Virtual Memory
Lecture - 30

* Physical Addressing

CPU

Physical Address (PA) 4

Main Memory

Data word

Address Space
Virtual Addressing

CPU chip.

Address translation

Physical address

Main Memory

0 1 2 3 4 5 6 7 8

M-1

data word.

Executing a program in a computer

1. Entire program in main memory

2. Segmentation.

3. Paging.
Segmentation

If \( P_1 \) and \( P_2 \) executing concurrently.

Only \( P_1 \) is executing.
Problem: Fragmentation

Memory

Virtual memory

Paging
Dynamic Memory Allocation.

Program Memory

- 4 GB
- Reserved for OS
- Stack
- Memory mapped region for shared libraries
- Heap
- Uninitialized data (.bss)
- Initialized data (.data)
- Code (.text)
- Data (.data)
- Text (.text)
- Program (.text)
- Don't touch

;bss = Block Started by Symbol
```c
#include <stdio.h>
#include <stdlib.h>

int g_counter = 0;
int g_status = 0;

int main ()
{
    int i;
    int n = 10;
    int *ptr = malloc(sizeof(int) * n);
    if (ptr == NULL) { printf("Error\n"); exit(1); }
    for (i = 0; i < n; i++)
    {
        ptr[i] = i;
    }
    free(ptr);
    return 0;
}
```
A dynamic memory allocator maintains an area of a process's virtual memory known as the **heap**.

- **heap** → collection of various-sized blocks.
- **block** → contiguous chunk of virtual memory.
- **block** ← allocated, free.

**Allocators**

- **Explicit allocators**
  - `malloc`, `free`
  - `new`, `delete`

- **Implicit allocators**
  - Garbage collectors
  - e.g., Java
Functions to access the heap memory.

```c
#include <stdlib.h>

void *malloc (size_t size);

Returns: ptr to allocated block if OK, NULL on error.

unsigned int free (void *ptr);

Returns: nothing.

free (p);

free (NULL);  \rightarrow\text{ no operation is performed.}

void *calloc (size_t nmemb, size_t size);

void *realloc (void *ptr, size_t size);

\text{if } ptr == \text{NULL} \Rightarrow \text{malloc (size)}

\text{if } \text{size} == 0 \&\& \text{ptr} \neq \text{NULL} \Rightarrow \text{free (ptr)}
```


```c
#include <unistd.h>

int brk (void *addr);

void *sbrk (intptr_t increment);
```

typedef of long or int

brk () and sbrk () change the location of the
program break.

brk () - sets the end of the "data segment"

top of the heap to the value specified by addr.

Returns: 0 on success
-1 on error.

sbrk () - increments the program's data space by

"increment" bytes.

Returns: On success, returns the previous program

on error, (void *) -1 is returned.

0xFFF00000
Address Space

An ordered set of non-negative integer addresses.

eq. 32-bit address space

\[ \{0, 1, 2, \ldots, 2^{32} - 1\} \]

In a 32-bit address space, there are \(2^{32}\) addresses.

Linear address space - if the integer addresses are consecutive.

Virtual Address Space

\[ \{0, 1, 2, \ldots, N\} \]

Physical Address Space

\[ \{0, 1, 2, \ldots, M\} \]

\[ N = 2^n \]

\[ M = 2^m \]

Each byte of main memory has a virtual address chosen from the virtual address space, and a physical address chosen from the physical address space.