Operating Systems

IV. Memory Management

Ludovic Apvrille
ludovic.apvrille@telecom-paris.fr
Eurecom, office 470

perso.telecom-paris.fr/apvrille/OS/

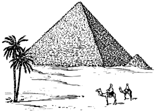
Memory Allocation in C Programs

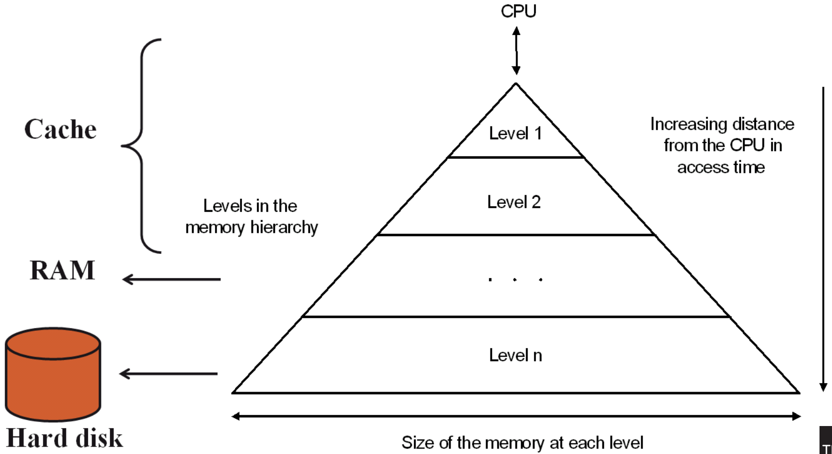
```
int a;

int funnyAllocation(char *buf, int b) {
  a = 5;
  b = b +1;
  strcpy(buf, "hello");

  return 7;
}

int main( int argc, char*argv[] ) {
  int b = 3;

  char *buf = (char *) ( malloc(sizeof(char) * 20));

  int returned = funnyAllocation(buf, b);

  printf("The returned value is: %d\n", returned);
  printf("The value of b is: %d\n", b);
  printf("The content of buf is: %s\n", buf);
}
```

Memory Allocation in C Programs (Cont.)

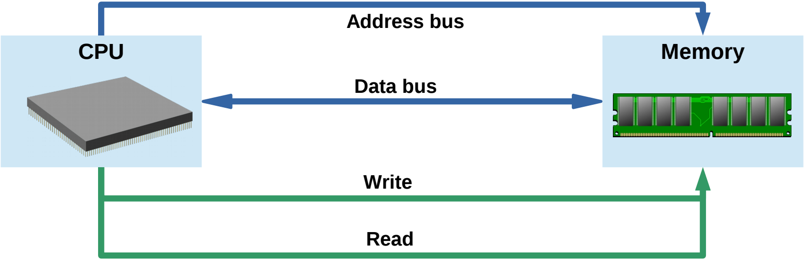
```
$ gcc -Wall -o procmem procmem.c

$ ./procmem
The returned value is: 7
The value of b is: 3
The content of buf is: hello
```

Memory Hierarchy

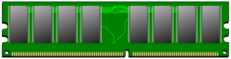




Memory Access



Memory Protection



Goal

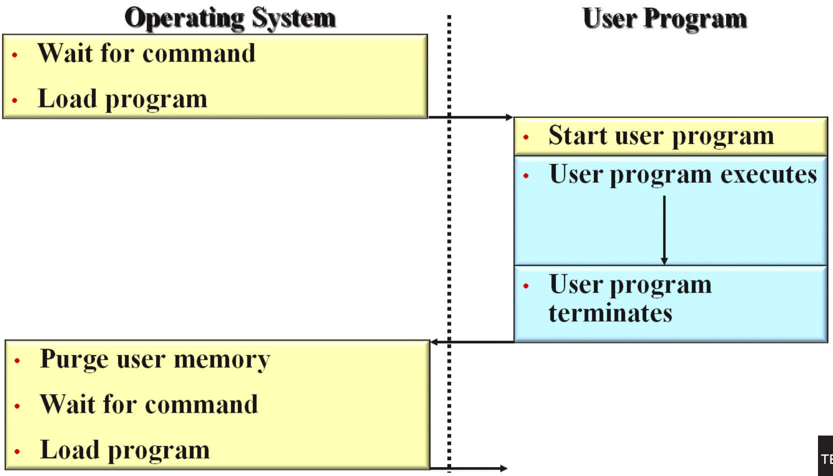
Prevent a process to access unauthorized memory

- Memory used by other processes
- Memory used by the Operating System
  - Interrupt vector, interrupt service routines, kernel, services, etc.

Mechanisms

Dual Mode and **MMU** - Memory Management Unit

Monoprogramming



Multiprogramming: Issues

Process Admittance

- OS estimates the required memory and allocates it

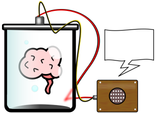
Dynamic Allocation

- A process may request additional memory space
- A process may release part of its memory space

Process termination

- OS must release all the allocated memory

Memory Allocation



Main issues

- Keep track of memory allocations
- Return requested memory chunks as fast as possible
- Avoid fragmentation

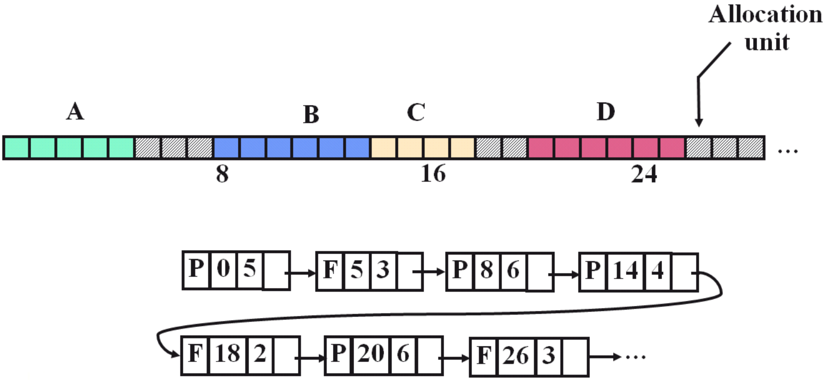
Allocation unit

- Smallest amount of continuous memory managed by the OS
  - From a few bytes to several KBytes
    - Impact of this size?

Keeping track of allocation units

- Linked Lists

Linked Lists



Allocation Algorithms

First Fit

The system scans the list along until a large enough continuous allocation is found

Next Fit

- Scanning begins at the last position where a free block has been found
- Performs slightly worse than First Fit

Best Fit

- Scans all the list and takes the smallest free block that is adequate
- Performs worse than First Fit!
  - Can you guess why?

Allocation Algorithms (Cont.)

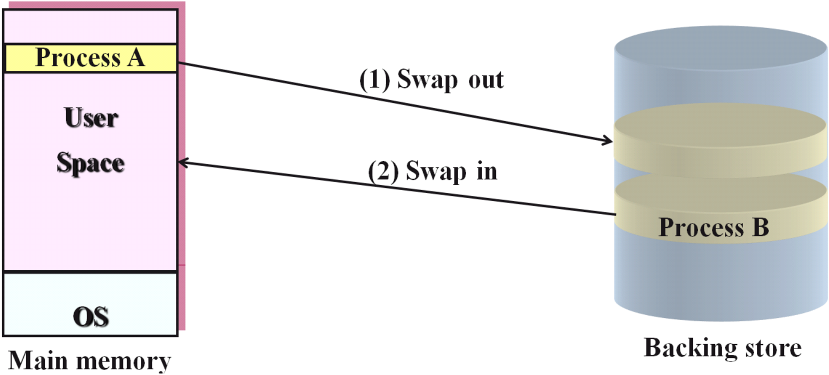
Worse Fit

- Searches for the largest free block
- Not very good either!

Quick Fit

- Separate lists for most usual size
- Additional complexity of memory management
  - Merging is expensive
- But very quick search!

Swapping: Principle



When to Swap?

Swapping is expensive!

- How much time at least is necessary to swap in a 1 GB process (e.g., Firefox) with a transfer rate of 500 MB / second (Typical transfer rate for a (very good) ssd)
- May have to perform a swap out before!

Swap out

- When?
  - Memory occupied over *threshold*
  - A memory allocation request fails
- Which process to swap out?
  - Recently executed processes (RR scheduling)
  - Processes with lower priorities (Priority-based scheduling)

Swap in

- When a process is ready to execute
  - I/O completed
- When a large amount of memory freed

Logical vs. Physical Address Space

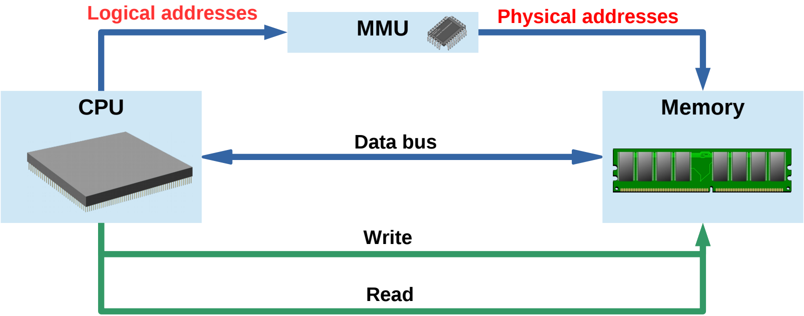
At compilation time, the exact memory location of a program may not be known $\rightarrow$ Virtual Memory

$\Rightarrow$ Two address spaces

- Virtual address space vs. physical address space
- **Logical** / **virtual** address: address used at CPU level (i.e., addresses generated at compilation time)
- **Physical** address: physical address of the RAM

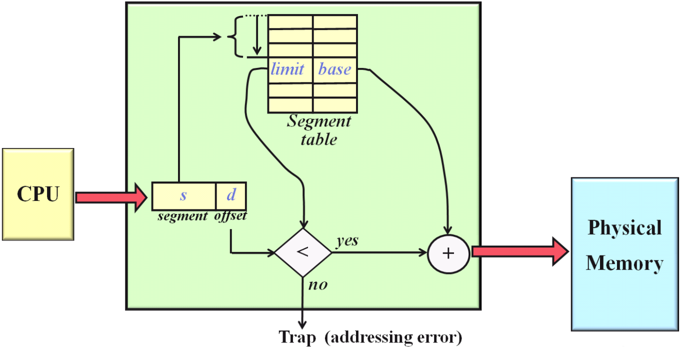
Address binding (virtual $\rightarrow$ physical) done at execution time: **Memory Management Unit**

Memory Management Unit



Segmentation of Memory

- Segment = logical memory unit of variable length
- Virtual segments mapped to physical memory segments
- Memory address = segment number + an address within the segment (= *offset*)

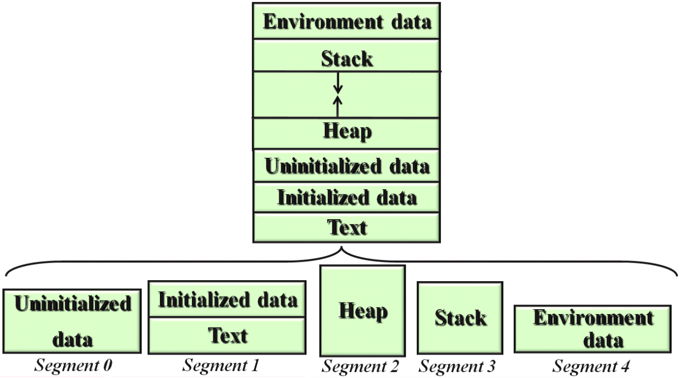


Segmentation of Processes

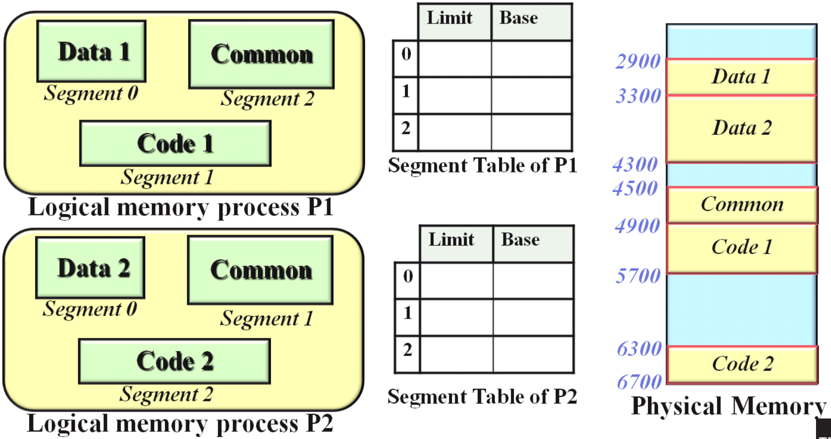
A process is a collection of different types of data

- Code, stack, heap, etc.

→ Use of several segments per Process



Data Sharing with Segmentation



Limitations of Segmentation

Fragmentation

Solution: Using algorithms to select segments (e.g., best-fit, first-fit, etc.)

Segment expansion is costly

If a process allocates more space in a segment and this segment cannot be expanded, and there is free memory available elsewhere
$\rightarrow$ **memory segment must be moved**

1. Process is blocked
2. OS makes a memory copy $\rightarrow$ segment is moved to another location
3. Process is unblocked

$\Rightarrow$ **Paging!**

Basics of Paging

Paging allows the logical address space of a process to be non contiguous in physical memory

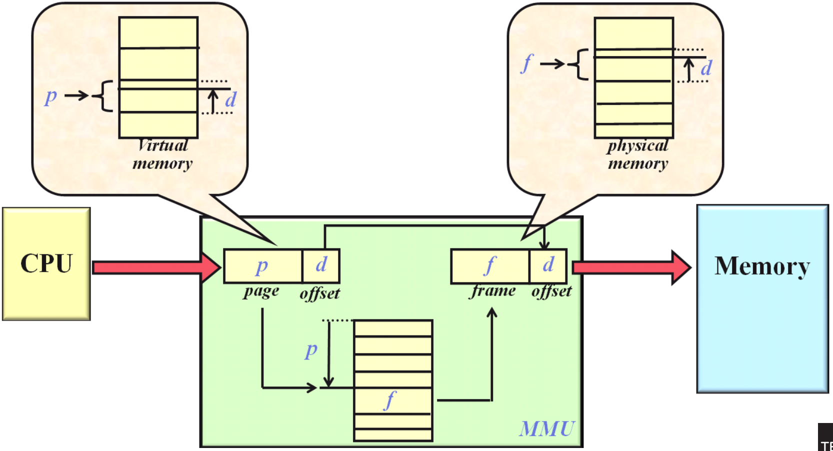
Physical memory

- All physical memory is cut into fixed-size blocks
- Physical memory includes swap partitions
- Logical memory: **page**
- Physical memory: **frame**

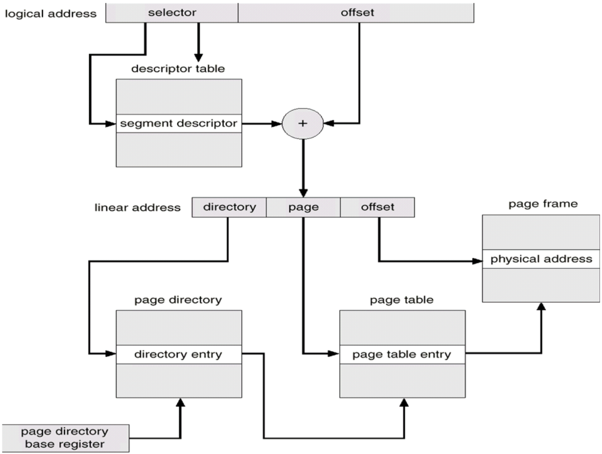
Virtual memory

- Address divided into two parts
  - Page number (p)
  - Page offset (d)

MMU with Paging



Combining Segmentation and Paging



Intel 80386 Address Translation

Segmentation and Page Faults

Memory protection

- Process switching: the OS updates the address table
- MMU detects addresses having no correspondence → **trap**

Reasons for segment / page faults

- **The address is invalid i.e. outside of the process address space**
  - Process is stopped (*segmentation fault*)
- **Segment / page has been swapped out**
  - The OS must make a *swap in* operation
    - A segment / page must first be swapped out if memory is full → *Page Replacement Algorithm*

Page Fault Replacement Algorithms

Issue: **the page-fault rate should be as low as possible**

- FIFO Page Replacement
- Optimal Page Replacement
- LRU Page Replacement
- LRU Approximation Page Replacement
- Counting-Based Page Replacement
- . . . : ongoing research work on this issue

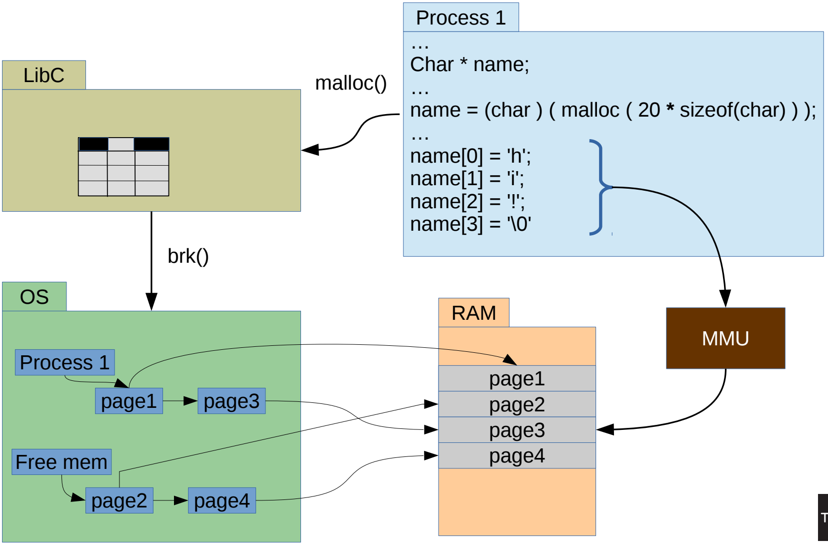
Hardware vs. OS Support

Might be system-dependent!

Mechanism	**Hardware or OS?**
Address translation	Hardware
Segment / page allocation	OS
MMU configuration (choice of active tables, etc.)	OS
Segment / page fault	Hardware detection, OS management

Finally:
Can you tell what are the main interests of MMUs for OS?

Use of Memory in Programs



Windows and Linux

Segments / Pages

- Linux and Windows: only pages for user processes
- Many Unix use both techniques

Copy and Write (*fork()*)

Frame is first shared. If a write operation is performed in the frame, the frame is duplicated

Background daemon

Invoked periodically: Page flushing, freeing unused memory

Memory mapped Files

A file can be mapped onto memory

Windows and Linux (Cont.)

Data structures to describe process space

Windows	**Linux**
Tree structure, each node is called a Virtual Address Descriptor	Linked lists (just like most UNIX) of *vm_area_structs*

OS vs. users process virtual address spaces (x86, 32-bit mode)

- Higher part: kernel code, and lower part: user code
- Linux: 3GB for the process, 1GB for the kernel
- Windows: 2GB for both
- When switching processes, upper part remains the same

Windows and Linux (Cont.)

OS vs. users process virtual address spaces (x86, 64-bit mode)

Windows	Linux
■ Support since March 2005 (*Windows XP Professional x64 Edition*). ■ User processes/OS: 128 TB of virtual address space (Since Windows 8.1)	■ Since kernel 2.4 (2001) ■ User processes: 128 TB of virtual address space

Windows and Linux (Cont.)

Windows: Page replacement

Clock algorithm: Circular list of pages in memory, with the "hand" (iterator) pointing to the oldest page in the list

Linux: Page replacement

- Linux 2.2: **NRU** (Not Recently Used)
  - OS Scans through memory and evicts every page that wasn't accessed since the last scan
- Since kernel 2.4: **LRU** (Improved in 2.6: "CLOCK-PRO")
  - Counter is increased when the page is referenced
  - Counter is divided by 2 when it was not referenced
- kswapd
  - Awakes periodically (e.g., every 1 sec.)
  - Frees memory if enough is not available