## Virtual Memory

The memory with which a process interacts is not the same layout as physical RAM.

The OS uses processor-dependent functionality to perform mapping, giving the process a predictable memory layout.

Virtual memory segregates processes so that they cannot access each other’s RAM. This is called protected memory.

Other uses of virtual memory include mmap – map a file on disk into memory.

## UNIX Virtual Address Space, I

Starting at 0, addresses increasing:

- OS-dependent initial region (first page at least): typically off limits to detect null pointer bug

- Text (instructions): loaded from executable file

  Typically read-only and protected by the OS. This is why you can’t modify a char * initialized from a literal string. It points to Text.

- Initialized static/global data: loaded from executable file

- Uninitialized static/global data (aka **bss**): set to zero (SIZE listed in executable file)

## UNIX Virtual Address Space, II

- Heap, growing up
- VAST unallocated region
- Stack, growing down
- OS/ architecture-dependent region

## UNIX Virtual Address Space, III

Shared libraries & "shared memory objects" placed in "hole" between stack & heap

When stack region overflows, extended automatically by OS (up to some hard limit)

Heap extended (and shrunk) under program control by sbrk(2)

Heap managed by language "runtime system" (Java, C++) or malloc library (C), which calls sbrk(2)

## UNIX Virtual Address Space, IV

**brk**() and **sbrk**() change the location of the *program break*, which defines the end of the process's data segment (i.e., the program break is the first location after the end of the uninitialized data segment). Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

**brk**() sets the end of the data segment to the value specified by *addr*, when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size. On success, **brk**() returns zero. On error, -1 is returned.

**sbrk**() increments the program's data space by a specified increment. Calling **sbrk**() with an increment of 0 can be used to find the current location of the program break. On success, **sbrk**() returns the previous program break. (If the break was increased, then this value is a pointer to the start of the newly allocated memory). On error, *(void *)-1* is returned.

## UNIX Address Space Example

```
int GlobalInit = 111;
int Global;

int main()
{
    static int StaticInit = 222;
    static int Static;
    int AutomaticInit = 333;
    int Automatic;

    printf(“ &GlobalInit = %p\n”, &GlobalInit);
    printf(“ &StaticInit = %p\n”, &StaticInit);
    printf(“ &Global = %p\n”, &Global);
    printf(“ &Static = %p\n”, &Static);
    printf(“ &AutomaticInit = %p\n”,
&AutomaticInit);
    printf(“ &Automatic = %p\n”, &Automatic);
    return 0;
}
```

## Linux Address Space Example

```
&AutomaticInit = 0xbffffcb8
&Automatic = 0xbffffcb4

&Global = 0x80496d8
&Static = 0x80496d4
&StaticInit = 0x80495b4
&GlobalInit = 0x80495b0
```

All pairs consecutive

BSS data in reverse order