The Abstraction: Address Spaces

How do programs share main memory?

By Matthew Tancreti
For COM S 352
Iowa State University

Address Spaces

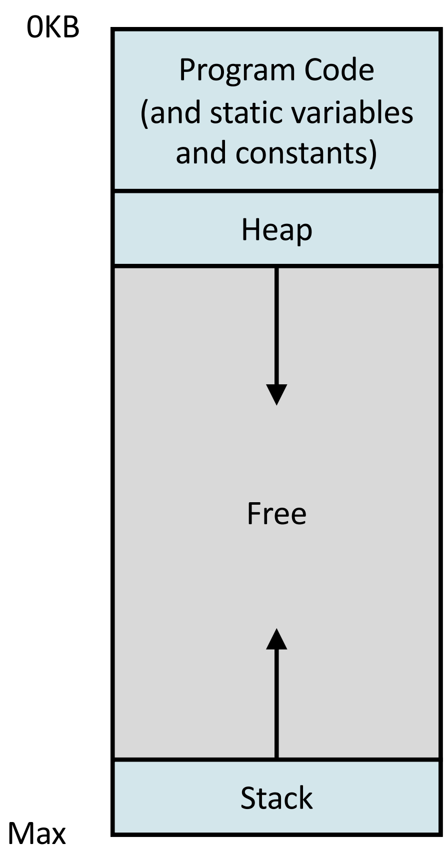
Processes must share main memory, we must account for processes

- created and destroyed
- requesting more memory during execution
- requesting more memory than is physically available

How to meet the memory demands of processes?

Premier Disque, Robert Delaunay, 1913 [source]

Process Address Space



A process' view of memory is called its **address space**

The address space is an abstraction of the physical memory, assumptions:

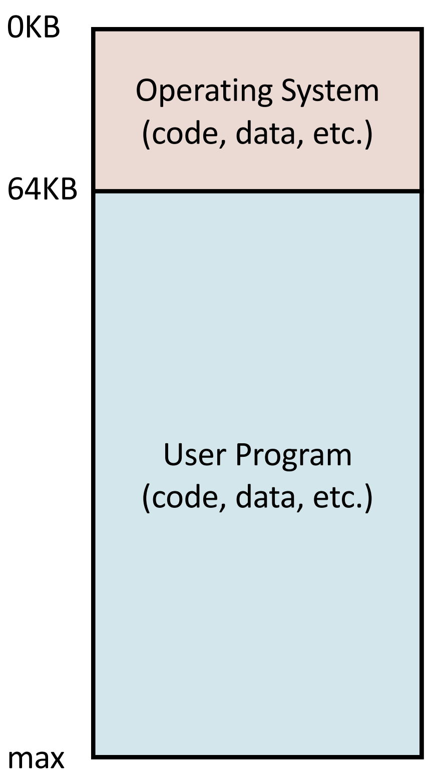
- Address space starts at 0
- Address space is contiguous
- All address available at any time

At the top are static items (e.g., code, global variables and constants)

Processes have **two forms of dynamic memory**: **heap** and the call **stack**

Because they both need to grow an unknown amount, it is logical to place them on opposite ends of the address space

Single Program

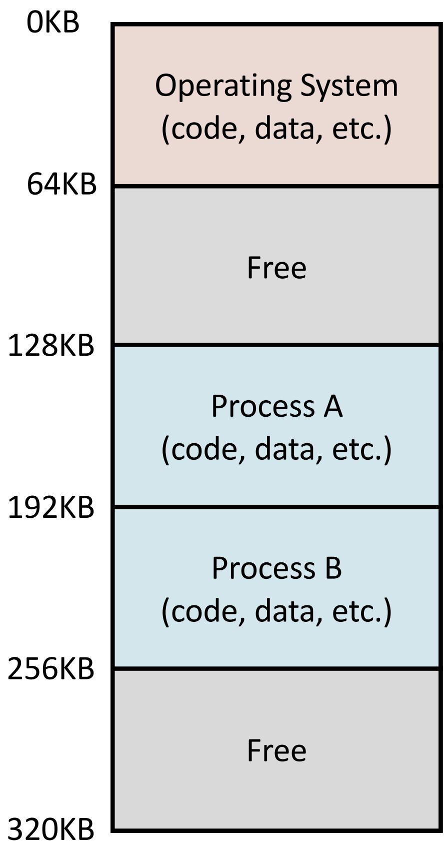


Main memory divided between the OS and user program

Addresses assigned statically (at compile time)

Still used in simple (single program) embedded systems

Multiprogramming

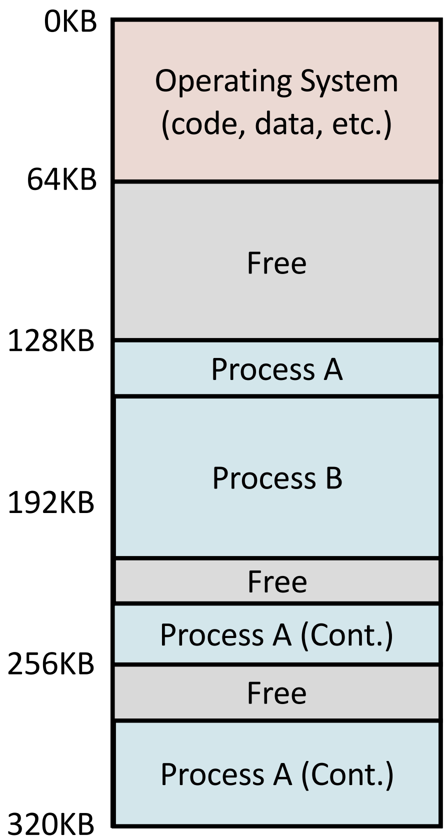


Processes share main memory

Simplest scheme is to assign contiguous regions of memory

Becomes costly when a process needs to grow its memory

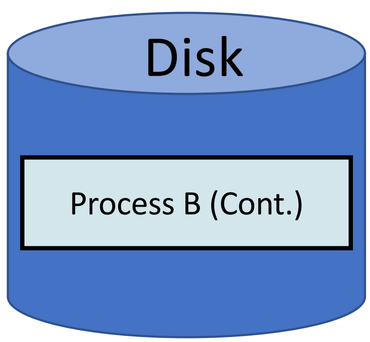
Memory Virtualization



We want to give users an easy view of memory, but the reality is much more messy

- Memory can become fragmented
- Unused address space doesn't need to be mapped to physical memory
- Not every process will fit in memory, use disk for extra storage

Called **memory virtualization**



GB vs GiB

Early programmers noticed an approximate relationship between binary and decimal numbers

2^10 = 1,024 which is almost 1 thousand

2^20 = 1,048,576 which is almost 1 million…

They adapted a binary version of the SI units

2^10 = 1 KB

2^20 = 1 MB…

This became confusing when marketing memory and disk sizes, so new SI binary units were created

In this class we will use the traditional interpretation (e.g., 1KB = 1,024, 1MB = 1,048,576…)

SI Decimal		**SI Binary**	
10^3	kilobyte (KB)	2^{10}=1024	kibibyte (KiB)
10^6	megabyte (MB)	2^{20}=1,048,576	mebibyte (MiB)
10^9	gigabyte (GB)	2^{30}=1,073,741,824	gibibyte (GiB)
10^{12}	terabyte (TB)	2^{40}=1,099,511,627,776	tebibyte (TiB)

Conversions

Problem: write 62,000,000,000 bytes in SI notation

62,000,000,000 / 2^{30} = 57.74GiB (or in this class GB, assuming binary interpretation)

Problem: how many bytes is 32KB (assuming binary interpretation of KB)?

32 * 2^{10} = 32,768 bytes

Word Size

Processor architecture determines word size

Registers, instructions, address bus and data bus are typically one word

Example: 32-bit machine can address 4GB of memory

7 0
8-bit 00000000
HEX: 0x00

15 0
16-bit 00000000 00000000
HEX: 0x00 00

31 0
32-bit 00000000 00000000 00000000 00000000
HEX: 0x00 00 00 00

63 0
64-bit 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
HEX: 0x00 00 00 00 00 00 00 00

Masking

Making is used to reveal any number of specific bits

Question: How to get the value of only to 4 least significant digits?

7 0

8-bit 01101010

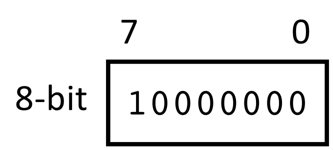
```
result = data & 0x0F
```

Flags

Flags are a single bit to represent a Boolean value

Used in many encoding schemes (e.g., compression, images, instructions, etc.)

For example: Suppose bit 7 indicates extended mode, how to test bit 7?




```
if (data & 0x80)
```