Address Space

Operating System (OS)
안인규

강의자료는 KAIST 원유집 교수님 OS 강의 자료를 참고했습니다.

Memory Virtualization

- What is **memory virtualization**?
  - OS virtualizes its physical memory.
  - OS provides an illusion memory space per each process.
  - It seems to be seen like each process uses the whole memory

Recap | Virtualization (Memory)

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: mem <value>\n");
                exit(1);
        }
        int *p = malloc(sizeof(int));  // a1: allocate some memory
        assert(p != NULL);
        printf("(%d) address of p: %08x\n", getpid(), (unsigned) p);
                // a2: print out the address of the memory
        *p = atoi(argv[1]); // assign value to addr stored in p
        while (1) {
                Spin(1);
                *p = *p + 1;
                printf("(%d) p: %d\n", getpid(), *p); // a4
        }
        return 0;
}
```

<mem.c>

Recap | Virtualization (Memory)

- Running the program (mem.c) multiple times

```
prompt> ./mem 1 & ./mem 5 &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 6
(24114) p: 2
(24113) p: 7
(24113) p: 3
(24114) p: 8
(24113) p: 4
(24114) p: 9
...
```

Virtualizing memory!

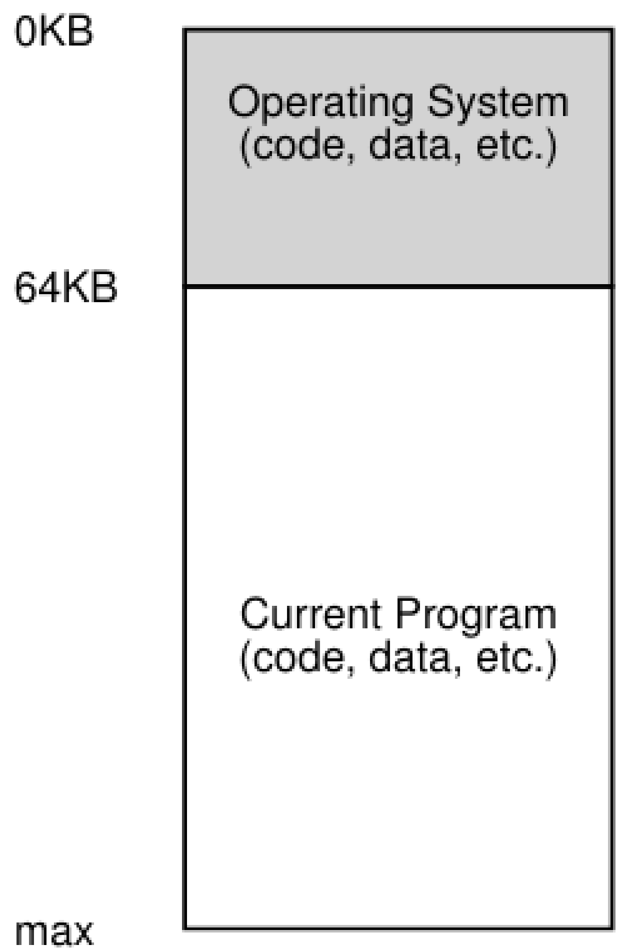
- 프로그램 (./mem)들이 같은 physical memory를 공유하는 것이 아닌, 각각 private memory를 가지고 있는 것처럼 보인다

Benefit of Memory Virtualization

- Ease of use in programming
- Memory efficiency in terms of times and space
  - Paging, TLB (Translation Lookaside Buffer) 등
- The guarantee of isolation for processes as well as OS
  - Protection from **errant accesses** of other processes

Early Systems

- Load only one process in memory
  - Early machines did not provide much of a abstraction to users
  - **Such early systems are not suitable for multiprogramming**

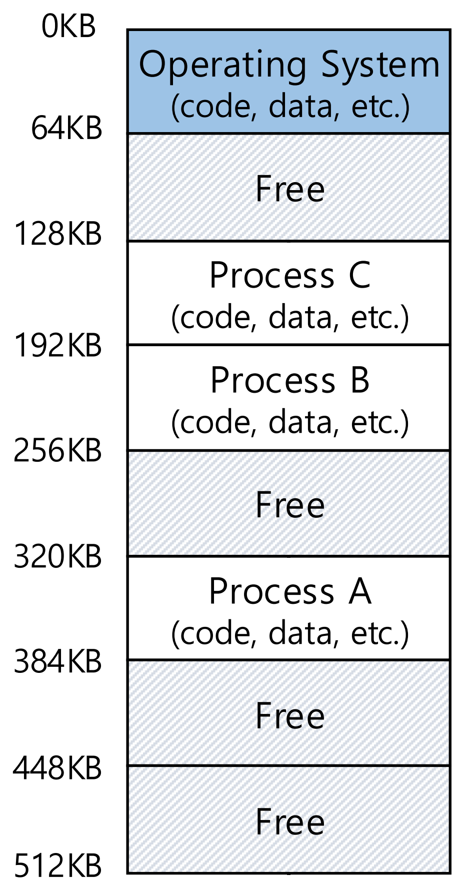

0KB
64KB
max
Operating System (code, data, etc.)
Current Program (code, data, etc.)

Physical Memory

Multiprogramming and Time Sharing

- **Load multiple processes** in memory.
  - Execute one for a short while.
  - Switch processes between them in memory.
  - Increase utilization and efficiency.

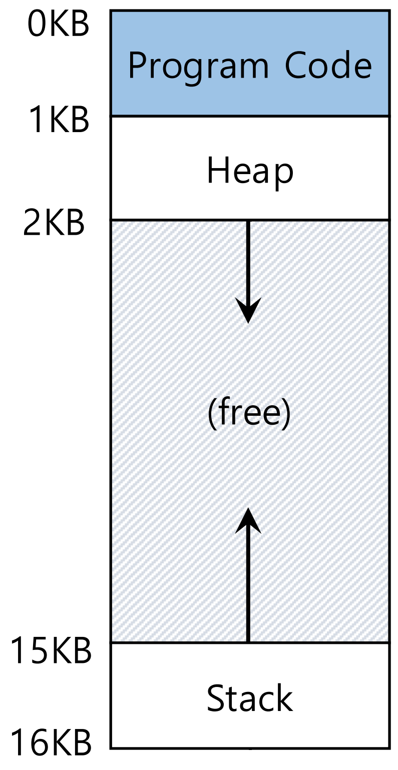
- Cause an important **protection issue**.
  - Errant memory accesses from other processes


0KB
64KB
128KB
192KB
256KB
320KB
384KB
448KB
512KB
Operating System (code, data, etc.)
Free
Process C (code, data, etc.)
Process B (code, data, etc.)
Free
Process A (code, data, etc.)
Free
Free

Physical Memory

Address Space (User)

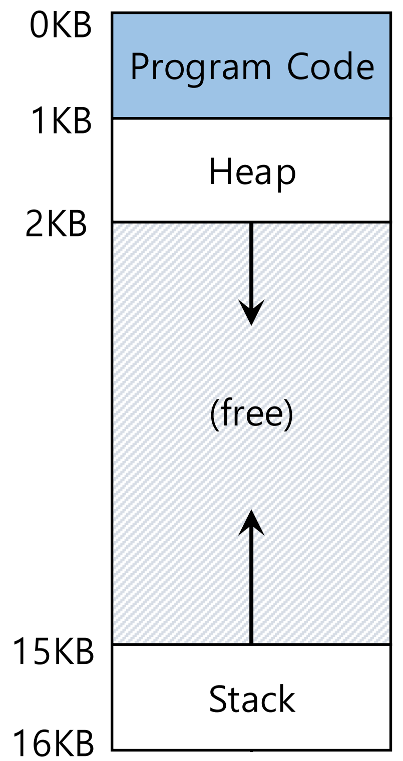
- OS creates an **abstraction** of physical memory.
  - The address space contains all about a running process.
  - That is consist of program code, heap, stack and etc.



Address Space

Address Space (User) (Cont.)

- Code
  - Where instructions live
- Heap
  - Dynamically allocate memory.
    - `malloc` in C language
    - `new` in object-oriented language
- Stack
  - Store return addresses or values.
  - Contain local variables arguments to routines.



Address Space

Virtual Address

- **Every address** in a running program is virtual.
  - OS translates the virtual address to physical address

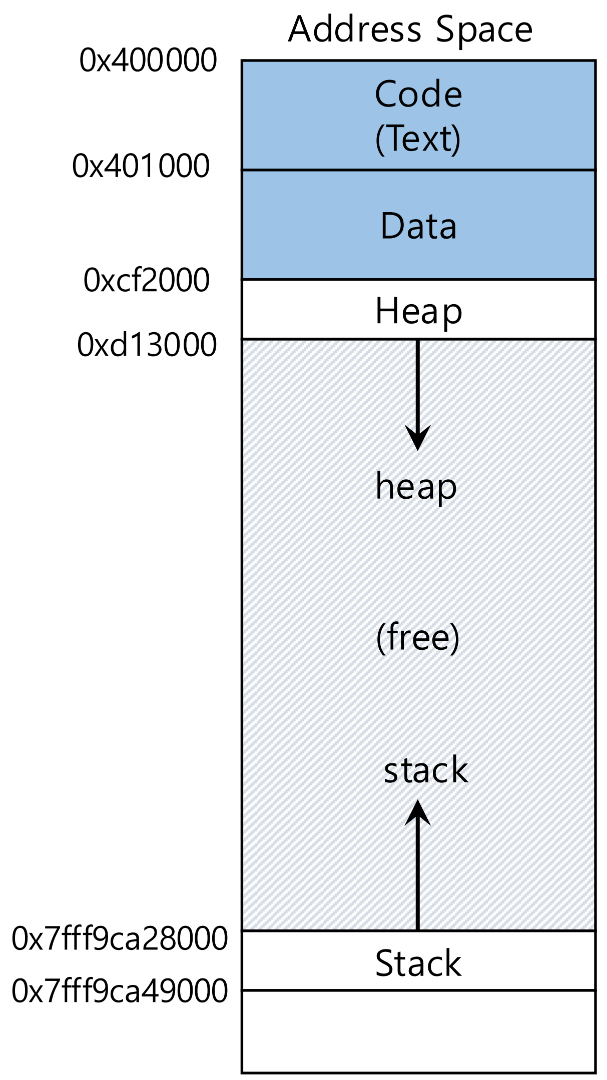
```c
#include <stdio.h>
#include <stdlib.h>

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

    printf("location of code  : %p\n", (void *) main);
    printf("location of heap  : %p\n", (void *) malloc(1));
    int x = 3;
    printf("location of stack : %p\n", (void *) &x);

    return x;
}
```

A simple program that prints out addresses

Virtual Address

- The output in 64-bit Linux machine

```
location of code  : 0x40057d
location of heap  : 0xcf2010
location of stack : 0x7fff9ca45fcc
```



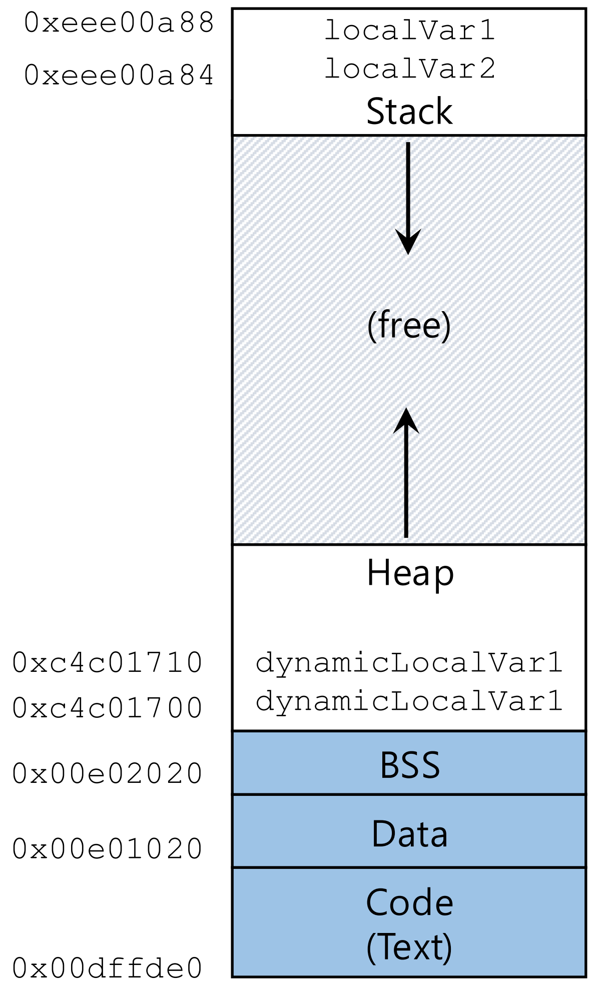
Components of Virtual Address Space

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

int InitializedGlobal[1024] = {0,};
int UnintGlobal[1024];

int main() {
    int localVar1;
    int localVar2;
    int *dynamicLocalVar1;
    int *dynamicLocalVar2;

    dynamicLocalVar1 = malloc(sizeof(int));
    dynamicLocalVar2 = malloc(sizeof(int));

    printf("code                 : 0x%x\n", main);
    printf("Data                 : 0x%x\n", &InitializedGlobal);
    printf("BSS(Uninit Data)     : 0x%x\n", &UnintGlobal);

    printf("stack localVar1      : 0x%x\n", &localVar1);
    printf("stack localVar2      : 0x%x\n", &localVar2);
    printf("heap dynamicLocalVar1: 0x%x\n", dynamicLocalVar1);
    printf("heap dynamicLocalVar2: 0x%x\n", dynamicLocalVar2);
    return 0;
}
```

Components of Virtual Address Space

```
BulGok:~ yjwon$ ./a.out
code                    : 0xdffde0
Data                    : 0xe01020
BSS(Uninit Data)        : 0xe02020
stack localVar1         : 0xeee00a88
stack localVar2         : 0xeee00a84
heap dynamicLocalVar1: 0xc4c01700
heap dynamicLocalVar2: 0xc4c01710
```

Minimum heap allocation unit: 16 Byte



구　분	Byte
1B	1
2B	2
4B	4
8B	8
16B	16
32B	32
64B	64
128B	128
256B	256
512B	512
1KB	1024
2KB	2048
4KB	4096
8KB	8192
16KB	16384
32KB	32768
64KB	65536
128KB	131072
256KB	262144
512KB	524288
1MB	1048576
2MB	2097152
4MB	4194304
8MB	8388608
16MB	16777216
32MB	33554432
64MB	67108864
128MB	134217728
256MB	268435456
512MB	536870912