CSGE602055 Operating Systems
CSF2600505 Sistem Operasi
Week 04: Addressing, Shared Lib, & Pointer

Rahmat M. Samik-Ibrahim (ed.)

University of Indonesia

`https://os.vlsm.org/`
Always check for the latest revision!

REV252 12-Oct-2020

Operating Systems 202^3) — **PJJ from HOME**
ZOOM: International [Tue 08-10] — A/Matrix [Tue 10-12]

Week	Schedule & Deadline^1)	Topic	OSC10^2)
Week 00	15 Sep - 21 Sep 2020	Overview 1, Virtualization & Scripting	Ch. 1, 2, 18.
Week 01	22 Sep - 28 Sep 2020	Overview 2, Virtualization & Scripting	Ch. 1, 2, 18.
Week 02	29 Sep - 05 Oct 2020	Security, Protection, Privacy, & C-language.	Ch. 16, 17.
Week 03	06 Oct - 12 Oct 2020	File System & FUSE	Ch. 13, 14, 15.
Week 04	13 Oct - 19 Oct 2020	Addressing, Shared Lib, & Pointer	Ch. 9.
Week 05	20 Oct - 26 Oct 2020	Virtual Memory	Ch. 10.
Week 06	27 Oct - 16 Nov 2020	Concurrency: Processes & Threads	Ch. 3, 4.
	29 Oct 2020	Maulid Nabi	
Week 07	17 Nov - 23 Nov 2020	Synchronization & Deadlock	Ch. 6, 7, 8.
Week 08	24 Nov - 30 Nov 2020	Scheduling + W06/W07	Ch. 5.
Week 09	01 Dec - 07 Dec 2020	Storage, Firmware, Bootloader, & Systemd	Ch. 11.
Week 10	08 Dec - 16 Dec 2020	I/O & Programming	Ch. 12.
	09 Dec 2020	Pil Kada	

---

^{1}) The **DEADLINE** of Week 00 is 21 Sep 2020, whereas the **DEADLINE** of Week 01 is 28 Sep 2020, and so on...

^{2}) Silberschatz et. al.: **Operating System Concepts**, 10th Edition, 2018.

^{3}) This information will be on **EVERY** page two (2) of this course material.

STARTING POINT — `https://os.vlsm.org/`

☐ **Text Book** — Any recent/decent OS book. Eg. (**OSC10**) Silberschatz et. al.: **Operating System Concepts**, 10th Edition, 2018. See also `http://codex.cs.yale.edu/avi/os-book/OS10/`.

☐ **Resources**

    ☐ **SCELE** — `https://scele.cs.ui.ac.id/course/view.php?id=3020`. The enrollment key is **XXX**.

    ☐ **Download Slides and Demos from GitHub.com**
    `https://github.com/UI-FASILKOM-OS/SistemOperasi/`:
    `os00.pdf` (W00), `os01.pdf` (W01), `os02.pdf` (W02), `os03.pdf` (W03),
    `os04.pdf` (W04), `os05.pdf` (W05), `os06.pdf` (W06), `os07.pdf` (W07),
    `os08.pdf` (W08), `os09.pdf` (W09), `os10.pdf` (W10).

    ☐ **Problems** — `https://rms46.vlsm.org/2/`:
    `195.pdf` (W00), `196.pdf` (W01), `197.pdf` (W02), `198.pdf` (W03),
    `199.pdf` (W04), `200.pdf` (W05), `201.pdf` (W06), `202.pdf` (W07),
    `203.pdf` (W08), `204.pdf` (W09), `205.pdf` (W10).

☐ **Build your own Virtual Guest**
`https://osp4diss.vlsm.org/`

Agenda

Agenda (2)

- Bits, bytes, and words
- Numeric data representation and number bases
- Representation of records and arrays

---

[1]Source: ACM IEEE CS Curricula 2013

- Explain why everything is data, including instructions, in computers. [Familiarity]
- Explain the reasons for using alternative formats to represent numerical data. [Familiarity]
- Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays. [Familiarity]

---

[1]Source: ACM IEEE CS Curricula 2013

Week 04: Addressing, Shared Lib, & Pointer

- Reference: (OSC10-ch09 demo-w04)
- This will be a difficult week
  - Pray! Pray! We got to pray just to make it today (McH)!
  - Goosfraba: Turn To Page 394 (AM-HP3)!
- Hardware Address Protection
- Binding & Linking
  - Address Binding
  - Address Space: Logical & Physical
  - Dynamic & Static Linking
  - MMU: Memory Management Unit
  - Base and Limit Registers
  - Swapping
  - Mobile Systems Problem: no swap
- Memory Allocation
  - Contiguous Allocation
  - Multiple-variable-partition Allocation
  - First, Best, Worst Fit Allocation Strategy
- Fragmentation: External / Internal / Compaction

Paging

- Logical/Virtual Address
  - Logical Memory Blocks: Pages
  - Page Number
  - Page Offset
- Page Table
  - Page number index $\Rightarrow$ frame number
  - PTE: Page Table Entry
  - Page Flags: Valid/ Invalid
  - TLB: Translation Look-aside Buffer (Associative Memory).
  - Two-Level Page-Table Scheme
    - OPT: Outer Page Table (P1)
    - PT: Page Table (P2)
  - Three-Level Page-Table Scheme
  - Hashed Page Tables
  - Inverted Page Table
- Physical Address
  - Physical Memory Blocks: Frames
  - Offset (D)
  - Hierarchical Page Tables

Addressing (Eg. 16 bits)

16 Bits Logical Address Table (HEX)																	Examples			
ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	bits	L/B	PTR	VALUE
000X	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF	8	—	**[0008]**	**A8**
001X	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF	8	—	**[0014]**	**B4**
002X	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	8	—	**[0015]**	**B5**
003X	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	16	LE	**[0014]**	**B5 B4**
004X	0A																16	BE	**[0014]**	**B4 B5**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	32	LE	**[0014]**	**B7 B6 B5 B4**
FFFX																	1 address == 1 byte LE: Little Endian BE: Big Endian			

Address Translation Scheme

DEC	HEX	OFFSET	PG	OFF	PG	OFF	PAGE	OFF	PAGE	OFF
00	00	00000	0	0000	00	000	000	00	0000	0
01	01	00001	0	0001	00	001	000	01	0000	1
02	02	00010	0	0010	00	010	000	10	0001	0
03	03	00011	0	0011	00	011	000	11	0001	1
04	04	00100	0	0100	00	100	001	00	0010	0
05	05	00101	0	0101	00	101	001	01	0010	1
06	06	00110	0	0110	00	110	001	10	0011	0
07	07	00111	0	0111	00	111	001	11	0011	1
08	08	01000	0	1000	01	000	010	00	0100	0
09	09	01001	0	1001	01	001	010	01	0100	1
10	0A	01010	0	1010	01	010	010	10	0101	0
11	0B	01011	0	1011	01	011	010	11	0101	1
12	0C	01100	0	1100	01	100	011	00	0110	0
13	0D	01101	0	1101	01	101	011	01	0110	1
14	0E	01110	0	1110	01	110	011	10	0111	0
15	0F	01111	0	1111	01	111	011	11	0111	1
16	10	10000	1	0000	10	000	100	00	1000	0
17	11	10001	1	0001	10	001	100	01	1000	1
18	12	10010	1	0010	10	010	100	10	1001	0
19	13	10011	1	0011	10	011	100	11	1001	1
20	14	10100	1	0100	10	100	101	00	1010	0
21	15	10101	1	0101	10	101	101	01	1010	1
22	16	10110	1	0110	10	110	101	10	1011	0
23	17	10111	1	0111	10	111	101	11	1011	1
24	18	11000	1	1000	11	000	110	00	1100	0
25	19	11001	1	1001	11	001	110	01	1100	1
26	1A	11010	1	1010	11	010	110	10	1101	0
27	1B	11011	1	1011	11	011	110	11	1101	1
28	1C	11100	1	1100	11	100	111	00	1110	0
29	1D	11101	1	1101	11	101	111	01	1110	1
30	1E	11110	1	1110	11	110	111	10	1111	0
31	1F	11111	1	1111	11	111	111	11	1111	1

Memory (20 bits)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
00010	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
00020	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
00030	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
FFFF0																

Variables and File Formats

- 8 bit Variable (eg. `int ii=10;`)
  - Value (10_{10} == 0x 0A)
  - Logical Address (eg. 0x 0040)
  - Meaning & Context (Variabel "ii" is an integer).
  - [0x 0040] == 0x 0A
- Multiple Address Variable (> 1 byte size)
  - Little-Endian (LE)
  - Big-Endian (BE)
  - Bi-Endian
- Executable File Format
  - Ancient Linux/Unix: Assembler Output $\rightarrow$ [a.out].
  - iOS, MacOS: Mach-Output (Mach-O).
  - Linux: Executable and Linking Format (ELF).
  - Windows: Portable Executable (PE) $\rightarrow$
    [.acm, .ax, .cpl, .dll, .drv, .efi, .exe, .mui, .ocx, .scr, .sys, .tsp].

Linux Libraries (1)

Figure: Linux Libraries

- Static Libraries (embeded in the program).
    - Self contained
    - StaticLib.a
- Shared Libraries
    - Dynamic Linking (run-time.so).
    - Dynamic Loading (controlled by the program, DL-API).

Linux Libraries (2)

- putchar(char)
- getpid()
- getppid()
- sprintf(char*, const chat*)
- fflush(NULL)
- MSIZE1 (10k) MSIZE2 (20k) MSIZE3 (50k) MSIZE4 (100k)
  MSIZE5 (1M) MSIZE6 (10M) MSIZE1
- top
    - PID (Process Id), PPID (Parent PID), %MEM (Memory), VIRT
      (Virtual Image KiB), RES (Residen Size KiB), SHR (Shared Memory
      KiB), SWAP (Swapped Size KiB), CODE (Code Size KiB), DATA
      (Data+Stack KiB), USED (Res+Swap Size KiB).
    - Save: ~/.toprc
    - top -b -n 1 -pYOUR_PID
- malloc(size_t)
- free(void*)
- system(const char*)

Makefile

```
CC=gcc
P00=00-global-variables
P01=01-local-variables
...

EXECS= \
        $(P00) \
        $(P01) \
...

DEMOFILES=\
    demo-file1.txt \
    demo-file2.txt \
...

all:  $(EXECS)

$(P00): $(P00).c
    $(CC) $(P00).c -o $(P00) -Xlinker -Map=$(P00).map

$(P01): $(P01).c
    $(CC) $(P01).c -o $(P01) -Xlinker -Map=$(P01).map
...

$(P04): $(P04).c
    $(CC) $(P04).c -o $(P04)
...
clean:
    rm -f ${EXECS}
...
demo:
    bash .shsh
```

00-global-variables

```
/* Global Variables in Data Segment*/
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';
char    varchr4='e';
char    varchr5='f';
char    varchr6='g';
char    varchr7='h';
VARIABLE  +++   VALUE +CHR+ + ADDRESS+
varchr0 =       0X61 = a    0x601038
varchr1 =       0X62 = b    0x601039
varchr2 =       0X63 = c    0x60103a
varchr3 =       0X64 = d    0x60103b
varchr4 =       0X65 = e    0x60103c
varchr5 =       0X66 = f    0x60103d
varchr6 =       0X67 = g    0x60103e
varchr7 =       0X68 = h    0x60103f
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60103X									'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'

Memory Map

```
Memory Configuration (00-global-char.map)

Name          Origin              Length              Attributes
*default*     0x0000000000000000  0xffffffffffffffff
                                                      PLT=Procedure Linkage Table
.plt          0x0000000000400420  0x30                /usr/lib/..../crt1.o
              0x0000000000400430                      puts@@GLIBC\_2.2.5
              0x0000000000400440                      printf@@GLIBC\_2.2.5

.text         0x0000000000400450  0x282

.data         0x0000000000601028  0x18
.data         0x0000000000601038  0x8                 /tmp/cc0DQ6wO.o
              0x0000000000601038                      varchr0
              0x0000000000601039                      varchr1
              ...                                     ...
              0x000000000060103e                      varchr6
              0x000000000060103f                      varchr7

.bss          0x0000000000601040  0x8
```

```
/* Local Variables in Stack Segment */
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';
char    varchr4='e';
char    varchr5='f';
char    varchr6='g';
char    varchr7='h';
VARIABLE  +++  VALUE +CHR+ +++  ADDRESS +++
varchr0 =       0X61 = a    0x7ffcc188b51f
varchr1 =       0X62 = b    0x7ffcc188b51e
varchr2 =       0X63 = c    0x7ffcc188b51d
varchr3 =       0X64 = d    0x7ffcc188b51c
varchr4 =       0X65 = e    0x7ffcc188b51b
varchr5 =       0X66 = f    0x7ffcc188b51a
varchr6 =       0X67 = g    0x7ffcc188b519
varchr7 =       0X68 = h    0x7ffcc188b518
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00007ffc-c188b51X									'h'	'g'	'f'	'e'	'd'	'c'	'b'	'a'

```
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';
char*   ptrchr0=&varchr0;
char*   ptrchr1=&varchr1;
char*   ptrchr2=&varchr2;
char*   ptrchr3=&varchr3;
VARIABLE  +++   VALUE +CHR+ +ADDRESS + +POINTS TO+
varchr0 =        0X61 = a      0x601038
varchr1 =        0X62 = b      0x601039
varchr2 =        0X63 = c      0x60103a
varchr3 =        0X64 = d      0x60103b
ptrchr0 =   0x601038          0x601040         a
ptrchr1 =   0x601039          0x601048         b
ptrchr2 =   0x60103a          0x601050         c
ptrchr3 =   0x60103b          0x601058         d
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									'a'	'b'	'c'	'd'				
00000000–0060104X	00000000–00601038								00000000–00601039							
00000000–0060105X	3A	10	60	00	00	00	00	00	3B	10	60	00	00	00	00	00

```
================================================
/* Global Variables in Data Segment*/
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';
char*   ptrchr0=&varchr0;
char*   ptrchr1=&varchr1;
char*   ptrchr2=&varchr2;
char*   ptrchr3=&varchr3;
char**  ptrptr0=&ptrchr0;
char**  ptrptr1=&ptrchr1;
char**  ptrptr2=&ptrchr2;
char**  ptrptr3=&ptrchr3;
VARIABLE  +++   VALUE +CHR+ +ADDRESS + +POINTS TO+
varchr0 =        0X61 = a   0x601038
varchr1 =        0X62 = b   0x601039
varchr2 =        0X63 = c   0x60103a
varchr3 =        0X64 = d   0x60103b
ptrchr0 =   0x601038       0x601040            a
ptrchr1 =   0x601039       0x601048            b
ptrchr2 =   0x60103a       0x601050            c
ptrchr3 =   0x60103b       0x601058            d
ptrptr0 =   0x601040       0x601060   0x601038
ptrptr1 =   0x601048       0x601068   0x601039
ptrptr2 =   0x601050       0x601070   0x60103a
ptrptr3 =   0x601058       0x601078   0x60103b
================================================
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60103X									'a'	'b'	'c'	'd'				
60104X	601038								601039							
60105X	60103A								60103B							
60106X	601040								601048							
60107X	601050								601058							

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									61	62	63	64				
00000000–0060104X	38	10	60	00	00	00	00	00	39	10	60	00	00	00	00	00
00000000–0060105X	3A	10	60	00	00	00	00	00	3B	10	60	00	00	00	00	00
00000000–0060106X	40	10	60	00	00	00	00	00	48	10	60	00	00	00	00	00
00000000–0060107X	50	10	60	00	00	00	00	00	58	10	60	00	00	00	00	00

```
/* Global Variables in Data Segment*/
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';
char*   ptrchr0=&varchr0;
char*   ptrchr1=&varchr1;
char*   ptrchr2=&varchr2;
char*   ptrchr3=&varchr3;
char**  ptrptr0=&ptrchr0;
char**  ptrptr1=&ptrchr1;
char**  ptrptr2=&ptrchr2;
char**  ptrptr3=&ptrchr3;
char*** ppptr0=&ptrptr0;
```

VARIABLE	+++	VALUE	+CHR+	+ADDRESS +	+POINTS TO+
varchr0 =		0X61	= a	0x601038	
varchr1 =		0X62	= b	0x601039	
varchr2 =		0X63	= c	0x60103a	
varchr3 =		0X64	= d	0x60103b	
ptrchr0 =	0x601038			0x601040	a
ptrchr1 =	0x601039			0x601048	b
ptrchr2 =	0x60103a			0x601050	c
ptrchr3 =	0x60103b			0x601058	d
ptrptr0 =	0x601040			0x601060	0x601038
ptrptr1 =	0x601048			0x601068	0x601039
ptrptr2 =	0x601050			0x601070	0x60103a
ptrptr3 =	0x601058			0x601078	0x60103b
ppptr0 =	0x601060			0x601080	0x601040

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60103X									'a'	'b'	'c'	'd'				
60104X			601038					601039								
60105X			60103A					60103B								
60106X			601040					601048								
60107X			601050					601058								
60108X			601060													

- `***ppptr0 = **ptrptr0 = *ptrchr = varchr0`
- `ppptr0  = [601080] = 601060`
- `ptrptr0 = [601060] = 601040`
- `ptrchr0 = [601040] = 601038`
- `varchr0 = [601038] = 'a'`

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									61	62	63	64				
00000000–0060104X	38	10	60	00	00	00	00	00	39	10	60	00	00	00	00	00
00000000–0060105X	3A	10	60	00	00	00	00	00	3B	10	60	00	00	00	00	00
00000000–0060106X	40	10	60	00	00	00	00	00	48	10	60	00	00	00	00	00
00000000–0060107X	50	10	60	00	00	00	00	00	58	10	60	00	00	00	00	00
00000000–0060108X	60	10	60	00	00	00	00	00								

```
======================================
/* Global Variables in Data Segment*/
int     varint0=0x41424344;
char    varchr0='a';
char    varchr1='b';
char    varchr2='c';
char    varchr3='d';

int*    ptrint0=&varint0;
char*   ptrchr0=&varchr0;

ptrint0=(int*) &varchr2;
varint0=*ptrint0;

ptrchr0=(char*) &varint0;
varchr0=*ptrchr0;

ptrchr0++;
varchr0=*ptrchr0;
======================================
```

```
VARIABLE  +++  VALUE +CHR+ +ADDRESS + +POINTS TO+++
varint0 = 0X41424344 = D     0x601038
varchr0 =       0X61 = a     0x60103c
varchr1 =       0X62 = b     0x60103d
varchr2 =       0X63 = c     0x60103e
varchr3 =       0X64 = d     0x60103f
ptrint0 =   0x601038          0x601048   0X41424344
ptrchr0 =   0x60103c          0x601050            a
!!! ptrint0=(int*) &varchr1;  varint0=*ptrint0; !!!
VARIABLE  +++  VALUE +CHR+ +ADDRESS + +POINTS TO+++
ptrint0 =   0x60103d          0x601048   0X65646362
varint0 = 0X65646362 = b     0x601038
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									44	43	42	41	61	62	63	64
00000000–0060104X	65								38	10	60	00	00	00	00	00
00000000–0060105X	3C	10	60	00	00	00	00	00								
00000000–0060103X									62	63	64	65	61	62	63	64
00000000–0060104X	65								3D	10	60	00	00	00	00	00

```
!!! ptrchr0=(char*) &varint0; varchr0=*ptrchr0; !!!
VARIABLE  +++  VALUE +CHR+ +ADDRESS + +POINTS TO+++
ptrchr0 =   0x601038        0x601050        0X62
varchr0 =        0X62 = b   0x60103c
!!!! !!!!  ptrchr0++;  varchr0=*ptrchr0;  !!!! !!!!
VARIABLE  +++  VALUE +CHR+ +ADDRESS + +POINTS TO+++
ptrchr0 =   0x601039        0x601050        0X63
varchr0 =        0X63 = c   0x60103c
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									44	43	42	41	61	62	63	64
00000000–0060104X	65								38	10	60	00	00	00	00	00
00000000–0060105X	3C	10	60	00	00	00	00	00								

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									62	63	64	65	61	62	63	64
00000000–0060104X	65								3D	10	60	00	00	00	00	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									62	63	64	65	62	62	63	64
00000000–0060105X	38	10	60	00	00	00	00	00								

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000–0060103X									62	63	64	65	63	62	63	64
00000000–0060105X	39	10	60	00	00	00	00	00								

```
unsigned char    varchr0='a';
unsigned char*   ptrchr0=&varchr0;
unsigned char*   ptrcopy=(char *) &ptrchr0;

VARIABLE   +++   VALUE +++ +CHR+ +++ ADDRESS   +++ +PTS TO+
varchr0 =           0X61 = a    0x7ffe7bb7369f
ptrchr0 = 0x7ffe7bb7369f        0x7ffe7bb73690        0X61

!!! !!!!! ptrcopy++; ptrcopy++; ptrcopy++; ... !!!!! !!!
ptrcopy = 0x7ffe7bb73690        0x7ffe7bb73688        0X9F
ptrcopy = 0x7ffe7bb73691        0x7ffe7bb73688        0X36
ptrcopy = 0x7ffe7bb73692        0x7ffe7bb73688        0XB7
ptrcopy = 0x7ffe7bb73693        0x7ffe7bb73688        0X7B
ptrcopy = 0x7ffe7bb73694        0x7ffe7bb73688        0XFE
ptrcopy = 0x7ffe7bb73695        0x7ffe7bb73688        0X7F
ptrcopy = 0x7ffe7bb73696        0x7ffe7bb73688          00
ptrcopy = 0x7ffe7bb73697        0x7ffe7bb73688          00
```

```
!!! !!!!! ptrcopy++; ptrcopy++; ptrcopy++; ... !!!!! !!!
VARIABLE   +++  VALUE +++ +CHR+ +++ ADDRESS  +++ +PTS TO+
ptrchr0 = 0x7ffe7bb7369f          0x7ffe7bb73690    0X61
ptrcopy = 0x7ffe7bb73690          0x7ffe7bb73688    0X9F
ptrcopy = 0x7ffe7bb73691          0x7ffe7bb73688    0X36
ptrcopy = 0x7ffe7bb73692          0x7ffe7bb73688    0XB7
ptrcopy = 0x7ffe7bb73693          0x7ffe7bb73688    0X7B
ptrcopy = 0x7ffe7bb73694          0x7ffe7bb73688    0XFE
ptrcopy = 0x7ffe7bb73695          0x7ffe7bb73688    0X7F
ptrcopy = 0x7ffe7bb73696          0x7ffe7bb73688      00
ptrcopy = 0x7ffe7bb73697          0x7ffe7bb73688      00
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00007FFE–7BB7368X									90	36	B7	7B	FE	7F	00	00
00007FFE–7BB7369X	9F	36	B7	7B	FE	7F	00	00								61
00007FFE–7BB7368X									91	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									92	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									93	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									94	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									95	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									96	36	B7	7B	FE	7F	00	00
00007FFE–7BB7368X									97	36	B7	7B	FE	7F	00	00

07-addresses (LE)

```
unsigned int  glInt1 = 0x41;
unsigned int  glInt2 = 0x42;
unsigned int  glInt3 = 0x43;
unsigned int  glInt4 = 0x44;
unsigned int  glInt5 = 0x45;
unsigned int* heapArray[] =
            {&glInt1, &glInt2, &glInt3, &glInt4, &glInt5};

Variable Name      Address    Size(S)/Value(V)
==========================================
glInt1             0x601060         0X41 (V)
glInt2             0x601064         0X42 (V)
glInt3             0x601068         0X43 (V)
glInt4             0x60106c         0X44 (V)
heapArray---       0x601080   0X601060 (V)
heapArray[0]       0x601080   0X601060 (V)
heapArray[1]       0x601088   0X601064 (V)
heapArray[2]       0x601090   0X601068 (V)
heapArray[3]       0x601098   0X60106C (V)
heapArray[4]       0x6010a0   0X601070 (V)
```

```
#define ALLOC0    0x4BD8
#define ALLOC1    0xFF8
#define ALLOC2    0x18
#define ALLOC3    0x19
#define ALLOC4    1
heapArray[0]=malloc(ALLOC0);
heapArray[1]=malloc(ALLOC1);
heapArray[2]=malloc(ALLOC2);
heapArray[3]=malloc(ALLOC3);
heapArray[4]=malloc(ALLOC4);

Variable Name     Address     Size(S)/Value(V)
=============================================
heapArray---      0x601080    0X23CF420 (V)
heapArray[0]      0x601080    0X23CF420 (V)
heapArray[1]      0x601088    0X23D4000 (V)
heapArray[2]      0x601090    0X23D5000 (V)
heapArray[3]      0x601098    0X23D5020 (V)
heapArray[4]      0x6010a0    0X23D5050 (V)
```

```
long printVariable(char* varName, void* varValue, long endAddr) { ... }
long printHeapArray(int mode) { ... }
long demoMalloc(int mode) { ... }
long tripleLoop(int mode) { ... }
void main(void)            { ... }

Variable Name     Address   Size(S)/Value(V)
============================================
printf            0x400480
malloc            0x400490
printVariable     0x400596       0XBE (S)
printHeapArray    0x400654       0XA3 (S)
demoMalloc        0x4006f7       0X7E (S)
tripleLoop        0x400775       0XFC (S)
main              0x400871      0X148 (S)
```

```
################################################################################

Memory Configuration
                0x0000000000400238        (SEGMENT-START ("text-segment", 0x400000) + SIZEOF-HEADERS)

 .plt           0x0000000000400460        0x40 /usr/lib/gcc/.../x86-64-linux-gnu/crt1.o
                0x0000000000400470               puts@@GLIBC\_2.2.5
                0x0000000000400480               printf@@GLIBC\_2.2.5
                0x0000000000400490               malloc@@GLIBC\_2.2.5

 .text          0x00000000004004a0        0x592
 .text          0x0000000000400596        0x41d /tmp/ccU78N7D.o
                0x0000000000400596               printVariable
                0x0000000000400654               printHeapArray
                0x00000000004006f7               demoMalloc
                0x0000000000400775               tripleLoop
                0x0000000000400871               main

 .data          0x0000000000601060        0x48 /tmp/ccU78N7D.o
                0x0000000000601060               glInt1
                0x0000000000601064               glInt2
                0x0000000000601068               glInt3
                0x000000000060106c               glInt4
                0x0000000000601070               glInt5
                0x0000000000601080               heapArray

################################################################################
```

```
#define NOP()    __asm__("nop")  /* No Operation inline gcc ASM *** */
#include <stdio.h>
int   varInt1   = 0x01;
int   varInt2   = 0x02;
int*  ptrInt1   = &varInt1;
int*  ptrInt2   = &varInt2;
void function1(void) {
   NOP();
}
void function2(int iif2) {
   printf("function2:    iif2 = %d\n", ++iif2);
}
void function3(int* iif3) {
   printf("function3:    iif3 = %d\n", ++(*iif3));
}
int  function4(void) {
   NOP();
}
int* function5(void) {
   NOP();
}
void main(void) {                                    // main-1:    *ptrInt1 = 1
   function1();                                      // function2:     iif2 = 2
   printf("main-1:    *ptrInt1 = %d\n", *ptrInt1);   // main-2:    *ptrInt1 = 1
   function2(*ptrInt1);                              // main-3:     varInt1 = 1
   printf("main-2:    *ptrInt1 = %d\n", *ptrInt1);   // function3:     iif3 = 2
   printf("main-3:     varInt1 = %d\n",  varInt1);   // main-4:     varInt1 = 2
   function3(&varInt1);
   printf("main-4:     varInt1 = %d\n",  varInt1);
}
```

```c
#include <stdio.h>

typedef struct {
    char* nama;
    int   umur;
    int   semester;
    char* NIM;
} student;

void printStruct(student* ss) {
    printf("%-10s %11s %3d %2d\n", ss->nama, ss->NIM, ss->umur, ss->semester);
}

student global;
void init(void) {
    global.nama     = "Burhan";
    global.NIM      = "1205000003";
    global.umur     = 10;
    global.semester = 2;
}

void main(void) {
    student mhs = {"Ali", 12, 1, "1205000001"};
    printStruct(&mhs);
    init();
    printStruct(&global);
}
==============================
Ali        1205000001 12  1
Burhan     1205000003 10  2
```

☐ Week 04 Token: **NEWHOPE**

☐ Week 04: Assignment

1. Read: (OSC10 chapter 09) and try Demos Week 04 (and before).
2. Update your Virtual Guest.
3. Visit https://os.vlsm.org/GitHubPages/. Review **Last Week TOP 10 List** and pick at least 3 out of your 10 closest neighbors. See https://cbkadal.github.io/os202/TXT/myrank.txt.
4. Create your **TOP 10 List** of Week 04 (eg. https://cbkadal.github.io/os202/W04/). Do not use lecture material. Please be more creative!
5. Run "chktoken NEWHOPE" and write the result into myW04token.txt.
6. Download https://os.vlsm.org/WEEK/W04.tar.bz2.asc and write the result into TXT/myW04.txt.
7. Update your log (eg. https://cbkadal.github.io/os202/TXT/mylog.txt).
8. Update bash script (eg. https://cbkadal.github.io/os202/TXT/myscript.sh).
9. Make SHA256SUM and sign it (detached, armor) as SHA256SUM.asc.

☐ The "Assignment Day" is every Thursday morning.

☐ This page is https://os.vlsm.org/Slides/check04.pdf.

The End

- ☐ This is the end of the presentation.
- ☒ This is the end of the presentation.
- This is the end of the presentation.