Operating Systems

IV. Memory Management

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

http://soc.eurecom.fr/OS/
@OS_Eurecom
Outline

Basics of Memory Management
Hardware Architecture
Monoprogramming
Multiprogramming

Main Mechanisms

What about Windows and Linux?
Memory Hierarchy

- Cache
- RAM
- Hard disk

Levels in the memory hierarchy:
- Level 1
- Level 2
- . . .
- Level n

Size of the memory at each level:
- Increasing distance from the CPU in access time
Memory Access

CPU

Address bus

Data bus

Memory

Write

Read
Memory Protection

Goal
Prevent a process to access unauthorized memory
  ▶ Memory used by other processes
  ▶ Memory used by Operating System
    ▶ Interrupt vector, interrupt service routines, kernel, services, etc.

Mechanisms
Dual Mode and MMU - Memory Management Unit
Monoprocessor Programming

Operating System
- Wait for command
- Load program

User Program
- Start user program
- User program executes
- User program terminates

- Purge user memory
- Wait for command
- Load program
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

A  B  C  D
8  16  24

Allocation unit

P 0 5  F 5 3  P 8 6  P 1 4  4
F 1 8 2  P 2 0 6  F 2 6  3

...
**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!
Outline

Basics of Memory Management

Main Mechanisms
- Swapping
- Virtual Memory
- Segmentation and Paging

What about Windows and Linux?
Swapping: Principle

(1) Swap out
(2) Swap in
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
At compilation time, the exact memory location of a program may not be known → Virtual Memory

⇒ 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 → physical) done at execution time: Memory Management Unit
Memory Management Unit

CPU

MMU

Memory

Data bus

Logical addresses

Physical addresses

Write

Read

Basics of Memory Management
Main Mechanisms
What about Windows and Linux?
Swapping
Virtual Memory
Segmentation and Paging
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)

![Diagram of segmentation and offset](image)
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

Logical memory process P1:
- **Data 1** (Segment 0)
- **Common** (Segment 2)
- **Code 1** (Segment 1)

Logical memory process P2:
- **Data 2** (Segment 0)
- **Common** (Segment 1)
- **Code 2** (Segment 2)

Segment Table of P1:
- Limit 0, Base
- Limit 1, Base
- Limit 2, Base

Segment Table of P2:
- Limit 0, Base
- Limit 1, Base
- Limit 2, Base

Physical Memory:
- 2900: Data 1
- 3300: Data 2
- 4300: Common
- 4900: Code 1
- 5700: Code 2
- 6300: Code 2
- 6700: Code 2
Limitations of Segmentation

Fragmentation

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

Cost of segment expansion!

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

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

⇒ 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!

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Hardware or OS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address translation</td>
<td>Hardware</td>
</tr>
<tr>
<td>Segment / page allocation</td>
<td>OS</td>
</tr>
<tr>
<td>MMU configuration (choice of active tables, etc.)</td>
<td>OS</td>
</tr>
<tr>
<td>Segment / page fault</td>
<td>Hardware detection, OS management</td>
</tr>
</tbody>
</table>

Finally:
Can you tell what are the main interests of MMUs for OS?
Use of Memory in Programs

```c
Char * name;
...
name = (char *) ( malloc ( 20 * sizeof(char) ) );
...
name[0] = 'h';
name[1] = 'i';
name[2] = '!
name[3] = '\0'
```
Outline

Basics of Memory Management
Main Mechanisms
What about Windows and Linux?
Windows and Linux

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

Copy and Write (\texttt{fork()})
- Frame is first shared
- If a write operation is performed, 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

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree structure, each node is called a Virtual Address Descriptor</td>
<td>Linked lists (just like most UNIX) of <code>vm_area_structs</code></td>
</tr>
</tbody>
</table>

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)**

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Support since March 2005 (Windows XP Professional x64 Edition).</td>
<td>• Since kernel 2.4 (2001)</td>
</tr>
<tr>
<td>• User processes/OS: 128 TB of virtual address space (Since Windows 8.1)</td>
<td>• User processes: 128 TB of virtual address space</td>
</tr>
</tbody>
</table>
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