Main Memory (Part II)

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Reminder

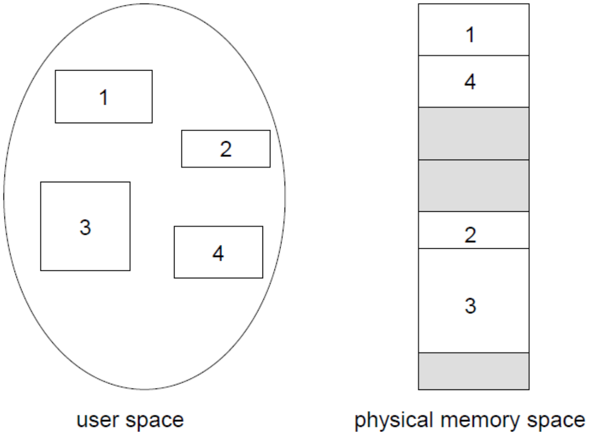
Reminder (1/3)

- External fragmentation vs. internal fragmentation

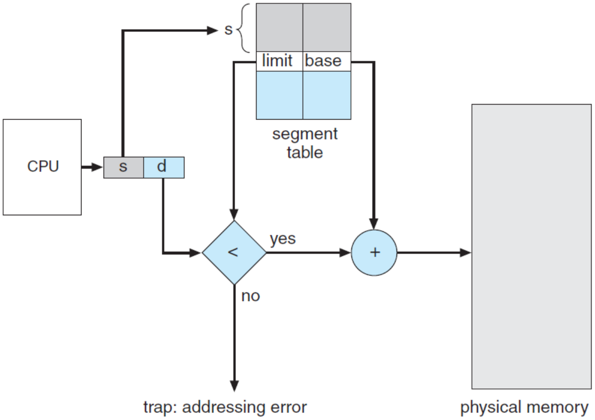
- Compaction: shuffle memory contents to place all free memory together in one large block.

- Other solutions:
  - Segmentation
  - Paging

Reminder (2/3)



Reminder (3/3)


s
limit
base
segment table
CPU
s
d
<
yes
no
+
trap: addressing error
physical memory

Paging

Paging vs. Segmentation

- Segmentation and paging, both, permit the physical address space of a process to be noncontiguous.

Paging vs. Segmentation

- Segmentation and paging, both, permit the physical address space of a process to be noncontiguous.
- Paging avoids external fragmentation and the need for compaction, whereas segmentation does not.

Paging (1/2)

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks

Paging (1/2)

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks

- Divide physical memory into fixed-sized blocks called frames.
  - Size is power of 2, between 512 bytes and 16 Mbytes.

Paging (1/2)

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks

- Divide physical memory into fixed-sized blocks called frames.
  - Size is power of 2, between 512 bytes and 16 Mbytes.

- Divide logical memory into blocks of same size called pages.

Paging (2/2)

- Keep track of all free frames.

Paging (2/2)

- Keep track of all free frames.
- To run a program of size N pages, need to find N free frames and load program.

Paging (2/2)

- Keep track of all free frames.
- To run a program of size N pages, need to find N free frames and load program.
- Set up a page table to translate logical to physical addresses.

Paging (2/2)

- Keep track of all free frames.
- To run a program of size N pages, need to find N free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Still have internal fragmentation.

Address Translation Scheme

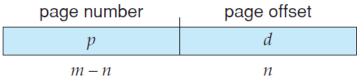
- Address generated by CPU (logical address) is divided into two parts:

Address Translation Scheme

- Address generated by CPU (logical address) is divided into two parts:
- Page number (p): used as an index into a page table that contains base address of each page in physical memory.

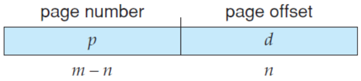
Address Translation Scheme

- Address generated by CPU (logical address) is divided into two parts:
- Page number (p): used as an index into a page table that contains base address of each page in physical memory.
- Page offset (d): combined with base address to define the physical memory address that is sent to the memory unit.



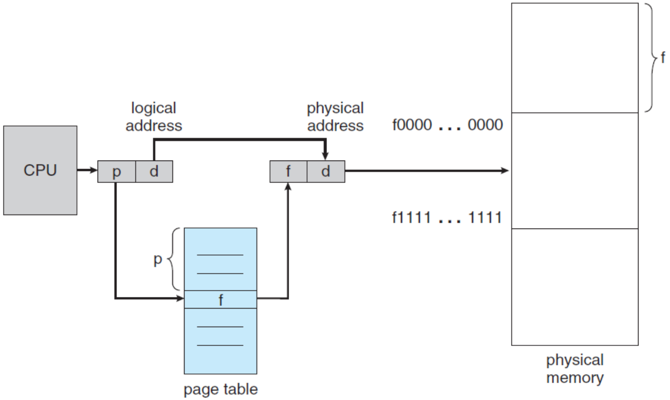
Address Translation Scheme

- Address generated by CPU (logical address) is divided into two parts:
- Page number (p): used as an index into a page table that contains base address of each page in physical memory.
- Page offset (d): combined with base address to define the physical memory address that is sent to the memory unit.

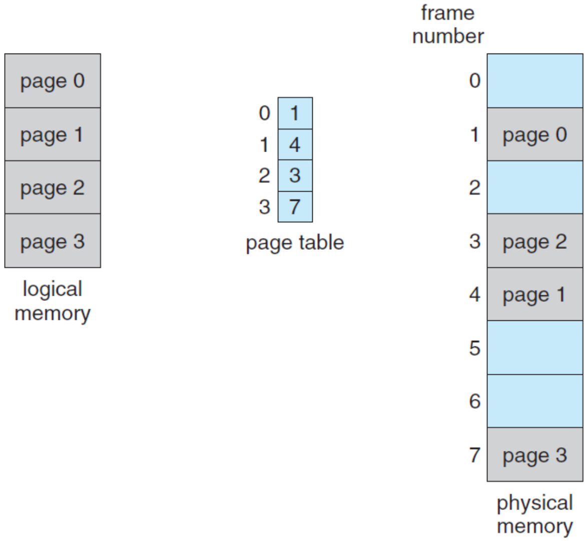


- For given logical address space 2^m and page size 2^n.

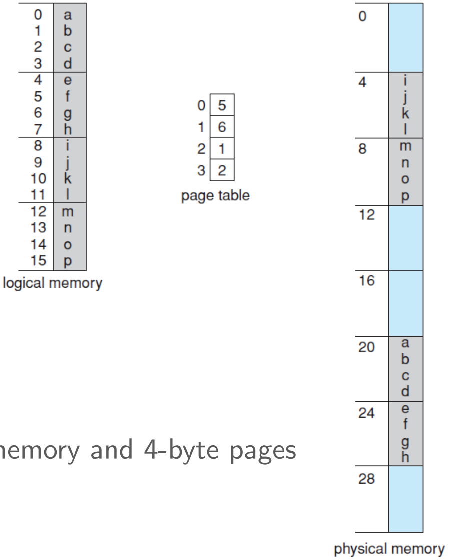
Paging Hardware



Paging Model of Logical and Physical Memory

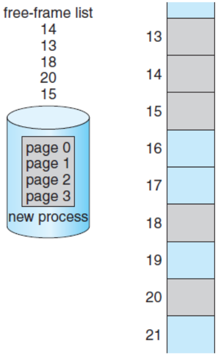


Paging Example

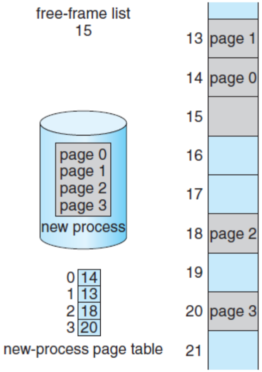


- $n = 2$ and $m = 4$, 32-byte memory and 4-byte pages

Free Frames



before allocation



after allocation

Paging Example - Internal Fragmentation

- Page size = 2048 bytes
- Process size = 72766 bytes

Paging Example - Internal Fragmentation

- Page size $= 2048$ bytes
- Process size $= 72766$ bytes
- $\frac{72766}{2048} = 35$ pages $+ 1086$ bytes

Paging Example - Internal Fragmentation

- Page size = 2048 bytes
- Process size = 72766 bytes
- $\frac{72766}{2048} = 35$ pages + 1086 bytes
- Internal fragmentation: 2048 - 1086 = 962 bytes

Paging Example - Internal Fragmentation

- Page size $= 2048$ bytes
- Process size $= 72766$ bytes
- $\frac{72766}{2048} = 35$ pages $+ 1086$ bytes
- Internal fragmentation: 2048 - 1086 $= 962$ bytes
- Worst case fragmentation $=$ 1 frame - 1 byte

Paging Example - Internal Fragmentation

- Page size = 2048 bytes
- Process size = 72766 bytes
- $\frac{72766}{2048}$ = 35 pages + 1086 bytes
- Internal fragmentation: 2048 - 1086 = 962 bytes
- Worst case fragmentation = 1 frame - 1 byte
- On average fragmentation = $\frac{1}{2}$ frame size

Small Page Size vs. Big Page Size

- On average fragmentation $= \frac{1}{2}$ page size, hence, small page sizes are desirable.

Small Page Size vs. Big Page Size

- On average fragmentation $= \frac{1}{2}$ page size, hence, small page sizes are desirable.
- Small pages, more overhead is in the page-table, this overhead is reduced as the size of the pages increases.

Small Page Size vs. Big Page Size

- On average fragmentation $= \frac{1}{2}$ page size, hence, small page sizes are desirable.
- Small pages, more overhead is in the page-table, this overhead is reduced as the size of the pages increases.
- Disk I/O is more efficient when the amount data being transferred is larger (e.g., big pages).

Small Page Size vs. Big Page Size

- On average fragmentation $= \frac{1}{2}$ page size, hence, small page sizes are desirable.
- Small pages, more overhead is in the page-table, this overhead is reduced as the size of the pages increases.
- Disk I/O is more efficient when the amount data being transferred is larger (e.g., big pages).
- Pages typically are between 4 KB and 8 KB in size.

```
getconf PAGESIZE
```

Page Table Implementation

Page Table

- Page table is kept in main memory.

Page Table

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.

Page Table

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses.

Page Table

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses.
    - One for the page table and one for the data/instruction.

Translation Look-aside Buffers (TLB)

- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs).

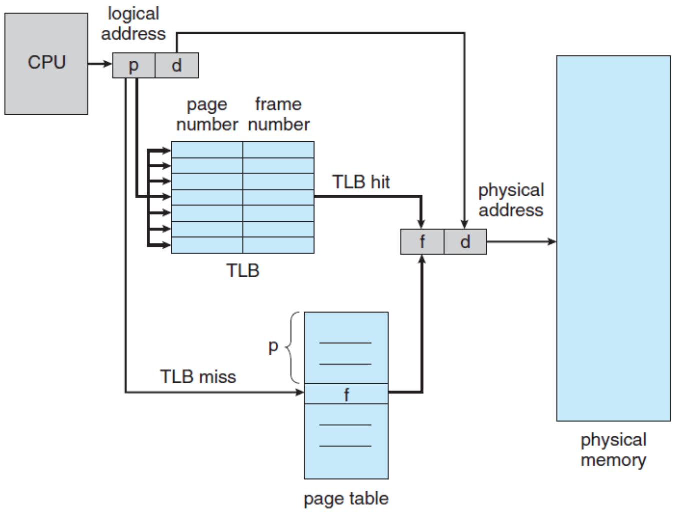
Associative Memory

- Associative memory: parallel search

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get *frame#* out
  - Otherwise, get *frame#* from page table in memory

Paging Hardware With TLB



Effective Access Time

- Hit ratio: percentage of times that a page number is found in the TLB.

Effective Access Time

- Hit ratio: percentage of times that a page number is found in the TLB.

- Effective Access Time (EAT)
  - α: memory access latency
  - h: hit ratio
  - $EAT = h \times \alpha + (1 - h) \times 2\alpha$

Effective Access Time

- Hit ratio: percentage of times that a page number is found in the TLB.

- Effective Access Time (EAT)
  - α: memory access latency
  - h: hit ratio
  - $EAT = h \times \alpha + (1 - h) \times 2\alpha$

- $h = 80\%, \alpha = 100ns \Rightarrow EAT = 0.80 \times 100 + 0.20 \times 200 = 120ns$

Effective Access Time

- Hit ratio: percentage of times that a page number is found in the TLB.

- Effective Access Time (EAT)
  - α: memory access latency
  - h: hit ratio
  - $EAT = h \times \alpha + (1 - h) \times 2\alpha$

- $h = 80\%, \alpha = 100ns \Rightarrow EAT = 0.80 \times 100 + 0.20 \times 200 = 120ns$

- $h = 99\%, \alpha = 100ns \Rightarrow EAT = 0.99 \times 100 + 0.01 \times 200 = 101ns$

More About TLB

- Some TLBs store address-space identifiers (ASIDs) in each TLB entry
    - Uniquely identifies each process to provide address-space protection for that process.
    - Otherwise, need to flush at every context switch.

More About TLB

- Some TLBs store address-space identifiers (ASIDs) in each TLB entry
  - Uniquely identifies each process to provide address-space protection for that process.
  - Otherwise, need to flush at every context switch.

- TLBs typically small (64 to 1,024 entries)

More About TLB

- Some TLBs store address-space identifiers (ASIDs) in each TLB entry
  - Uniquely identifies each process to provide address-space protection for that process.
  - Otherwise, need to flush at every context switch.

- TLBs typically small (64 to 1,024 entries)

- On a TLB miss, value is loaded into the TLB for faster access next time.
  - Replacement policies must be considered.

Memory Protection

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.

- Valid-invalid bit attached to each entry in the page table:
  - Valid indicates that the associated page is in the process logical address space, and is thus a legal page.
  - Invalid indicates that the page is not in the process logical address space.

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.

- Valid-invalid bit attached to each entry in the page table:
  - Valid indicates that the associated page is in the process logical address space, and is thus a legal page.
  - Invalid indicates that the page is not in the process logical address space.
  - Or use page-table length register (PTLR).

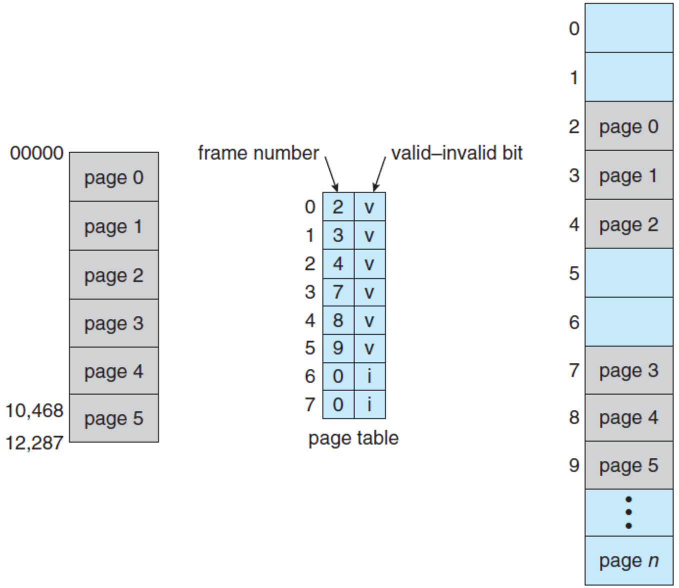
Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.

- Valid-invalid bit attached to each entry in the page table:
  - Valid indicates that the associated page is in the process logical address space, and is thus a legal page.
  - Invalid indicates that the page is not in the process logical address space.
  - Or use page-table length register (PTLR).

- Any violations result in a trap to the kernel.

Valid/Invalid Bit In A Page Table



Shared Pages

Shared Pages

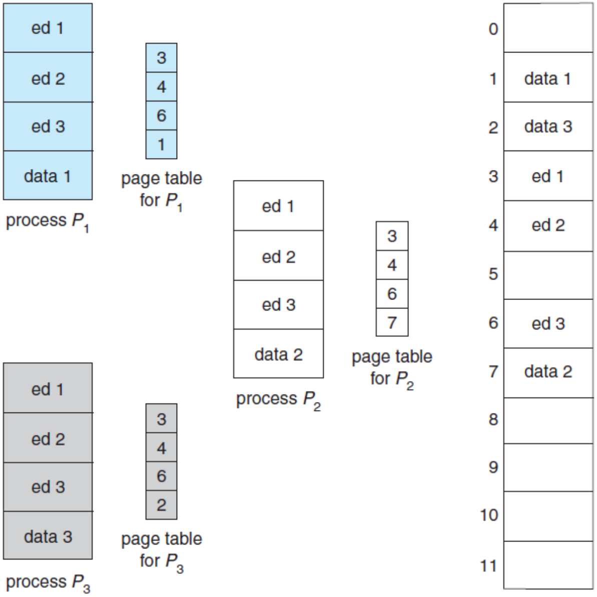
- Shared code
  - One copy of read-only (reentrant) code shared among processes (e.g., text editors).
  - Similar to multiple threads sharing the same process space.

Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (e.g., text editors).
  - Similar to multiple threads sharing the same process space.

- Private code and data
  - Each process keeps a separate copy of the code and data.
  - The pages for the private code and data can appear anywhere in the logical address space.

Shared Pages Example



Structure of the Page Table

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.
- Consider a 32-bit logical address space as on modern computers:

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.

- Consider a 32-bit logical address space as on modern computers:
  - Page size of $4KB = 2^{12}$.

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.
- Consider a 32-bit logical address space as on modern computers:
  - Page size of $4KB = 2^{12}$.
  - Page table would have 1 million entries $(\frac{2^{32}}{2^{12}})$.

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.
- Consider a 32-bit logical address space as on modern computers:
  - Page size of $4KB = 2^{12}$.
  - Page table would have 1 million entries $(\frac{2^{32}}{2^{12}})$.
  - If each entry is $4B$: $4MB$ of physical address space memory for page table alone.

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.
- Consider a 32-bit logical address space as on modern computers:
  - Page size of $4KB = 2^{12}$.
  - Page table would have 1 million entries $(\frac{2^{32}}{2^{12}})$.
  - If each entry is $4B$: $4MB$ of physical address space memory for page table alone.
  - That amount of memory used to cost a lot.

Structure of the Page Table (1/2)

- Memory structures for paging can get huge using straight-forward methods.
- Consider a 32-bit logical address space as on modern computers:
  - Page size of $4KB = 2^{12}$.
  - Page table would have 1 million entries $(\frac{2^{32}}{2^{12}})$.
  - If each entry is $4B$: $4MB$ of physical address space memory for page table alone.
  - That amount of memory used to cost a lot.
  - Don't want to allocate that contiguously in main memory.

Structure of the Page Table (2/2)

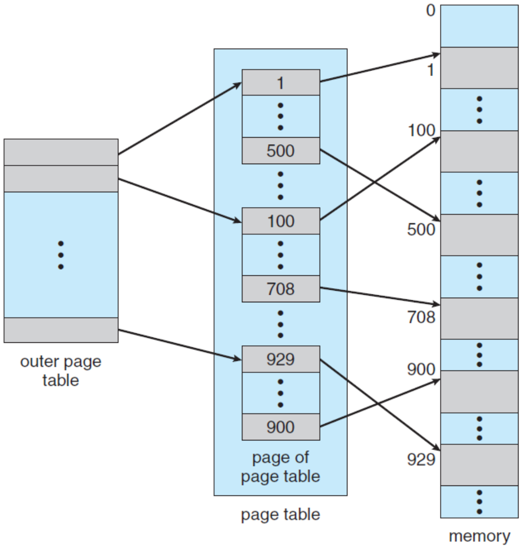
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Paging

Hierarchical Page Tables

- Break up the logical address space into multiple page tables.
- A simple technique is a two-level page table.
- We then page the page table.

Two-Level Page-Table Scheme



Two-Level Paging Example

- A logical address, on 32-bit machine with 1K page size, is divided:

Two-Level Paging Example

- A logical address, on 32-bit machine with 1K page size, is divided:
  - A page number consisting of 22 bits.

Two-Level Paging Example

- A logical address, on 32-bit machine with 1K page size, is divided:
  - A page number consisting of 22 bits.
  - A page offset consisting of 10 bits.

Two-Level Paging Example

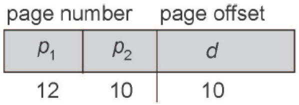
- A logical address, on 32-bit machine with 1K page size, is divided:
  - A page number consisting of 22 bits.
  - A page offset consisting of 10 bits.
- Since the page table is paged, the page number is divided into:
  - A 12-bit page number.
  - A 10-bit page offset.

Two-Level Paging Example

- A logical address, on 32-bit machine with 1K page size, is divided:
  - A page number consisting of 22 bits.
  - A page offset consisting of 10 bits.

- Since the page table is paged, the page number is divided into:
  - A 12-bit page number.
  - A 10-bit page offset.

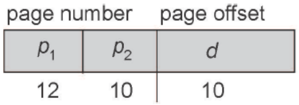
- Thus, a logical address is:



- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table.

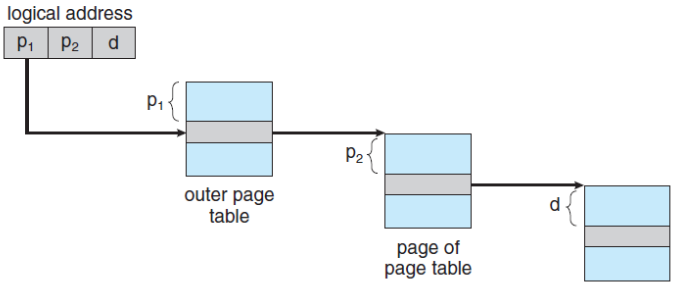
Two-Level Paging Example

- A logical address, on 32-bit machine with 1K page size, is divided:
  - A page number consisting of 22 bits.
  - A page offset consisting of 10 bits.
- Since the page table is paged, the page number is divided into:
  - A 12-bit page number.
  - A 10-bit page offset.
- Thus, a logical address is:



- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table.
- Known as forward-mapped page table.

Address-Translation Scheme



64-bit Logical Address Space

- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})

64-bit Logical Address Space

- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})
  - Then page table has 2^{52} entries

64-bit Logical Address Space

- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10}, 4B entries

64-bit Logical Address Space

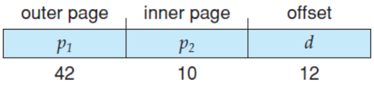
- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10}, 4B entries
  - Outer page table has 2^{42} entries or $2^{44} B$

64-bit Logical Address Space

- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10}, 4B entries
  - Outer page table has 2^{42} entries or $2^{44}B$
  - Address would look like:

64-bit Logical Address Space

- Even two-level paging scheme not sufficient.
- If page size is 4KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10}, 4B entries
  - Outer page table has 2^{42} entries or $2^{44}B$
  - Address would look like:



Three-level Paging Scheme

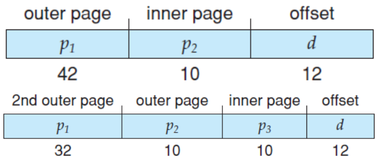
- One solution is to add a 2nd outer page table.

Three-level Paging Scheme

- One solution is to add a 2nd outer page table.
- But in the following example the 2nd outer page table is still 2^{34} bytes in size.

Three-level Paging Scheme

- One solution is to add a 2nd outer page table.
- But in the following example the 2nd outer page table is still 2^{34} bytes in size.
- And possibly 4 memory access to get to one physical memory location.



Hashed Page Tables

Hashed Page Tables (1/2)

- Common in address spaces > 32 bits
- The logical page number is hashed into a page table.
- This page table contains a chain of elements hashing to the same location.

Hashed Page Tables (2/2)

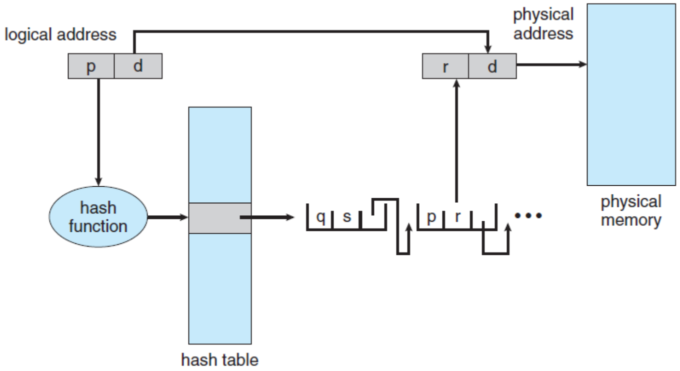
- Each element contains
  1. The logical page number
  2. The value of the mapped page frame
  3. A pointer to the next element

Hashed Page Tables (2/2)

- Each element contains
  1. The logical page number
  2. The value of the mapped page frame
  3. A pointer to the next element

- Logical page numbers are compared in this chain searching for a match.
  - If a match is found, the corresponding physical frame is extracted.

Hashed Page Table Architecture



Inverted Page Tables

Inverted Page Table (1/2)

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.

Inverted Page Table (1/2)

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.
- One entry for each real page of memory.

Inverted Page Table (1/2)

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.
- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Inverted Page Table (2/2)

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.

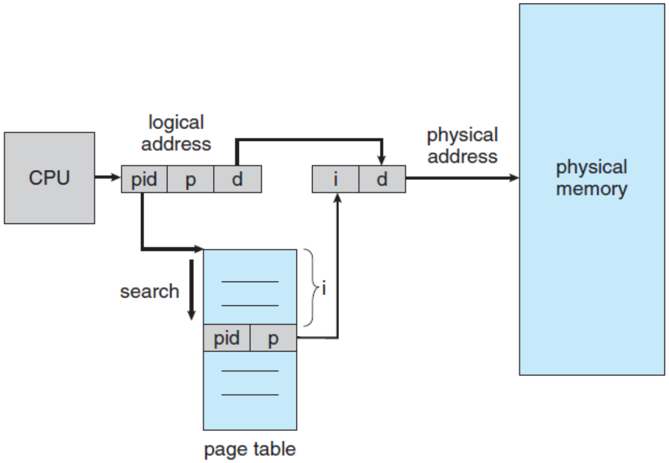
Inverted Page Table (2/2)

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one, or at most a few, page-table entries.

Inverted Page Table (2/2)

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one, or at most a few, page-table entries.
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

Inverted Page Table Architecture



Summary

Summary

- Paging vs. Segmentation

Summary

- Paging vs. Segmentation
- Physical memory: frames, Logical memory: pages

Summary

- Paging vs. Segmentation
- Physical memory: frames, Logical memory: pages
- Page table: translates logical to physical addresses

Summary

- Paging vs. Segmentation
- Physical memory: frames, Logical memory: pages
- Page table: translates logical to physical addresses
- Translation Look-aside Buffer (TLB)

Summary

- Paging vs. Segmentation
- Physical memory: frames, Logical memory: pages
- Page table: translates logical to physical addresses
- Translation Look-aside Buffer (TLB)
- Memory protection: valid-invalid bit

Summary

- Paging vs. Segmentation
- Physical memory: frames, Logical memory: pages
- Page table: translates logical to physical addresses
- Translation Look-aside Buffer (TLB)
- Memory protection: valid-invalid bit
- Page table structure: hierarchical paging, hashed page tables, inverted page tables

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.