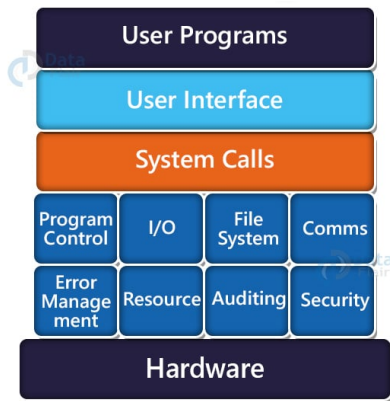


# CSE 4733/6733 - Operating System 1

Stephen A. Torri, Ph.D.

Mississippi State University

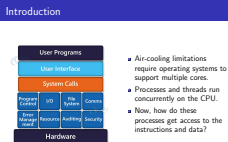


- Air-cooling limitations require operating systems to support multiple cores.
- Processes and threads run concurrently on the CPU.
- Now, how do these processes get access to the instructions and data?

2023-01-30

CSE 4733/6733

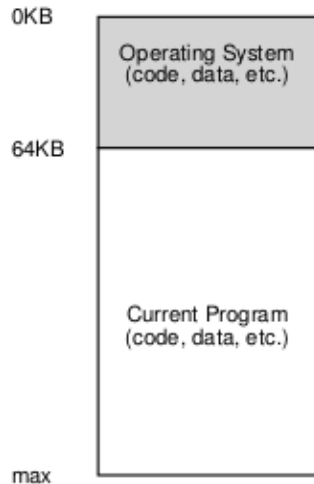
## Introduction



1. The limitation of running a single core faster moved chip designers into thinking about doing multiple tasks simultaneously to tackle the heat issue of a single core.
2. Now software engineers are rewriting software applications to run in parallel using processes and threads for better CPU utilizations.
3. CPUs now try to ensure that frequently used instructions are kept in a cache next to instructions to follow.

# Early Systems

Ref: M. Vutukuru



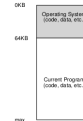
- Early Operating systems only ran a problem like in this figure.
- Microsoft DOS (MS-DOS) would only run a single program.

2023-01-30

CSE 4733/6733

## Early Systems

Early Systems  
Ref: M. Vutukuru



- Early Operating systems only ran a problem like in this figure.
- Microsoft DOS (MS-DOS) would only run a single program.

1. When MS-DOS runs a program, it gives all the available memory to the running application. Any attempt to allocate memory without giving unused memory back to the system will produce an "insufficient memory" error.[1]
2. DOS provides several services dealing with loading, executing, and terminating programs:[1]
  - Program termination: Close all open resources on shutdown.
  - Terminate and stay resident: Memory would be reserved by the program upon shutdown (e.g., device drivers).
  - Execute a program: This command allows you to load or load and execute a program on the disk drive.

# Time Sharing Requires Automatic Memory Allocation[2]

Image from Boston Globe[3]



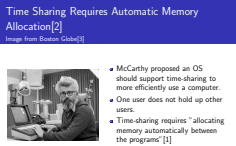
- McCarthy proposed an OS should support time-sharing to more efficiently use a computer.
- One user does not hold up other users.
- Time-sharing requires "allocating memory automatically between the programs" [1]

2023-01-30

CSE 4733/6733

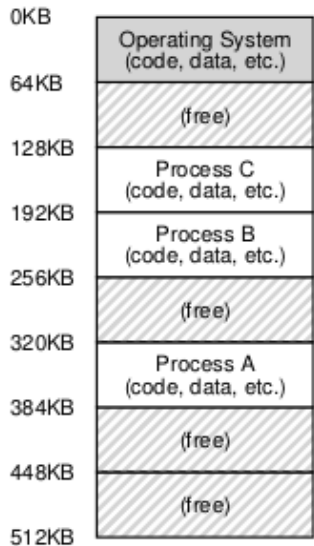
## Time Sharing Requires Automatic Memory Allocation[2]

1. McCarthy states that the response time of staff to changes varied from 3 to 36 hours depending on the machine state, operator efficiency, and backlog.
2. The only way for a computer to be responsive was by time-sharing. So the "computer must attend to other customers while one customer is reacting to some output." [2]
3. To prevent a bad program from destroying another program, it is necessary that programs cannot get outside their assigned address space. These should make it unlikely for a program to accidentally cause others to fail.



# Why virtualize memory?

Ref: M. Vutukuru



- Because real view of memory is messy!
- Earlier, memory had only code of one running process (and OS code)
- Now, multiple active processes timeshare CPU
  - Memory of many processes must be in memory
  - Non-contiguous too
- Need to hide this complexity from user

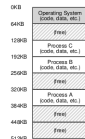
2023-01-30

CSE 4733/6733

Why virtualize memory?

Why virtualize memory?

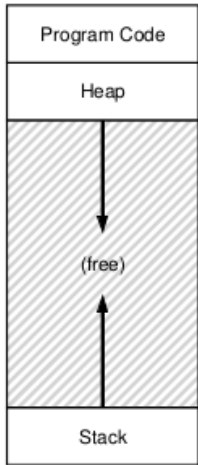
Ref: M. Vutukuru



- Because real view of memory is messy!
- Earlier, memory had only code of one running process (and OS code)
- Now, multiple active processes timeshare CPU
  - Memory of many processes must be in memory
  - Non-contiguous too
- Need to hide this complexity from user

# Abstraction: (Virtual) Address Space

Ref: M. Vutukuru



- Virtual address space: every process assumes it has access to a large space of memory from address 0 to a MAX
- Contains program code (and static data), heap (dynamic allocations), and stack (used during function calls)
- Stack and heap grow during runtime
- CPU issues loads and stores to virtual addresses

2023-01-30

CSE 4733/6733

## └─ Abstraction: (Virtual) Address Space

1. When we describe the address space, what we are describing is the abstraction that the OS is providing to the running program. The program file rather it is loaded at some arbitrary physical address(es).

Abstraction: (Virtual) Address Space

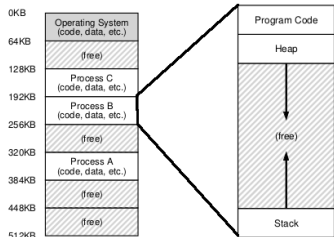
Ref: M. Vutukuru



- Virtual address space: every process assumes it has access to a large space of memory from address 0 to a MAX
- Contains program code (and static data), heap (dynamic allocations), and stack (used during function calls)
- Stack and heap grow during runtime
- CPU issues loads and stores to virtual addresses

# How Is Actual Memory Reached?

Ref: M. Vutukuru



- Address translation from virtual addresses (VA) to physical addresses (PA) – CPU issues loads/stores to VA, but memory hardware accesses PA.
- OS allocates memory and tracks the location of processes.
- Translation done by memory hardware called Memory Management Unit (MMU) – OS makes the necessary information available

2023-01-30

CSE 4733/6733

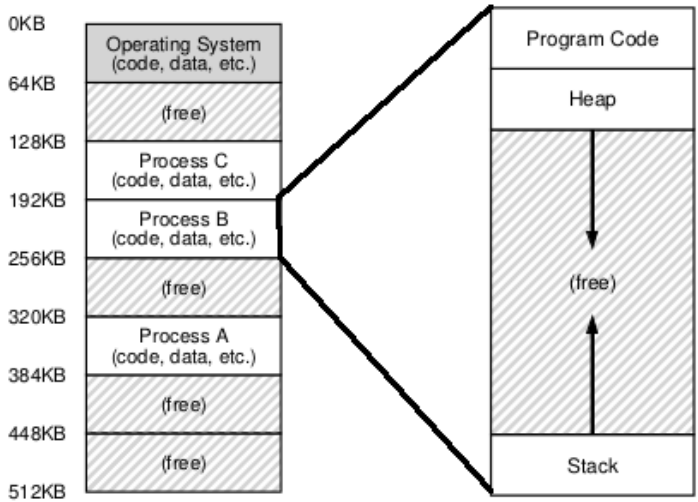
└─How Is Actual Memory Reached?

How Is Actual Memory Reached?  
Ref: M. Vutukuru

- Address translation from virtual addresses (VA) to physical addresses (PA) – CPU issues loads/stores to VA, but memory hardware accesses PA.
- OS allocates memory and tracks the location of processes.
- Translation done by memory hardware called Memory Management Unit (MMU) – OS makes the necessary information available

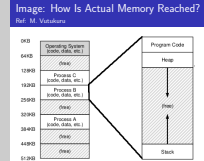
# Image: How Is Actual Memory Reached?

Ref: M. Vutukuru



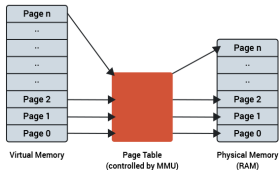
2023-01-30 CSE 4733/6733

Image: How Is Actual Memory Reached?



# Example: Paging

Ref: M. Vutukuru



- OS divides virtual address space into fixed-size pages and physical memory into frames.
- To allocate memory, a page is mapped to a free physical frame
- Page table stores mappings from virtual page number to physical frame number for a process (e.g, page 0 to frame 3)
- MMU has access to page tables, and uses it to translate VA to PA

2023-01-30

CSE 4733/6733

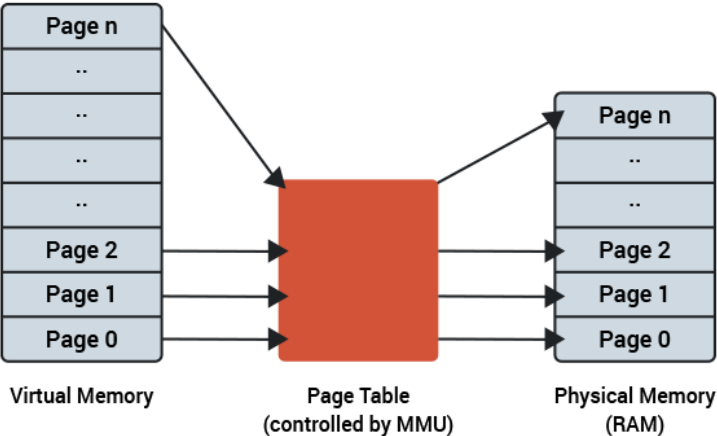
## Example: Paging

Example: Paging  
Ref: M. Vutukuru



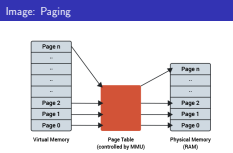
- OS divides virtual address space into fixed-size pages and physical memory into frames.
- To allocate memory, a page is mapped to a free physical frame
- Page table stores mappings from virtual page number to physical frame number for a process (e.g, page 0 to frame 3)
- MMU has access to page tables, and uses it to translate VA to PA

# Image: Paging



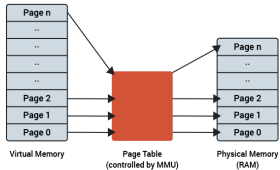
2023-01-30 CSE 4733/6733

Image: Paging



# Goals Of Memory Virtualization

Ref: M. Vutukuru




- **Transparency:** user programs should not be aware of the messy details
- **Efficiency:** minimize overhead and wastage in terms of memory space and access time
- **Isolation and protection:** a user process should not be able to access anything outside its address space

2023-01-30

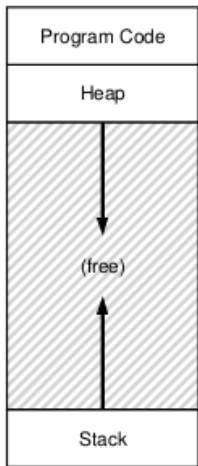
CSE 4733/6733

## Goals Of Memory Virtualization

- 
- **Transparency:** user programs should not be aware of the messy details
  - **Efficiency:** minimize overhead and wastage in terms of memory space and access time
  - **Isolation and protection:** a user process should not be able to access anything outside its address space

# How can a user allocate memory?

Ref: M. Vutukuru



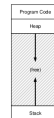
- OS allocates a set of pages to the memory image of the process
- Within this image:
  - Static/global variables are allocated in the executable
  - Local variables of a function on the stack
  - Dynamic allocation with malloc on the heap

2023-01-30

CSE 4733/6733

└ How can a user allocate memory?

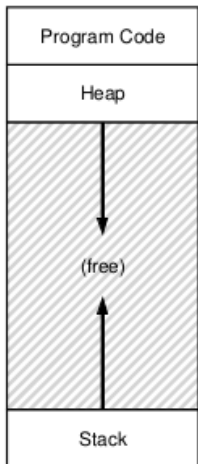
How can a user allocate memory?  
Ref: M. Vutukuru



- OS allocates a set of pages to the memory image of the process
- Within this image:
  - Static/global variables are allocated in the executable
  - Local variables of a function on the stack
  - Dynamic allocation with malloc on the heap

# Memory allocation system calls

Ref: M. Vutukuru



- **malloc** implemented by the C library provides algorithms for efficient memory allocation and free space management.
- **brk** or **sbrk** are used to grow heap.
- **mmap** is used to allocate a page sized memory using the `mmap()` system call.

2023-01-30

CSE 4733/6733

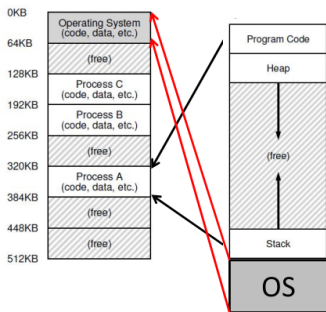
└ Memory allocation system calls



- **malloc** implemented by the C library provides algorithms for efficient memory allocation and free space management.
- **brk** or **sbrk** are used to grow heap.
- **mmap** is used to allocate a page sized memory using the `mmap()` system call.

# A subtle point: what is the address space of the OS?

Ref: M. Vutukuru

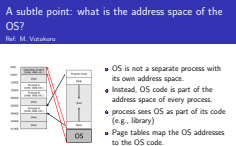


- OS is not a separate process with its own address space.
- Instead, OS code is part of the address space of every process.
- process sees OS as part of its code (e.g., library)
- Page tables map the OS addresses to the OS code.

2023-01-30

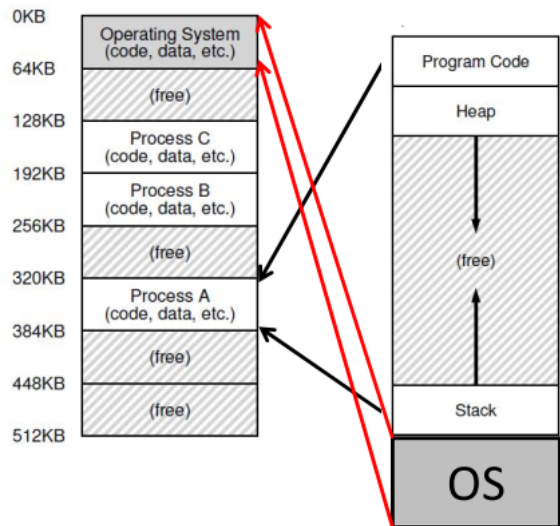
CSE 4733/6733

└ A subtle point: what is the address space of the OS?



# Image: OS memory

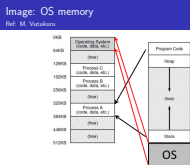
Ref: M. Vutukuru



2023-01-30

CSE 4733/6733

Image: OS memory



# How does the OS allocate memory?

Ref: M. Vutukuru

- OS needs memory for its data structures.
- For large allocations, OS allocates a page.
- For smaller allocations, OS uses various memory allocation algorithms (more later) – Cannot use libc and malloc in the kernel!

2023-01-30

CSE 4733/6733

└ How does the OS allocate memory?

How does the OS allocate memory?

Ref: M. Vutukuru

- OS needs memory for its data structures.
- For large allocations, OS allocates a page.
- For smaller allocations, OS uses various memory allocation algorithms (more later) – Cannot use libc and malloc in the kernel!

# References I



Randall Hyde.  
*The Art of Assembly Language*.  
No Starch Press, USA, 2nd edition, 2010.



J. McCarthy.  
Memorandum to p. m. morse proposing time sharing (1959), 1996.



J. Markoff.  
John mccarthy; pioneered interactive computing, ai, 2011.

2023-01-30

CSE 4733/6733

## References

-  Randall Hyde.  
*The Art of Assembly Language*.  
No Starch Press, USA, 2nd edition, 2010.
-  J. McCarthy.  
Memorandum to p. m. morse proposing time sharing (1959), 1996.
-  J. Markoff.  
John mccarthy; pioneered interactive computing, ai, 2011.