**19**  Virtual Memory

# Creation, Propagation, and Destruction of Addresses

1 ; ex1901.a Creation, propagation, and destruction of addresses

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | ld | | r0, | a | ; | address | now in r0 |
| 3 | st | | r0, | b | ; | address | now at b |
| 4 | ld | | r1, | c |  |  |  |
| 5 | st | | r1, | a | ; | address | at a overlaid with a constant |
| 6 |  | halt | |  |  |  | |
| 7 | a: | .word | | d | ; | assembled to 16-bit address of d | |
| 8 | b: | .word | | 3 |  |  | |
| 9 | c: | .word | | 5 |  |  | |
| 10 | d: | .word | | 17 | ; | the address of this word is 0008 | |

*Observation*: On the LCC, once a program starts executing, it cannot stop, move to a new location, and resume executing.

Why?

Because address adjustment required by the new load point cannot be performed because where the addresses are is unknown. The A entries indicate where the addresses are ONLY before execution starts.

# Memory Fragmentation Problem

# Suppose word processing program terminates:

|  |
| --- |
| Operating system |
| Malware program |
| Word processing program |
| Web-surfing program |
| Fragment |

Game playing program cannot fit into either fragment although the total amount of available memory is sufficient.

Game-playing program cannot fit here

Game-playing program cannot fit here

Fragment

Web-surfing program

Fragment

Malware program

Operating system

Simple Paging

|  |
| --- |
| Page 0 |
| Page 1 |
| Page 2 |
| Page 3 |

0

0FFF

1000

1FFF

2000

2FFF3000

3FFF

Imaginary line

Page Address Range (hex) 0 0000 to 0FFF

1 1000 to 1FFF

2 2000 to 2FFF

3 3000 to 3FFF

16-bit address

2 1 3 C

Page Displacement

Number Within a 4K Page

Main Memory

0

|  |
| --- |
| Frame 0 |
| Frame 1 |
| Frame 2 |
| .  .  . |
| Frame F |

0FFF

1000

1FFF

2000

2FFF

EFFF F000

FFFF

Frame Main Memory

0

|  |
| --- |
| Operating System |
| Operating System |
| Page 2 |
|  |
| Page 0 |
| Page 3 |
| Page 1 |
| .  .  . |

1

Page tables kept here

2

3

4

5

6

Page Table

Page Frame Number Number

0

|  |
| --- |
| 4 |
| 6 |
| 2 |
| 5 |

1

2

3

Memory Unit

Main Memory

DAT Unit

Physical Address

Page Table

Page Table

ptar

CPU

Logical Address

Contents

Dynamic Address Translation

Use left hex digit as index into page table

logical address

Page Table

0 0 0 0

4

5

4 0 0 0

from program 6

2

physical address

logical address

Page Table

1 0 0 0

5

6 0 0 0

2

6

4

from program

physical address

Computation of physical address requires a *substitution*, which can be performed quickly.

# Associative Memory

The obvious solution to the problem with paging—two reads performed every time the CPU fetches an item from memory—is to keep a copy of the page table in a local memory area within the DAT unit:

Memory System Black Box

Main Memory

DAT Unit

Physical Address

Page Table

Page Table

Local Memory

Page Table

CPU

Logical Address

Contents

Problem if Subset of Page Table in DAT Unit

Page Table

Page Frame Number Number

0

|  |
| --- |
| 4 |
| 6 |
| 2 |
| 5 |

1

2

3

DAT Unit

Local Memory

0

2

1

5

frame number for page 3

Solution

DAT Unit

To page table

ptar

Associative Memory

Page Frame Valid Number Number Bit

Page Number

Frame Number

Match

|  |  |  |
| --- | --- | --- |
| 2 | 2 | 1 |
| 3 | 5 | 1 |

# Virtual Memory: Demand Paging

Page

Number Frame Number Valid Bit 0

|  |  |
| --- | --- |
| 0 | 0 |
| 0 | 0 |
| 2 | 1 |
| 5 | 1 |

1

2

3

# Page Replacement Policies for Demand Paging

* FIFO
* LRU
* NUR

Page Size Considerations

Small page size

Page 0

Page 1

Page 2

Page 3

Big page size

Page 0

|

|

Page 1

# Supervisor/User Modes

* Machine instructions that a user program is not allowed to execute *privileged instructions*.
* Privileged instructions are implemented by means of two CPU running modes: a supervisor mode and a user mode.

# Memory Protection

# Paging provides a simple memory protection mechanism.

# Segmentation with Paging

* To solve the problem of page tables that are excessively large because they map unused gaps in the logical address space, we can divide our program into functional segments.
* Each page table would be sufficiently large to map only its corresponding segment.
* The unused gap in the address space would not be represented by any page table. Thus, the combined size of all the page tables would be minimized.

Advantages of Segmentation with Paging

1. It allows the specification of a privilege level (the level of memory protection) and an access mode (i.e, permissible accesses, such as read/write, read-only, execute, etc.) for each segment that is tailored to that segment.
2. Because each segment has, in effect, its own virtual memory, it simplifies the mechanics of dynamically increasing or decreasing the size of a segment.
3. Because segments are logical units of a program, sharing of segments among users is simplified.