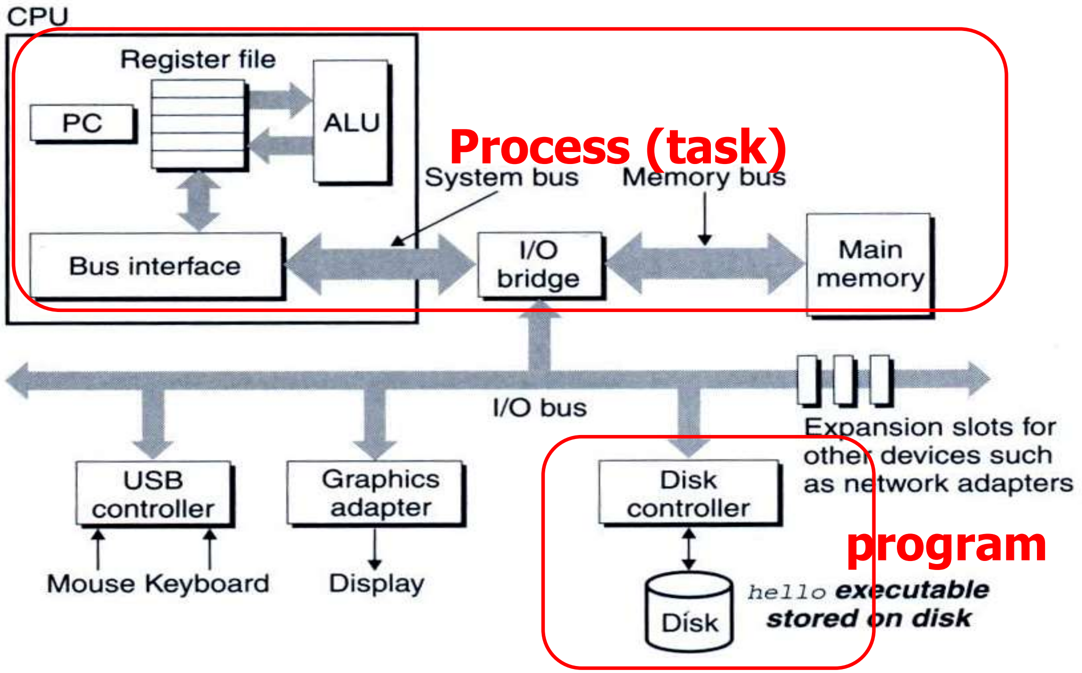
When the mode bit is not set, the process is running in *user mode*. A process in user mode is not allowed to execute *privileged instructions* that do things such as halt the processor, change the mode bit, or initiate an I/O operation. Nor is it allowed to directly reference code or data in the kernel 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.

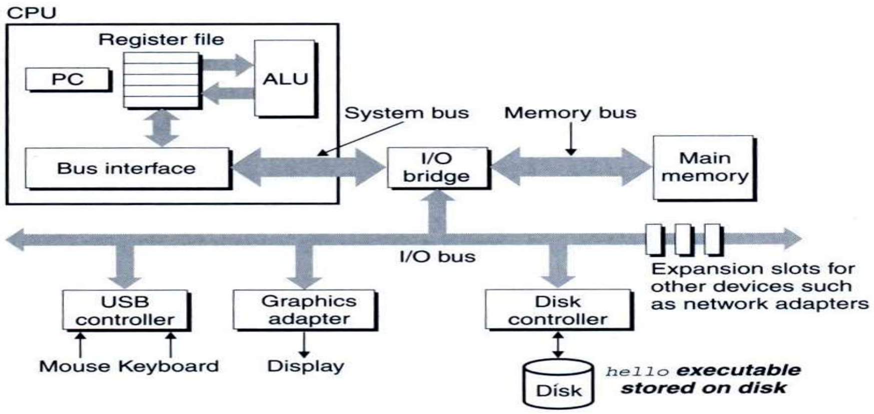


(Source: CSAPP)

# 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

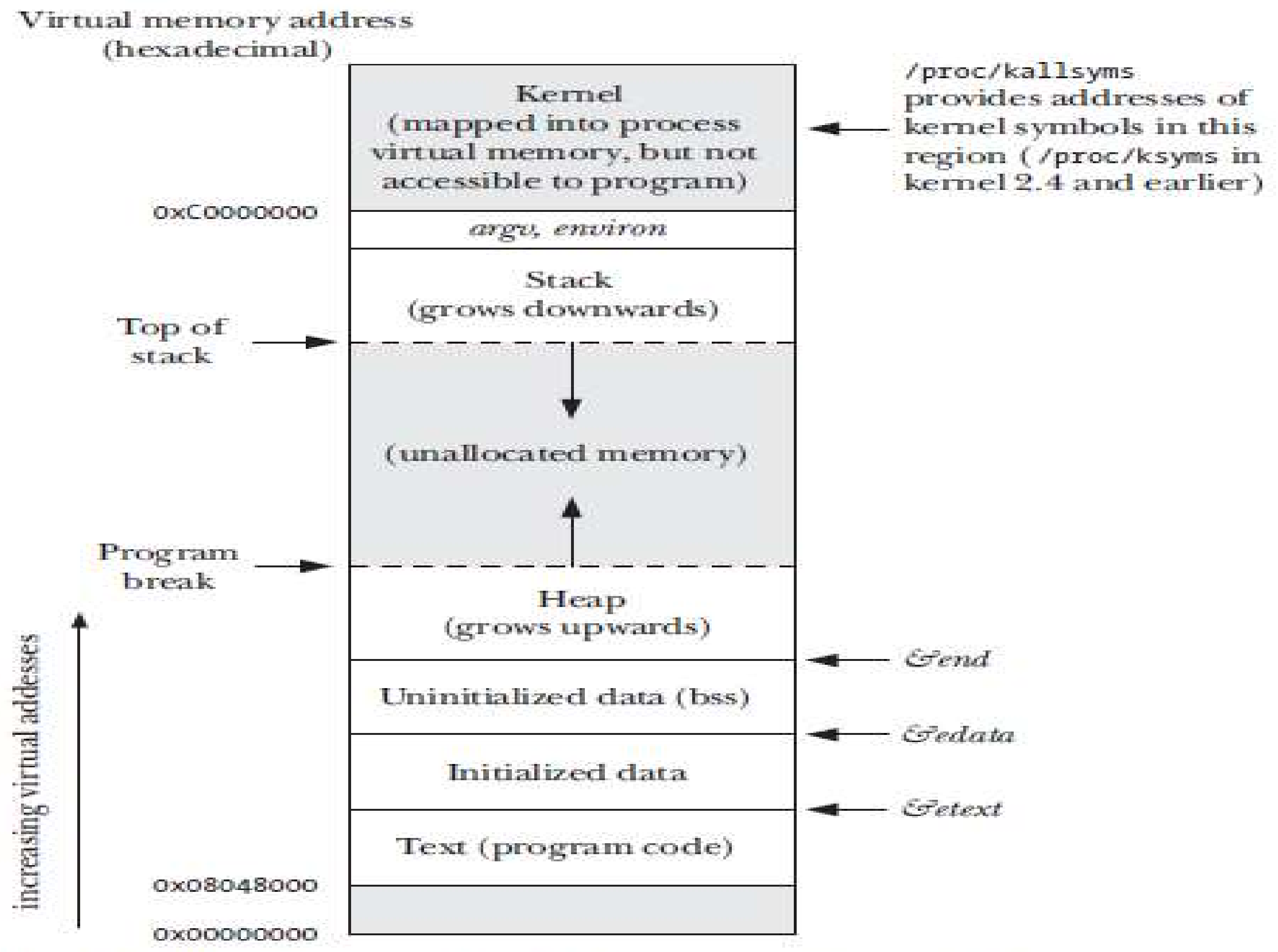

Virtual memory address (hexadecimal)
Kernel (mapped into process virtual memory, but not accessible to program)
/proc/kallsyms provides addresses of kernel symbols in this region (/proc/ksyms in kernel 2.4 and earlier)
0xC0000000
argv, environ
Stack (grows downwards)
Top of stack
(unallocated memory)
Program break
Heap (grows upwards)
&end
Uninitialized data (bss)
&edata
Initialized data
&etext
Text (program code)
increasing virtual addesses
0x08048000
0x00000000

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);
}

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");
}
```

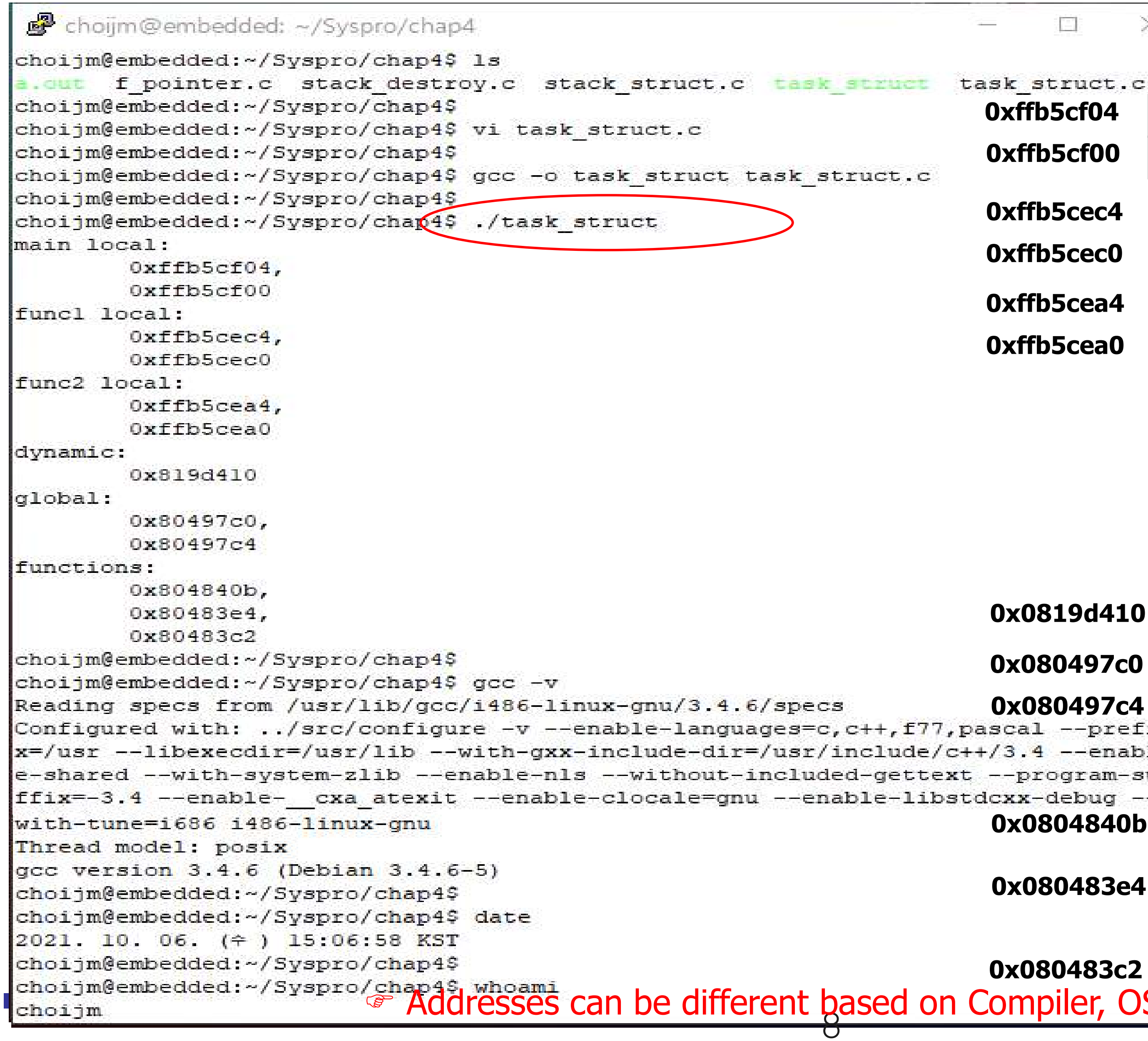
# Process Structure (3/6)

- Process structure in C program: address printing

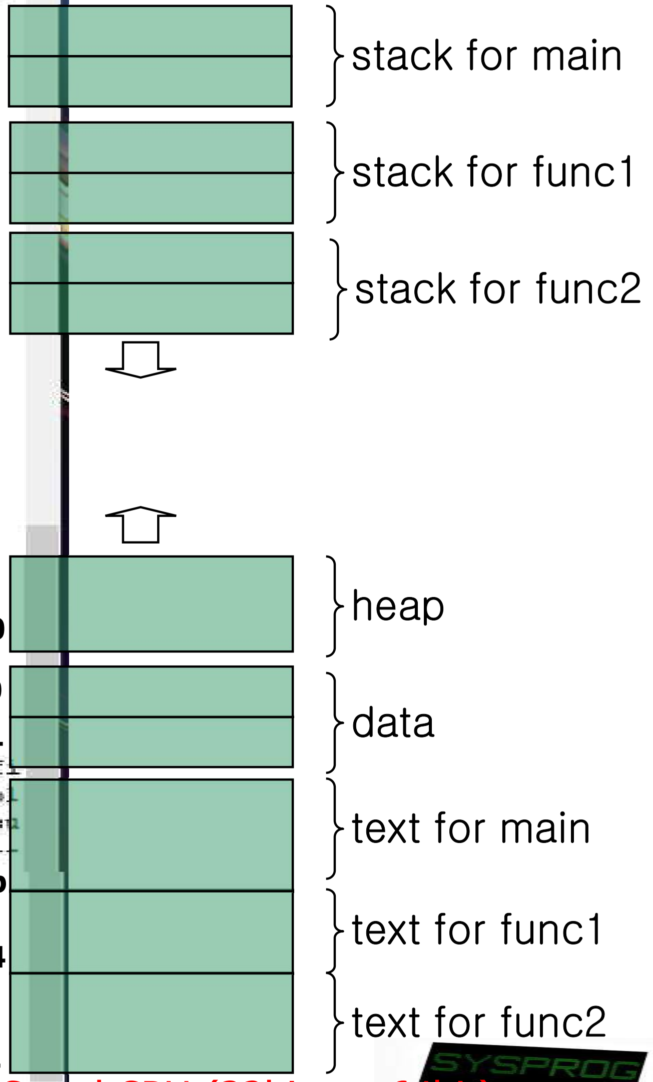
```
/* task_struct.c: display addresses of variables and functions, choijm@dku.edu  */
#include <stdlib.h>
#include <stdio.h>

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();
}

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", &glob1, &glob2);
    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@embedded:~/Syspro/chap4$ ls
a.out  f_pointer.c  stack_destroy.c  stack_struct.c  task_struct  task_struct.c
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ vi task_struct.c
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ gcc -o task_struct task_struct.c
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ ./task_struct
main local:
        0xffb5cf04,
        0xffb5cf00
func1 local:
        0xffb5cec4,
        0xffb5cec0
func2 local:
        0xffb5cea4,
        0xffb5cea0
dynamic:
        0x819d410
global:
        0x80497c0,
        0x80497c4
functions:
        0x804840b,
        0x80483e4,
        0x80483c2
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ gcc -v
Reading specs from /usr/lib/gcc/i486-linux-gnu/3.4.6/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 --
with-tune=i686 i486-linux-gnu
Thread model: posix
gcc version 3.4.6 (Debian 3.4.6-5)
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ date
2021. 10. 06. (수) 15:06:58 KST
choijm@embedded:~/Syspro/chap4$
choijm@embedded:~/Syspro/chap4$ whoami
choijm
```



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

# Process Structure (5/6)

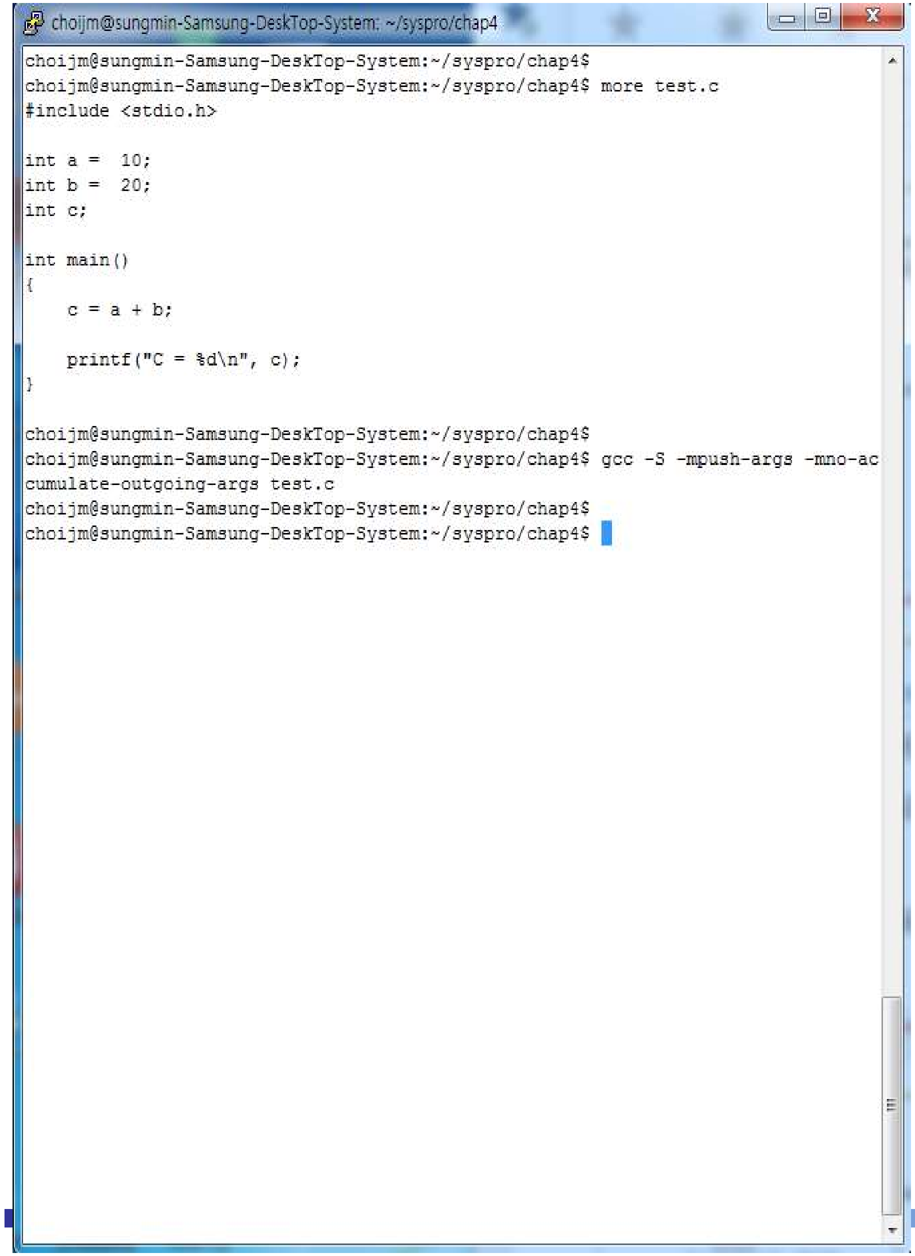
- Summary
  - ✓ Process: consist of four regions, text, data, stack and heap

    **Also called as segment or vm_object**

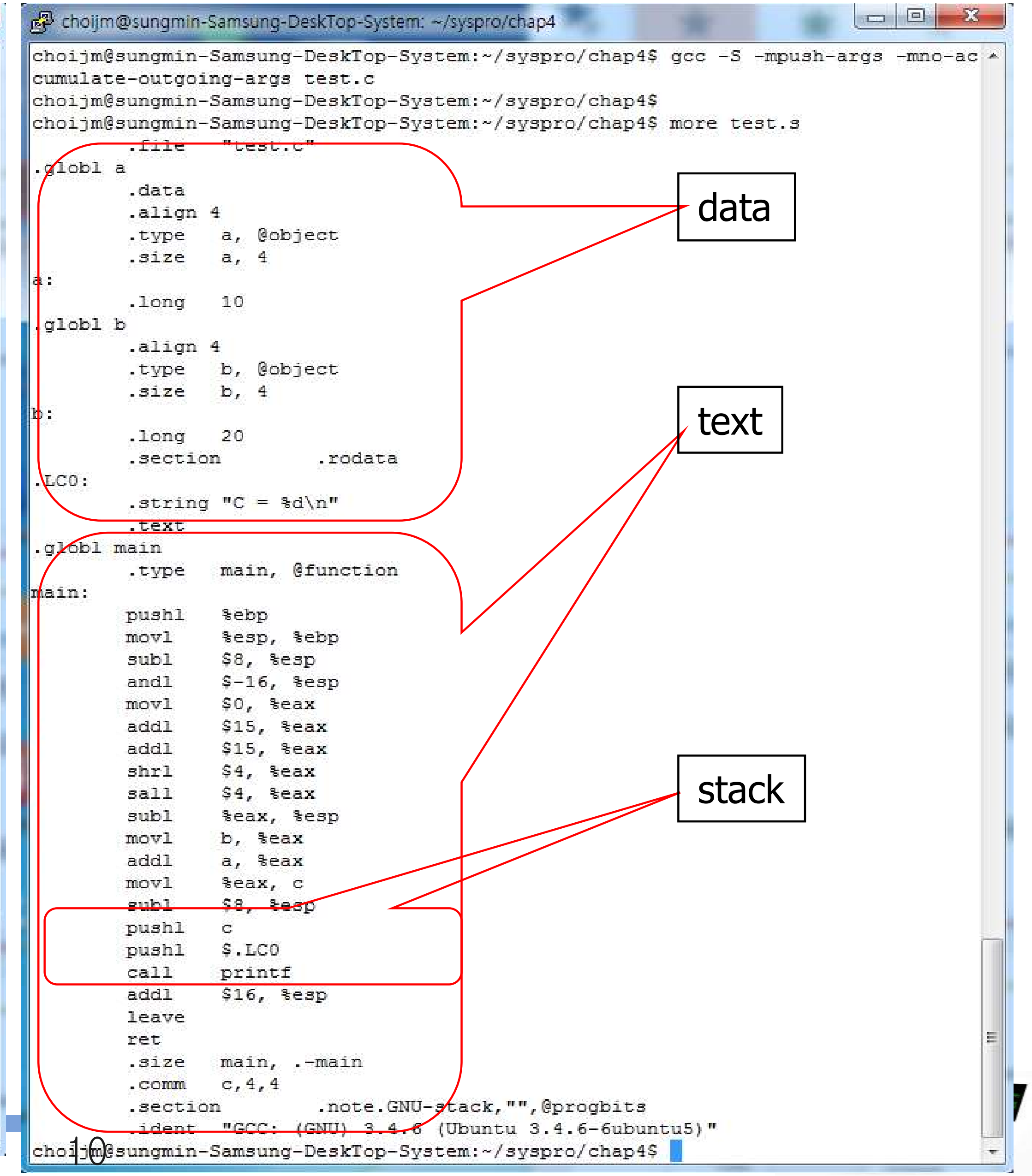
  - ✓ Text
    - Program code (assembly language)
    - Go up to the higher address according to coding order
  - ✓ Data
    - Global variable
    - Initialized and uninitialized data are managed separately (for the performance reason)
  - ✓ Stack
    - Local variable, argument, return address
    - Go 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@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/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@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ gcc -S -mpush-args -mno-ac
cumulate-outgoing-args test.c
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
```



```
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ gcc -S -mpush-args -mno-ac
cumulate-outgoing-args test.c
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/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    $16, %esp
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Ubuntu 3.4.6-6ubuntu5)"
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
```

# Process Structure in CSAPP

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

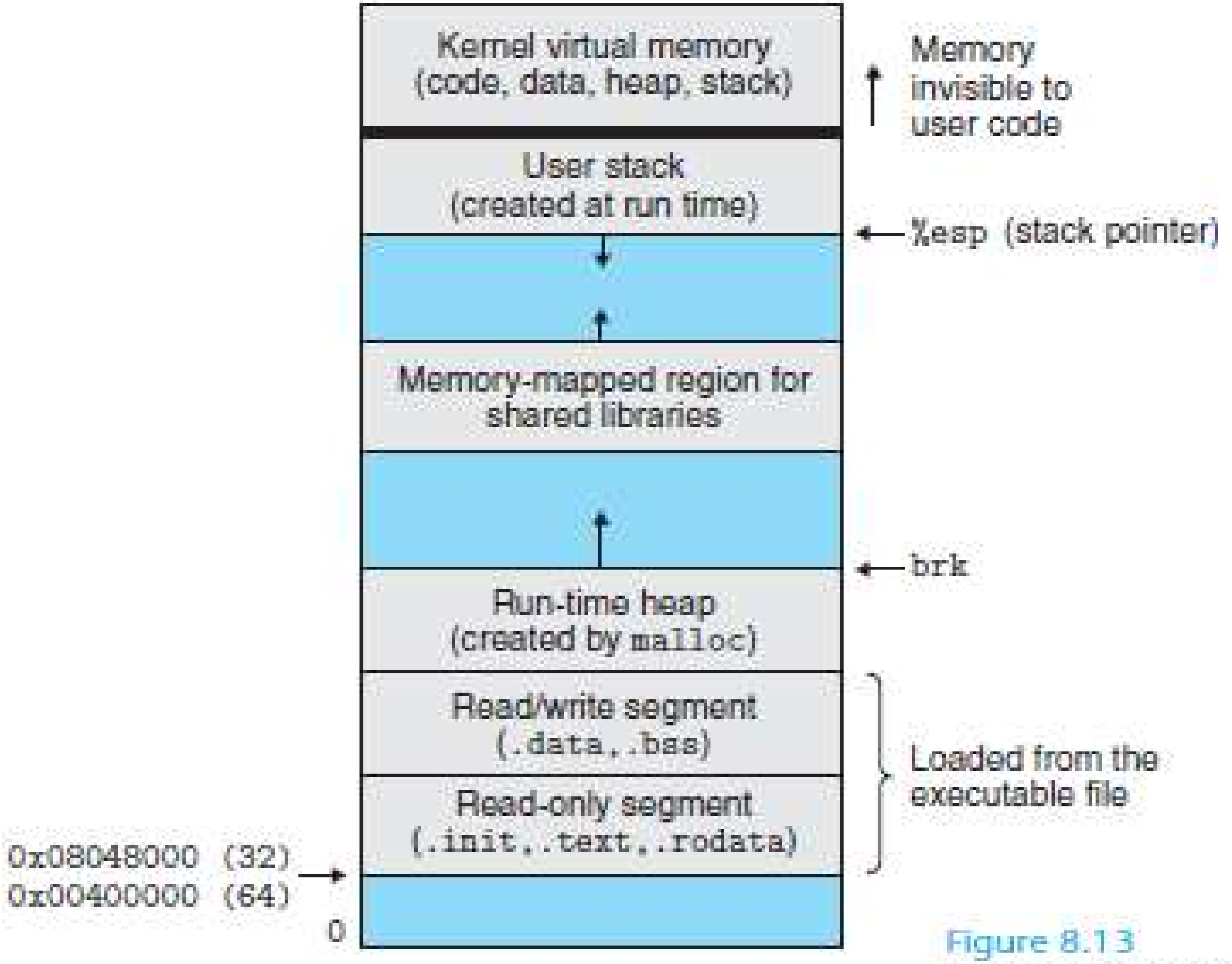
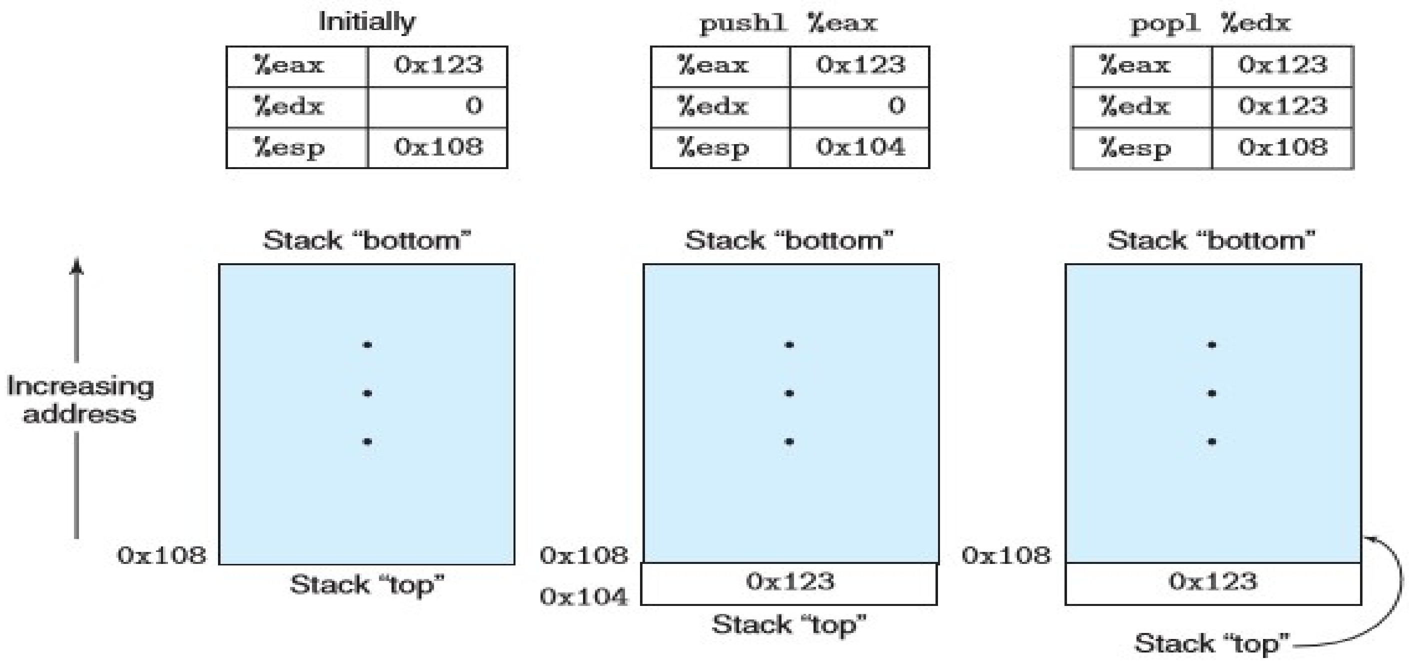

Kernel virtual memory (code, data, heap, stack)
Memory invisible to user code
User stack (created at run time)
%esp (stack pointer)
Memory-mapped region for shared libraries
brk
Run-time heap (created by malloc)
Read/write segment (.data, .bss)
Read-only segment (.init, .text, .rodata)
Loaded from the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

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)



Initially

| | |
|---|---|
| %eax | 0x123 |
| %edx | 0 |
| %esp | 0x108 |

pushl %eax

| | |
|---|---|
| %eax | 0x123 |
| %edx | 0 |
| %esp | 0x104 |

popl %edx

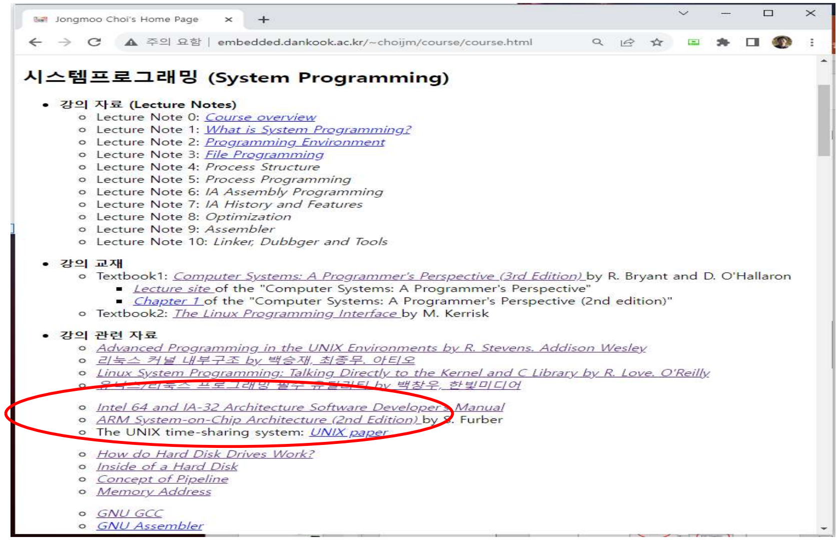
| | |
|---|---|
| %eax | 0x123 |
| %edx | 0x123 |
| %esp | 0x108 |

**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)**

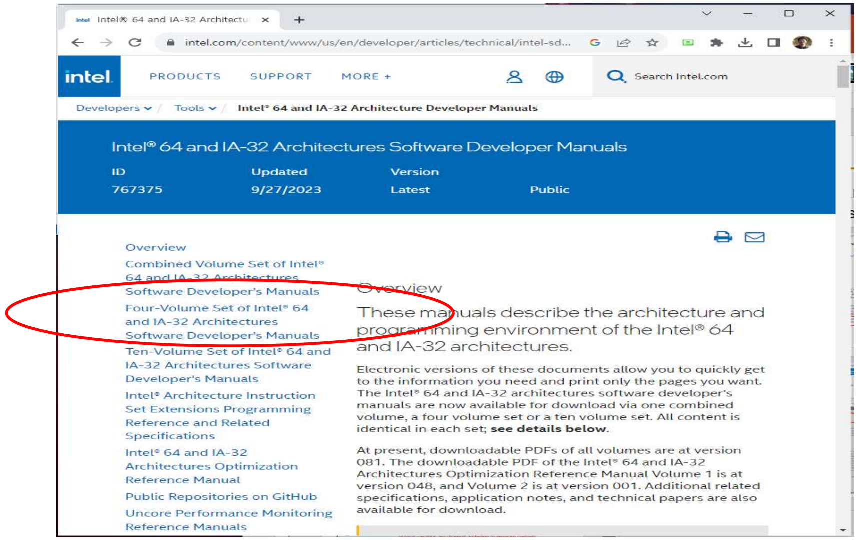
# Stack Details (2/6)

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


Jongmoo Choi's Home Page
주의 요함 | embedded.dankook.ac.kr/~choijm/course/course.html

## 시스템프로그래밍 (System Programming)

- 강의 자료 (Lecture Notes)
  - Lecture Note 0: *Course overview*
  - Lecture Note 1: *What is System Programming?*
  - Lecture Note 2: *Programming Environment*
  - Lecture Note 3: *File Programming*
  - Lecture Note 4: *Process Structure*
  - Lecture Note 5: *Process Programming*
  - Lecture Note 6: *IA Assembly Programming*
  - Lecture Note 7: *IA History and Features*
  - Lecture Note 8: *Optimization*
  - Lecture Note 9: *Assembler*
  - Lecture Note 10: *Linker, Dubbger and Tools*
- 강의 교재
  - Textbook1: *Computer Systems: A Programmer's Perspective (3rd Edition)* by R. Bryant and D. O'Hallaron
    - *Lecture site* of the "Computer Systems: A Programmer's Perspective"
    - *Chapter 1* of the "Computer Systems: A Programmer's Perspective (2nd edition)"
  - Textbook2: *The Linux Programming Interface* by M. Kerrisk
- 강의 관련 자료
  - *Advanced Programming in the UNIX Environments by R. Stevens. Addison Wesley*
  - *리눅스 커널 내부구조 by 백승재, 최종무. 아티오*
  - *Linux System Programming: Talking Directly to the Kernel and C Library by R. Love. O'Reilly*
  - *유닉스/리눅스 프로그래밍 필수 유틸리티 by 백창우, 한빛미디어*
  - *Intel 64 and IA-32 Architecture Software Developer's Manual*
  - *ARM System-on-Chip Architecture (2nd Edition)* by S. Furber
  - The UNIX time-sharing system: *UNIX paper*
  - *How do Hard Disk Drives Work?*
  - *Inside of a Hard Disk*
  - *Concept of Pipeline*
  - *Memory Address*
  - *GNU GCC*
  - *GNU Assembler*


Intel® 64 and IA-32 Architectu
intel.com/content/www/us/en/developer/articles/technical/intel-sd...
intel
PRODUCTS
SUPPORT
MORE +
Search Intel.com

Developers / Tools / **Intel® 64 and IA-32 Architecture Developer Manuals**

Intel® 64 and IA-32 Architectures Software Developer Manuals

| ID | Updated | Version | |
|---|---|---|---|
| 767375 | 9/27/2023 | Latest | Public |

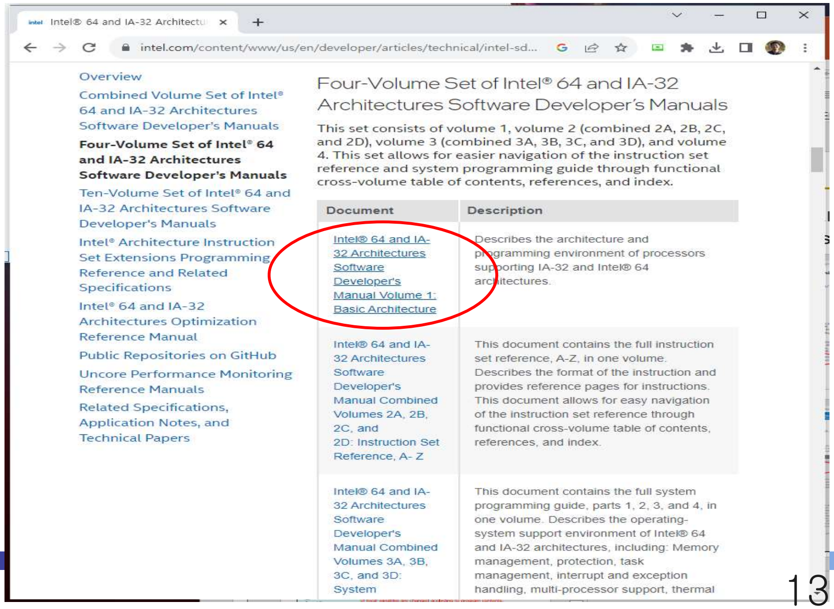
- Overview
- Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals
- Four-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals
- Ten-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals
- Intel® Architecture Instruction Set Extensions Programming Reference and Related Specifications
- Intel® 64 and IA-32 Architectures Optimization Reference Manual
- Public Repositories on GitHub
- Uncore Performance Monitoring Reference Manuals

Overview

These manuals describe the architecture and programming environment of the Intel® 64 and IA-32 architectures.

Electronic versions of these documents allow you to quickly get to the information you need and print only the pages you want. The Intel® 64 and IA-32 architectures software developer's manuals are now available for download via one combined volume, a four volume set or a ten volume set. All content is identical in each set; **see details below.**

At present, downloadable PDFs of all volumes are at version 081. The downloadable PDF of the Intel® 64 and IA-32 Architectures Optimization Reference Manual Volume 1 is at version 048, and Volume 2 is at version 001. Additional related specifications, application notes, and technical papers are also available for download.

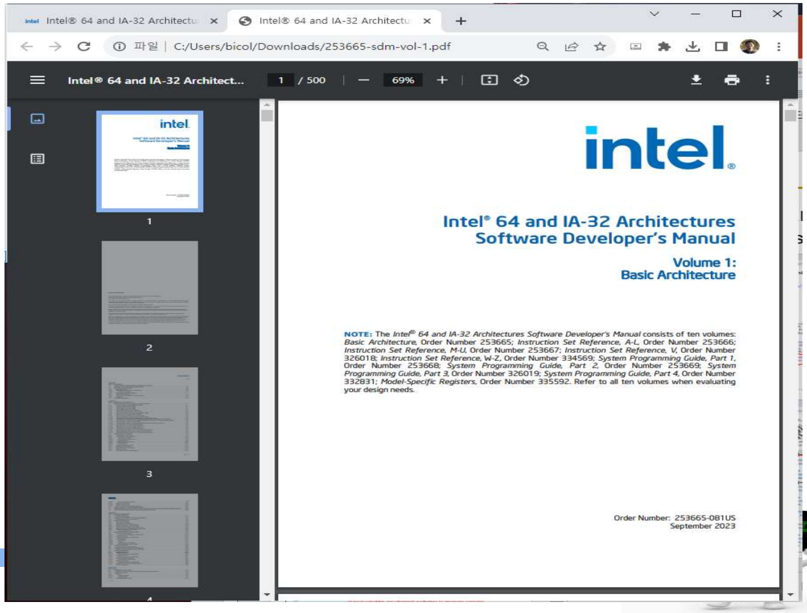

Intel® 64 and IA-32 Architectu
intel.com/content/www/us/en/developer/articles/technical/intel-sd...

- Overview
- Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals
- **Four-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals**
- Ten-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals
- Intel® Architecture Instruction Set Extensions Programming Reference and Related Specifications
- Intel® 64 and IA-32 Architectures Optimization Reference Manual
- Public Repositories on GitHub
- Uncore Performance Monitoring Reference Manuals
- Related Specifications, Application Notes, and Technical Papers

Four-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals

This set consists of volume 1, volume 2 (combined 2A, 2B, 2C, and 2D), volume 3 (combined 3A, 3B, 3C, and 3D), and volume 4. This set allows for easier navigation of the instruction set reference and system programming guide through functional cross-volume table of contents, references, and index.

| Document | Description |
|---|---|
| Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture | Describes the architecture and programming environment of processors supporting IA-32 and Intel® 64 architectures. |
| Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 2A, 2B, 2C, and 2D: Instruction Set Reference, A- Z | This document contains the full instruction set reference, A-Z, in one volume. Describes the format of the instruction and provides reference pages for instructions. This document allows for easy navigation of the instruction set reference through functional cross-volume table of contents, references, and index. |
| Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, 3C, and 3D: System | This document contains the full system programming guide, parts 1, 2, 3, and 4, in one volume. Describes the operating-system support environment of Intel® 64 and IA-32 architectures, including: Memory management, protection, task management, interrupt and exception handling, multi-processor support, thermal |


Intel® 64 and IA-32 Architectu
Intel® 64 and IA-32 Architectu
파일 | C:/Users/bicol/Downloads/253665-sdm-vol-1.pdf
Intel® 64 and IA-32 Architect...
1 / 500
69%
1
2
3
intel

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 1: Basic Architecture

**NOTE:** The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of ten volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference, A-L*, Order Number 253666; *Instruction Set Reference, M-U*, Order Number 253667; *Instruction Set Reference, V*, Order Number 326018; *Instruction Set Reference, W-Z*, Order Number 334569; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019; *System Programming Guide, Part 4*, Order Number 332831; *Model-Specific Registers*, Order Number 335592. Refer to all ten volumes when evaluating your design needs.

Order Number: 253665-081US
September 2023

# 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")

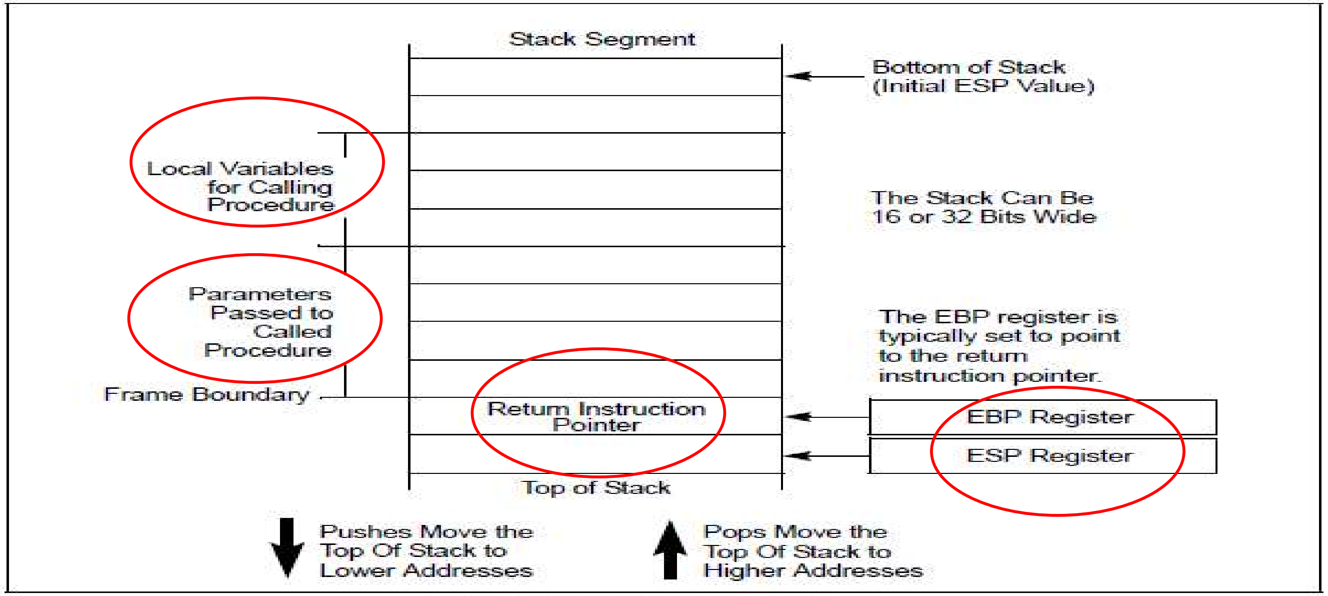

Stack Segment
Bottom of Stack (Initial ESP Value)
Local Variables for Calling Procedure
The Stack Can Be 16 or 32 Bits Wide
Parameters Passed to Called Procedure
The EBP register is typically set to point to the return instruction pointer.
Frame Boundary
Return Instruction Pointer
EBP Register
ESP Register
Top of Stack
Pushes Move the Top Of Stack to Lower Addresses
Pops Move the Top Of Stack to Higher Addresses

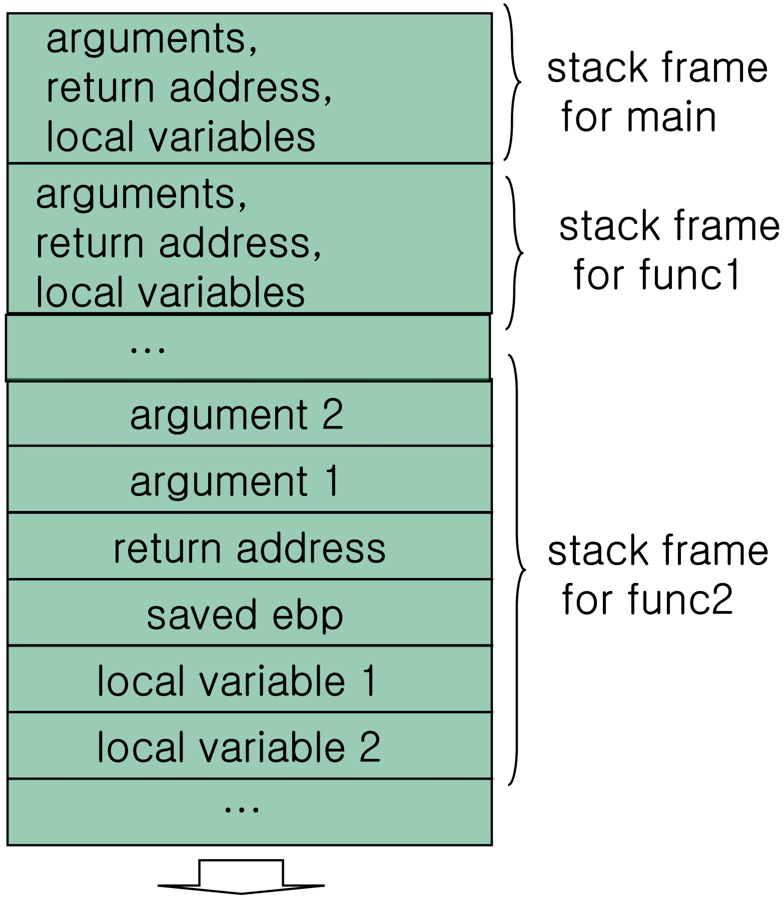
Figure 6-1. Stack Structure

(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();
    ...
}
```



arguments,
return address,
local variables

stack frame for main

arguments,
return address,
local variables

stack frame for func1

…

argument 2

argument 1

return address

saved ebp

local variable 1

local variable 2

…

stack frame for func2

- ☞ Compiler (and version) dependent (see **Appendix 1**)
- ☞ 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)

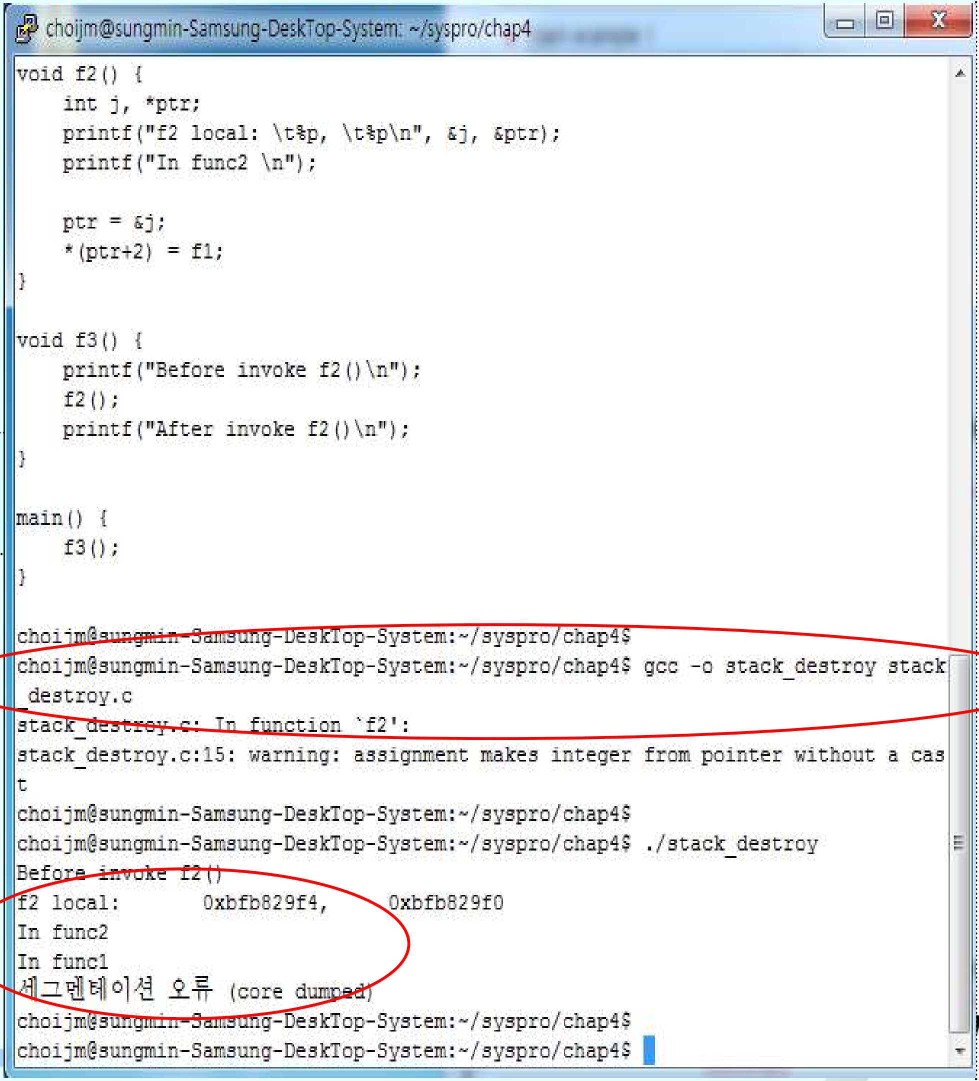
- Stack example 1

```
/* 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);
}

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();
}
```



```
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@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ gcc -o stack_destroy stack
_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:15: warning: assignment makes integer from pointer without a cas
t
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$ ./stack_destroy
Before invoke f2()
f2 local:       0xbfb829f4,     0xbfb829f0
In func2
In func1
세그멘테이션 오류 (core dumped)
choijm@sungmin-Samsung-DeskTop-System:~/syspro/chap4$
choijm@sungmin-Samsung-DeskTop-System:~/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)**

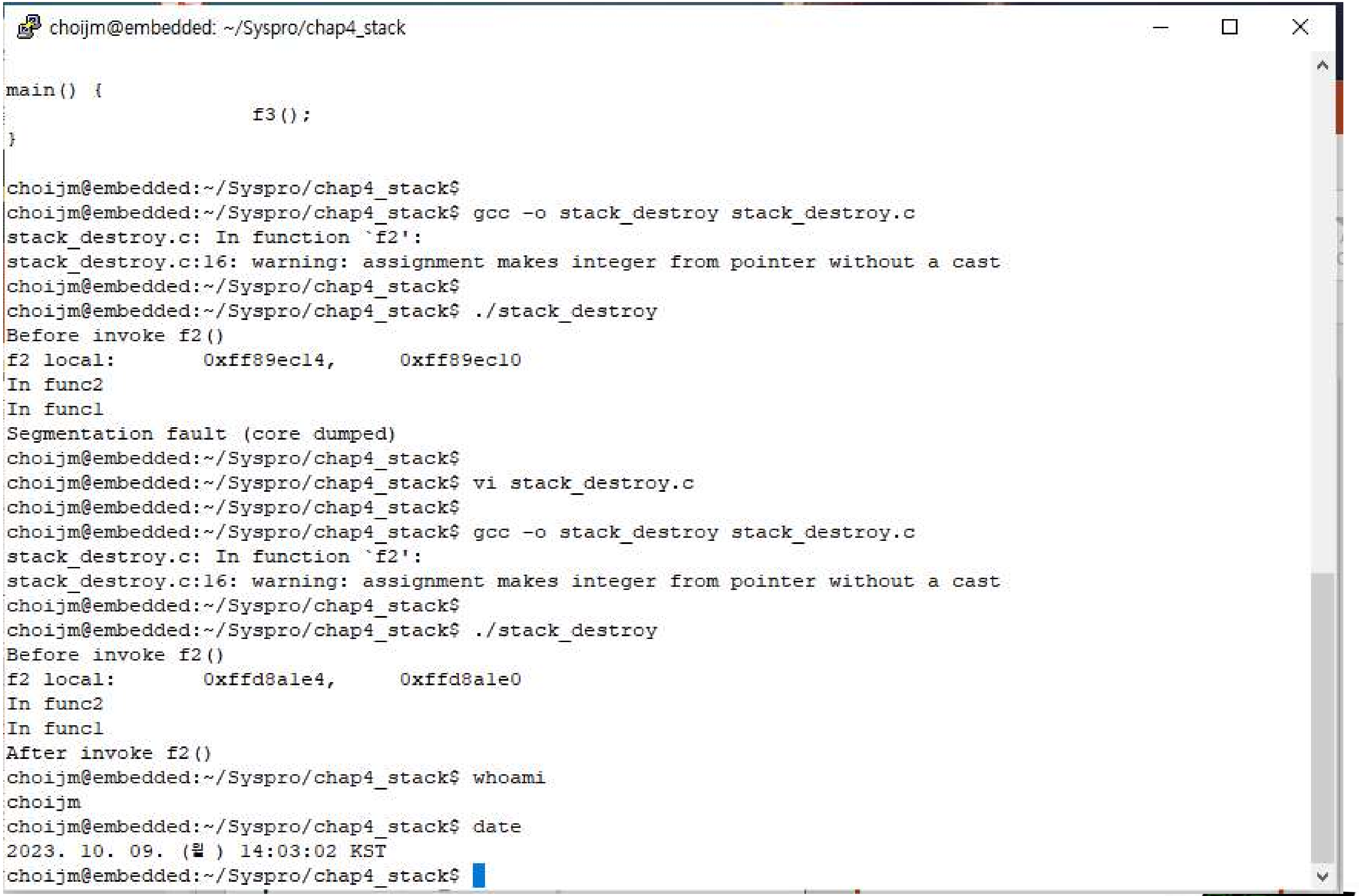
**- discuss why the segmentation fault occurs in this program**

**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 mgchoi@dankook.ac.kr**

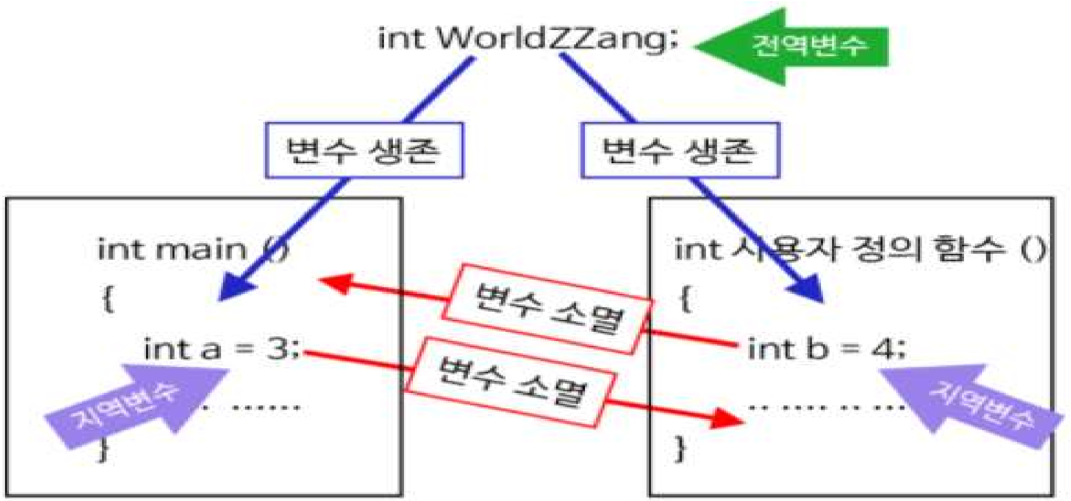
# Homework 4: Snapshot example



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

choijm@embedded:~/Syspro/chap4_stack$
choijm@embedded:~/Syspro/chap4_stack$ gcc -o stack_destroy stack_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:16: warning: assignment makes integer from pointer without a cast
choijm@embedded:~/Syspro/chap4_stack$
choijm@embedded:~/Syspro/chap4_stack$ ./stack_destroy
Before invoke f2()
f2 local:       0xff89ec14,     0xff89ec10
In func2
In func1
Segmentation fault (core dumped)
choijm@embedded:~/Syspro/chap4_stack$
choijm@embedded:~/Syspro/chap4_stack$ vi stack_destroy.c
choijm@embedded:~/Syspro/chap4_stack$
choijm@embedded:~/Syspro/chap4_stack$ gcc -o stack_destroy stack_destroy.c
stack_destroy.c: In function `f2':
stack_destroy.c:16: warning: assignment makes integer from pointer without a cast
choijm@embedded:~/Syspro/chap4_stack$
choijm@embedded:~/Syspro/chap4_stack$ ./stack_destroy
Before invoke f2()
f2 local:       0xffd8a1e4,     0xffd8a1e0
In func2
In func1
After invoke f2()
choijm@embedded:~/Syspro/chap4_stack$ whoami
choijm
choijm@embedded:~/Syspro/chap4_stack$ date
2023. 10. 09. (월 ) 14:03:02 KST
choijm@embedded:~/Syspro/chap4_stack$
```

# Quiz for this Lecture

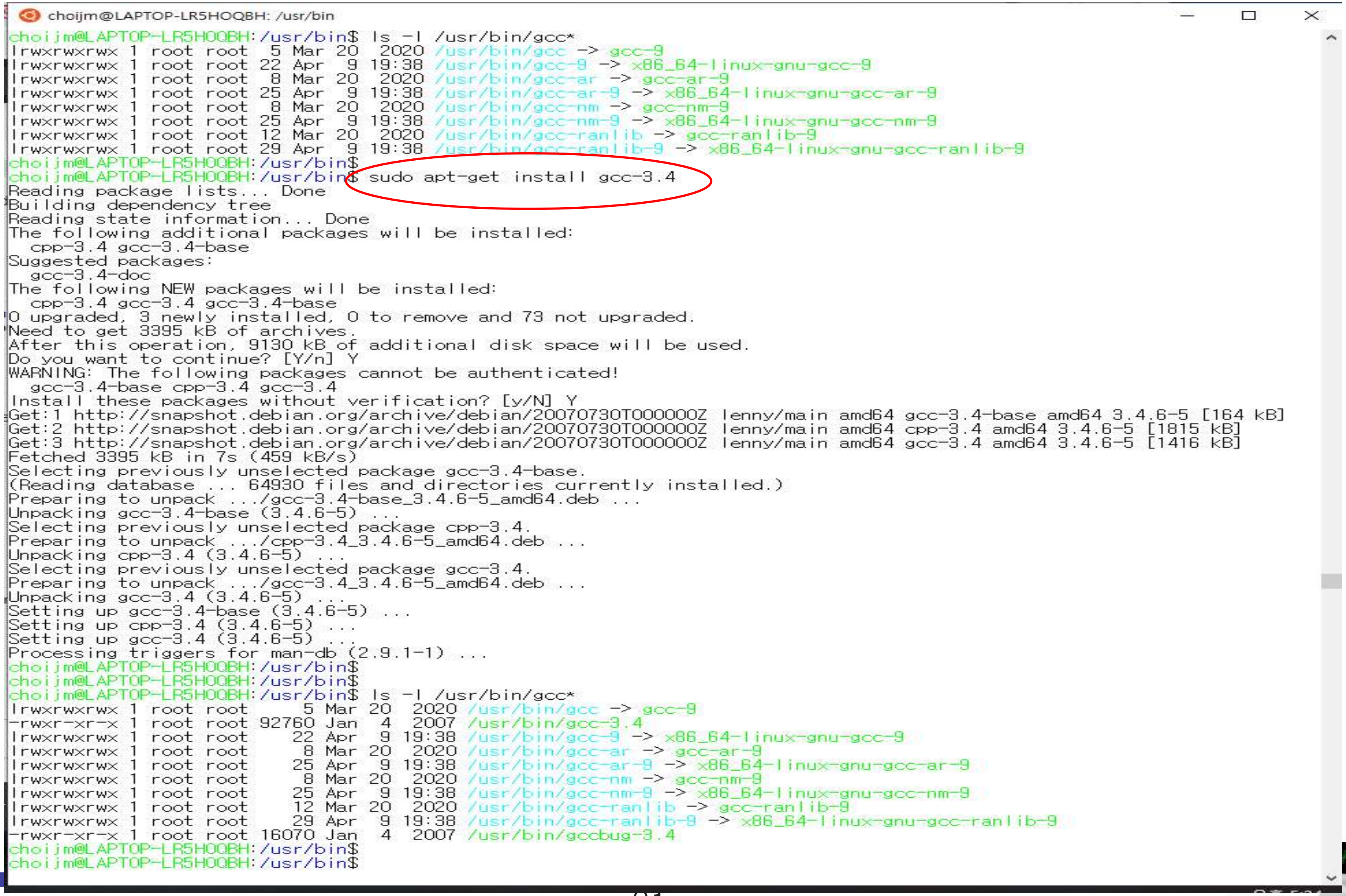
- Quiz
  - ✓ 1. Explain the differences among 1) high-level program, 2) binary program, and 3) process.
  - ✓ 2. In C language, the scope of local variables and global variables are different. Discuss the reason of the differences using the process structure.
  - ✓ 3. Discuss the differences between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



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

# Appendix 1

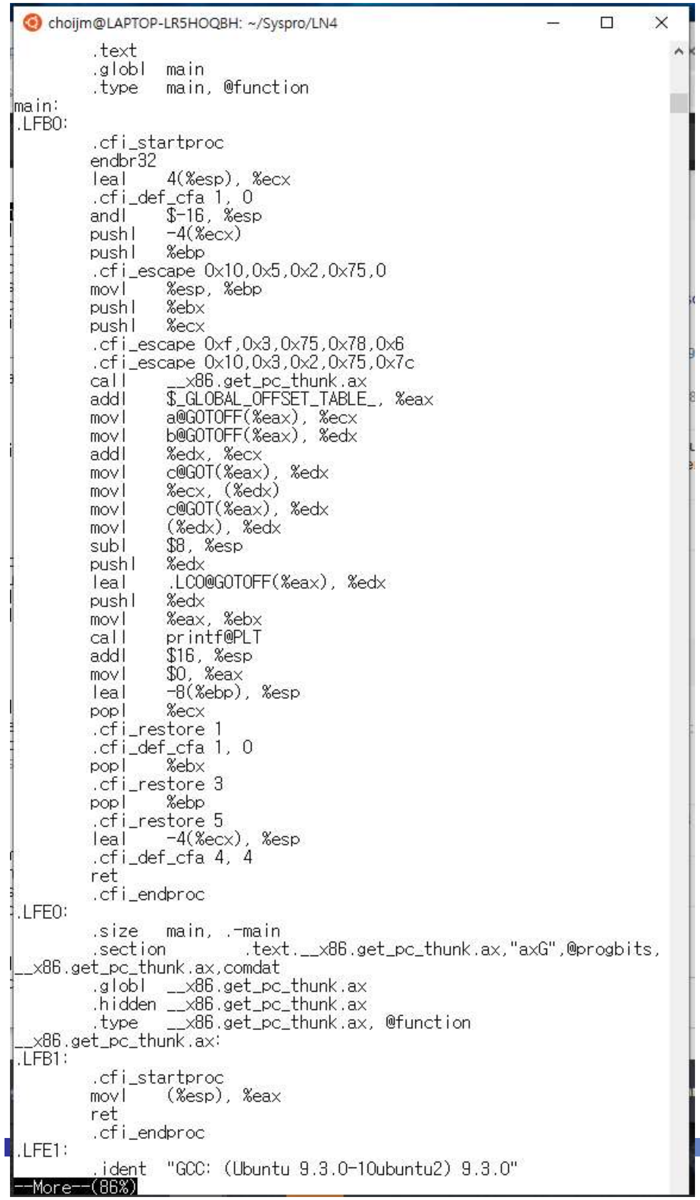
- Assembly differences between gcc 9.* and gcc 3.4.*
  - ✓ Using WSL (Windows subsystem for Linux) in my computer



```
choijm@LAPTOP-LR5HOQBH:/usr/bin$ ls -l /usr/bin/gcc*
lrwxrwxrwx 1 root root  5 Mar 20  2020 /usr/bin/gcc -> gcc-9
lrwxrwxrwx 1 root root 22 Apr  9 19:38 /usr/bin/gcc-9 -> x86_64-linux-gnu-gcc-9
lrwxrwxrwx 1 root root  8 Mar 20  2020 /usr/bin/gcc-ar -> gcc-ar-9
lrwxrwxrwx 1 root root 25 Apr  9 19:38 /usr/bin/gcc-ar-9 -> x86_64-linux-gnu-gcc-ar-9
lrwxrwxrwx 1 root root  8 Mar 20  2020 /usr/bin/gcc-nm -> gcc-nm-9
lrwxrwxrwx 1 root root 25 Apr  9 19:38 /usr/bin/gcc-nm-9 -> x86_64-linux-gnu-gcc-nm-9
lrwxrwxrwx 1 root root 12 Mar 20  2020 /usr/bin/gcc-ranlib -> gcc-ranlib-9
lrwxrwxrwx 1 root root 29 Apr  9 19:38 /usr/bin/gcc-ranlib-9 -> x86_64-linux-gnu-gcc-ranlib-9
choijm@LAPTOP-LR5HOQBH:/usr/bin$
choijm@LAPTOP-LR5HOQBH:/usr/bin$ sudo apt-get install gcc-3.4
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cpp-3.4 gcc-3.4-base
Suggested packages:
  gcc-3.4-doc
The following NEW packages will be installed:
  cpp-3.4 gcc-3.4 gcc-3.4-base
0 upgraded, 3 newly installed, 0 to remove and 73 not upgraded.
Need to get 3395 kB of archives.
After this operation, 9130 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
WARNING: The following packages cannot be authenticated!
  gcc-3.4-base cpp-3.4 gcc-3.4
Install these packages without verification? [y/N] Y
Get:1 http://snapshot.debian.org/archive/debian/20070730T000000Z lenny/main amd64 gcc-3.4-base amd64 3.4.6-5 [164 kB]
Get:2 http://snapshot.debian.org/archive/debian/20070730T000000Z lenny/main amd64 cpp-3.4 amd64 3.4.6-5 [1815 kB]
Get:3 http://snapshot.debian.org/archive/debian/20070730T000000Z lenny/main amd64 gcc-3.4 amd64 3.4.6-5 [1416 kB]
Fetched 3395 kB in 7s (459 kB/s)
Selecting previously unselected package gcc-3.4-base.
(Reading database ... 64930 files and directories currently installed.)
Preparing to unpack .../gcc-3.4-base_3.4.6-5_amd64.deb ...
Unpacking gcc-3.4-base (3.4.6-5) ...
Selecting previously unselected package cpp-3.4.
Preparing to unpack .../cpp-3.4_3.4.6-5_amd64.deb ...
Unpacking cpp-3.4 (3.4.6-5) ...
Selecting previously unselected package gcc-3.4.
Preparing to unpack .../gcc-3.4_3.4.6-5_amd64.deb ...
Unpacking gcc-3.4 (3.4.6-5) ...
Setting up gcc-3.4-base (3.4.6-5) ...
Setting up cpp-3.4 (3.4.6-5) ...
Setting up gcc-3.4 (3.4.6-5) ...
Processing triggers for man-db (2.9.1-1) ...
choijm@LAPTOP-LR5HOQBH:/usr/bin$
choijm@LAPTOP-LR5HOQBH:/usr/bin$
choijm@LAPTOP-LR5HOQBH:/usr/bin$ ls -l /usr/bin/gcc*
lrwxrwxrwx 1 root root     5 Mar 20  2020 /usr/bin/gcc -> gcc-9
-rwxr-xr-x 1 root root 92760 Jan  4  2007 /usr/bin/gcc-3.4
lrwxrwxrwx 1 root root    22 Apr  9 19:38 /usr/bin/gcc-9 -> x86_64-linux-gnu-gcc-9
lrwxrwxrwx 1 root root     8 Mar 20  2020 /usr/bin/gcc-ar -> gcc-ar-9
lrwxrwxrwx 1 root root    25 Apr  9 19:38 /usr/bin/gcc-ar-9 -> x86_64-linux-gnu-gcc-ar-9
lrwxrwxrwx 1 root root     8 Mar 20  2020 /usr/bin/gcc-nm -> gcc-nm-9
lrwxrwxrwx 1 root root    25 Apr  9 19:38 /usr/bin/gcc-nm-9 -> x86_64-linux-gnu-gcc-nm-9
lrwxrwxrwx 1 root root    12 Mar 20  2020 /usr/bin/gcc-ranlib -> gcc-ranlib-9
lrwxrwxrwx 1 root root    29 Apr  9 19:38 /usr/bin/gcc-ranlib-9 -> x86_64-linux-gnu-gcc-ranlib-9
-rwxr-xr-x 1 root root 16070 Jan  4  2007 /usr/bin/gccbug-3.4
choijm@LAPTOP-LR5HOQBH:/usr/bin$
choijm@LAPTOP-LR5HOQBH:/usr/bin$
```

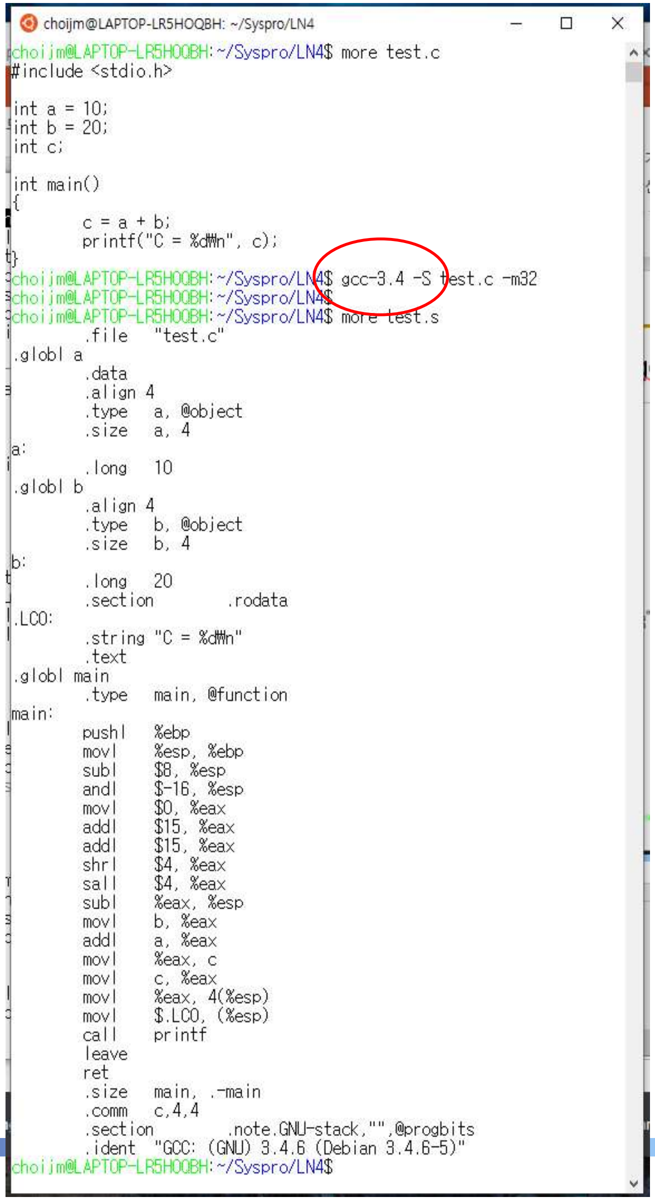
# Appendix 1

- Assembly differences between gcc 9.* and gcc 3.4.*
  - ✓ 1) Obfuscation, 2) Optimization, 3) CFI, …



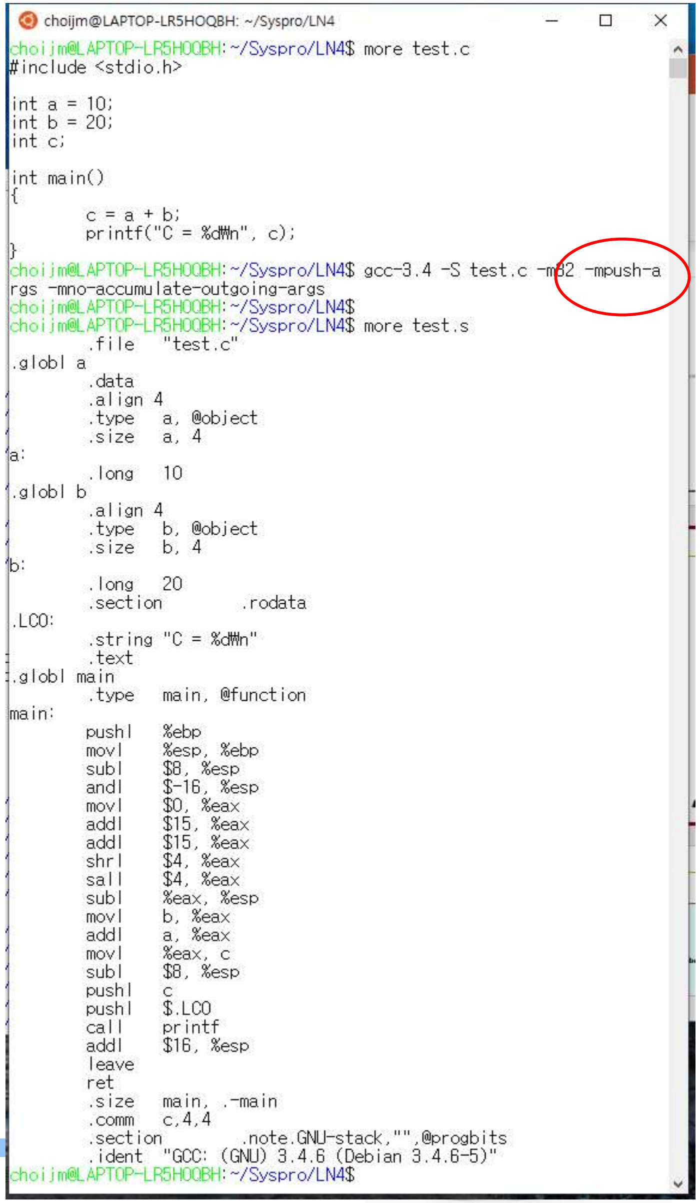
choijm@LAPTOP-LR5HOQBH: ~/Syspro/LN4

```
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr32
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ebx
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x78,0x6
        .cfi_escape 0x10,0x3,0x2,0x75,0x7c
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    a@GOTOFF(%eax), %ecx
        movl    b@GOTOFF(%eax), %edx
        addl    %edx, %ecx
        movl    c@GOT(%eax), %edx
        movl    %ecx, (%edx)
        movl    c@GOT(%eax), %edx
        movl    (%edx), %edx
        subl    $8, %esp
        pushl   %edx
        leal    .LC0@GOTOFF(%eax), %edx
        pushl   %edx
        movl    %eax, %ebx
        call    printf@PLT
        addl    $16, %esp
        movl    $0, %eax
        leal    -8(%ebp), %esp
        popl    %ecx
        .cfi_restore 1
        .cfi_def_cfa 1, 0
        popl    %ebx
        .cfi_restore 3
        popl    %ebp
        .cfi_restore 5
        leal    -4(%ecx), %esp
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .text.__x86.get_pc_thunk.ax,"axG",@progbits,
__x86.get_pc_thunk.ax,comdat
        .globl  __x86.get_pc_thunk.ax
        .hidden __x86.get_pc_thunk.ax
        .type   __x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
.LFB1:
        .cfi_startproc
        movl    (%esp), %eax
        ret
        .cfi_endproc
.LFE1:
        .ident  "GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0"
--More--(86%)
```



choijm@LAPTOP-LR5HOQBH: ~/Syspro/LN4

```
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ 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@LAPTOP-LR5HOQBH:~/Syspro/LN4$ gcc-3.4 -S test.c -m32
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ 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    $.LC0, (%esp)
        call    printf
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Debian 3.4.6-5)"
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
```

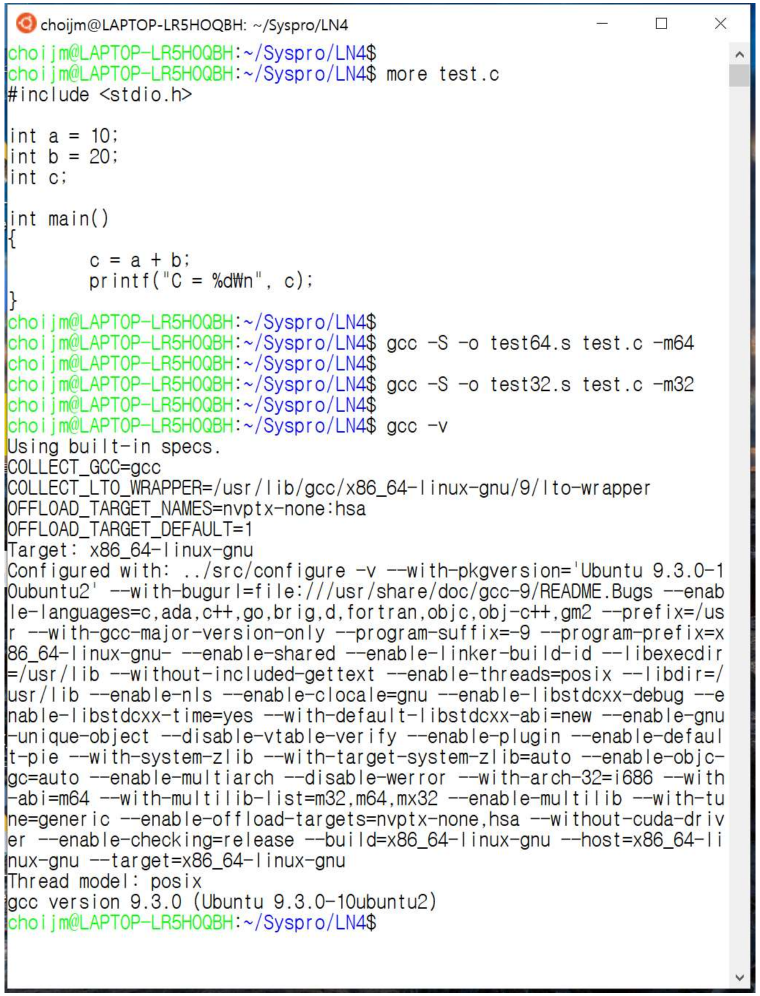


choijm@LAPTOP-LR5HOQBH: ~/Syspro/LN4

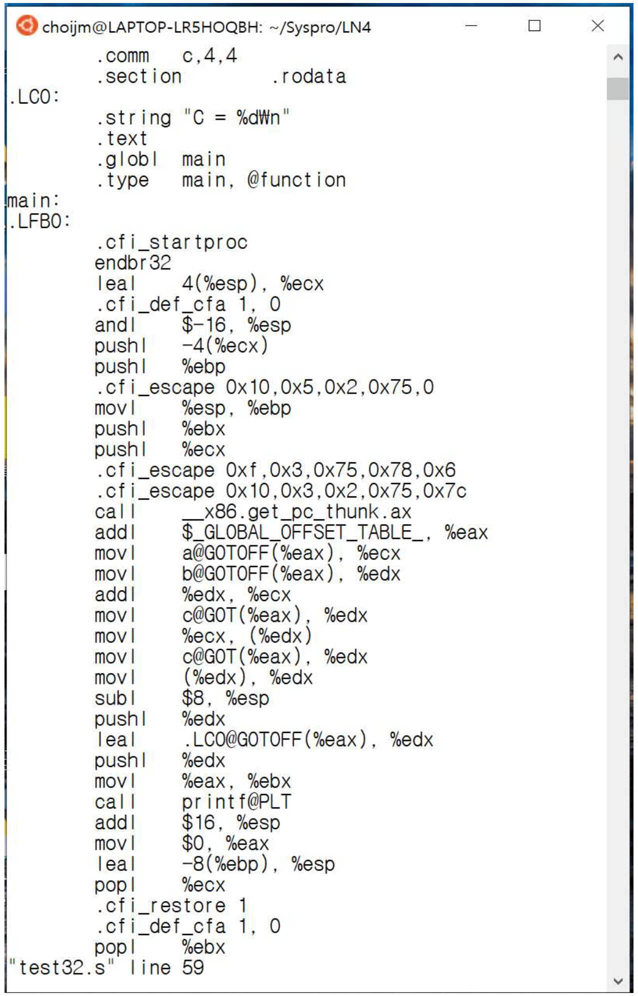
```
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ more test.c
#include <stdio.h>

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

int main()
{
        c = a + b;
        printf("C = %d₩n", c);
}
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ gcc-3.4 -S test.c -m32 -mpush-a
rgs -mno-accumulate-outgoing-args
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ 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    $16, %esp
        leave
        ret
        .size   main, .-main
        .comm   c,4,4
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.4.6 (Debian 3.4.6-5)"
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
```

# Appendix 1

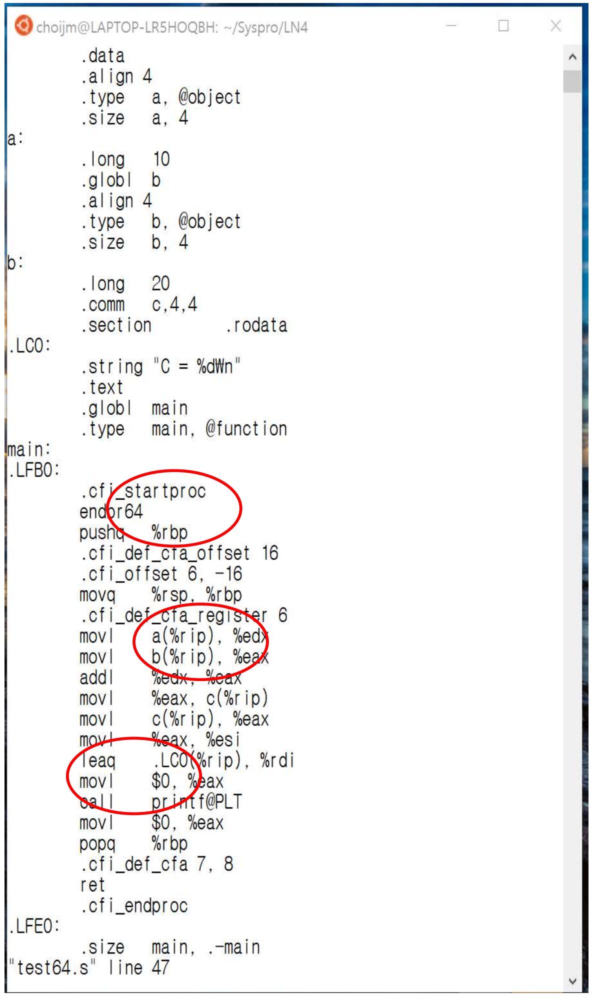
- Assembly differences between 32-bit and 64-bit CPU
  - ✓ 1) Register (eax vs rax), 2) PIC, 3) Argument passing, 4) …
  - ✓ We will discuss further in LN6 and LN9



```
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ 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@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ gcc -S -o test64.s test.c -m64
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ gcc -S -o test32.s test.c -m32
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:hsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 9.3.0-1
0ubuntu2' --with-bugurl=file:///usr/share/doc/gcc-9/README.Bugs --enab
le-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,gm2 --prefix=/us
r --with-gcc-major-version-only --program-suffix=-9 --program-prefix=x
86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir
=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/
usr/lib --enable-nls --enable-clocale=gnu --enable-libstdcxx-debug --e
nable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu
-unique-object --disable-vtable-verify --enable-plugin --enable-defaul
t-pie --with-system-zlib --with-target-system-zlib=auto --enable-objc-
gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with
-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tu
ne=generic --enable-offload-targets=nvptx-none,hsa --without-cuda-driv
er --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-li
nux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
choijm@LAPTOP-LR5HOQBH:~/Syspro/LN4$
```



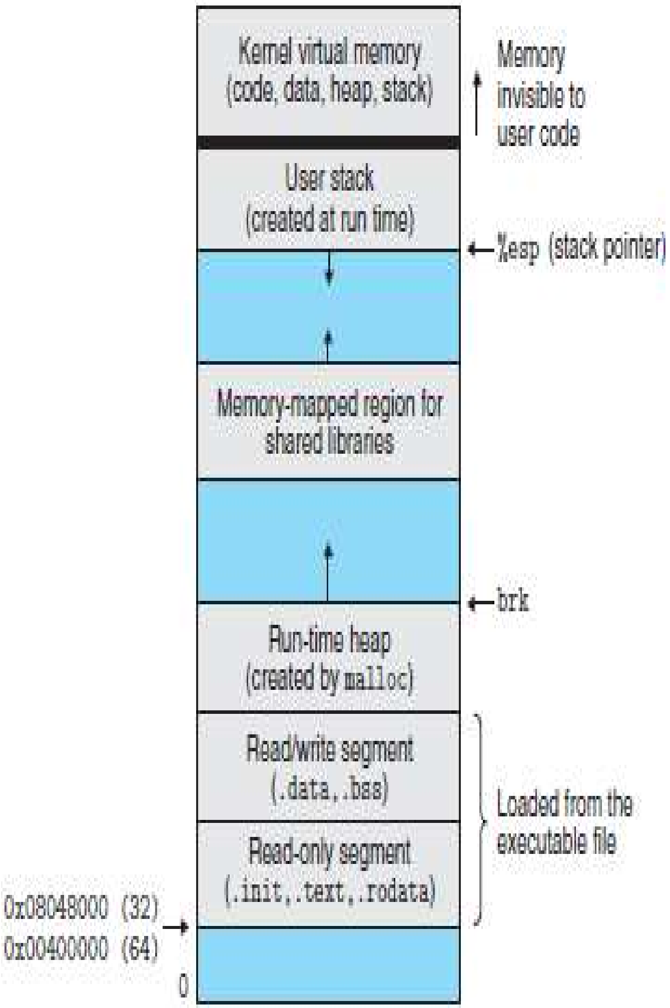
```
        .comm   c,4,4
        .section        .rodata
.LC0:
        .string "C = %d₩n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr32
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ebx
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x78,0x6
        .cfi_escape 0x10,0x3,0x2,0x75,0x7c
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    a@GOTOFF(%eax), %ecx
        movl    b@GOTOFF(%eax), %edx
        addl    %edx, %ecx
        movl    c@GOT(%eax), %edx
        movl    %ecx, (%edx)
        movl    c@GOT(%eax), %edx
        movl    (%edx), %edx
        subl    $8, %esp
        pushl   %edx
        leal    .LC0@GOTOFF(%eax), %edx
        pushl   %edx
        movl    %eax, %ebx
        call    printf@PLT
        addl    $16, %esp
        movl    $0, %eax
        leal    -8(%ebp), %esp
        popl    %ecx
        .cfi_restore 1
        .cfi_def_cfa 1, 0
        popl    %ebx
"test32.s" line 59
```



```
        .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
        .comm   c,4,4
        .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
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
"test64.s" line 47
```

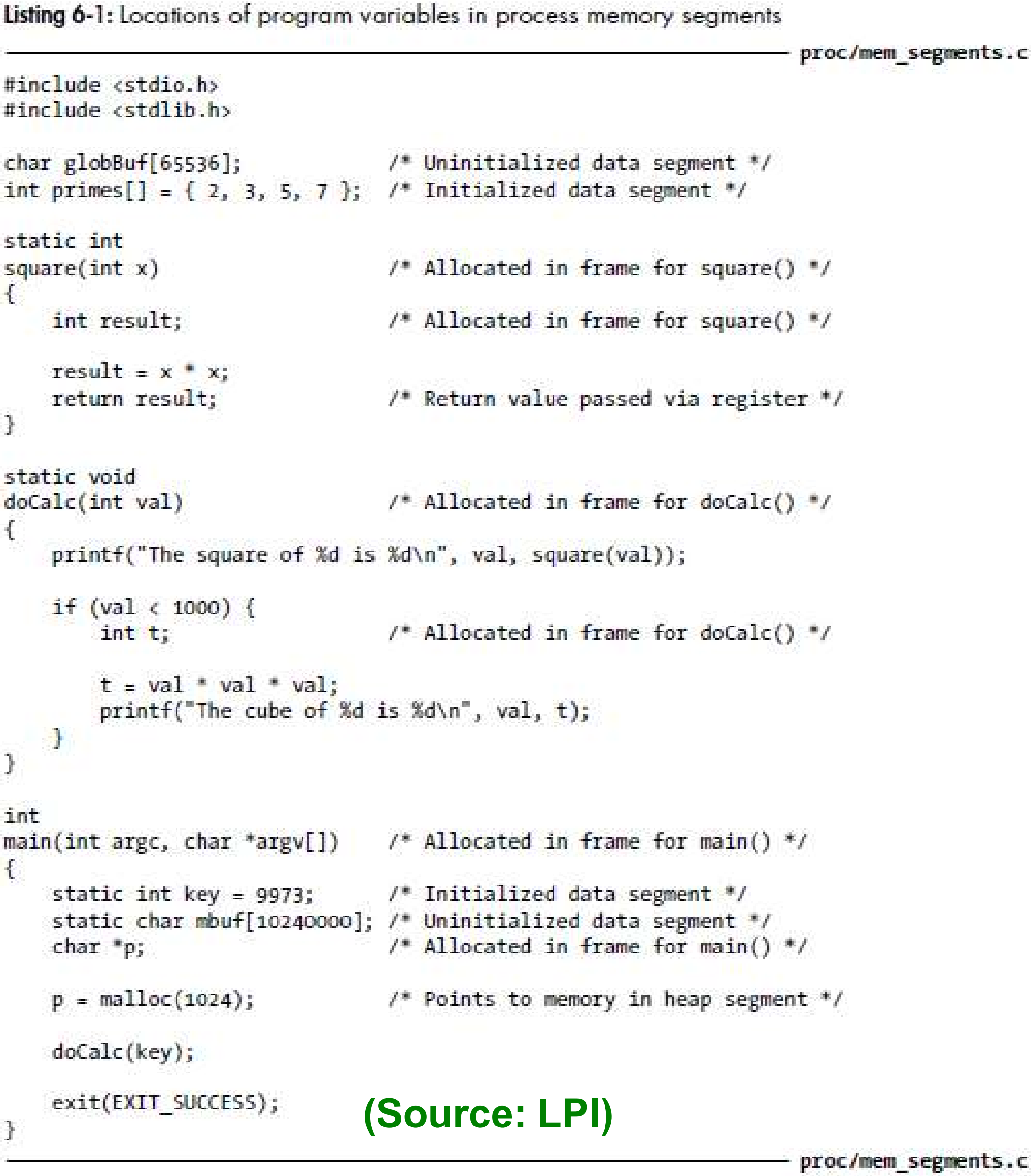
# Appendix 2

- Another code  for process structure



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

**(Source: CSAPP)**



**Listing 6-1:** Locations of program variables in process memory segments

proc/mem_segments.c

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

char globBuf[65536];            /* Uninitialized data segment */
int primes[] = { 2, 3, 5, 7 };  /* Initialized data segment */

static int
square(int x)                   /* Allocated in frame for square() */
{
    int result;                 /* Allocated in frame for square() */

    result = x * x;
    return result;              /* Return value passed via register */
}

static void
doCalc(int val)                 /* Allocated in frame for doCalc() */
{
    printf("The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                  /* Allocated in frame for doCalc() */

        t = val * val * val;
        printf("The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char *argv[])    /* Allocated in frame for main() */
{
    static int key = 9973;      /* Initialized data segment */
    static char mbuf[10240000]; /* Uninitialized data segment */
    char *p;                    /* Allocated in frame for main() */

    p = malloc(1024);           /* Points to memory in heap segment */

    doCalc(key);

    exit(EXIT_SUCCESS);
}
```

proc/mem_segments.c

**(Source: LPI)**