

Chapter 12

Paging an Virtual Memory Systems

Paging & Virtual Memory

- **Virtual Memory** - giving the illusion of more physical memory than there really is (via demand paging)
- **Pure Paging** - The total program is kept in memory as sets of (non-contiguous) pages
 - No illusion of virtual memory
- **Demand Paging** - A program's "working set" is kept in memory, reference outside WS causes corresponding code to be retrieved from disk ("page fault")
 - Provides the illusion of virtual memory

CS3204 - Arthur

Paging Systems

- Processes (programs) are divided into fixed size pieces called **Pages**
- Main memory is divided into fixed size partitions called **Blocks (Page Frames)**
- **Pure Paging** - entire program is kept in memory during execution, but pages are not kept in contiguous blocks
- **Demand paging** - only parts of program kept in memory during execution, pages are not kept in contiguous blocks



CS3204 - Arthur

Virtual Versus Physical Addresses

- A *virtual address* is represented as $\langle \text{page}, \text{offset} \rangle$ where the page is determined by dividing each process into fixed size pages, the offset is a number in the range $0 - (\text{page_size} - 1)$.
- Memory is divided into fixed size blocks (or page frames) and accommodates a process' pages. The physical address (PA) then is $(\text{block_number} * \text{page_size} + \text{offset})$.
- In pure paging systems the entire VA space of a process must reside in physical memory during execution, but pages are *not* kept in contiguous blocks.

CS3204 - Arthur

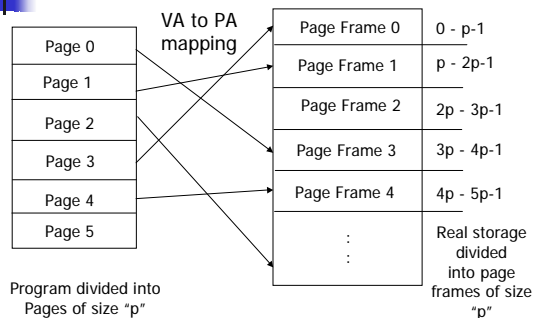
Pure Paging Virtual Addresses...

- VA is determined from the compiled address
- VA has two components:

page number	address in page (or offset or displacement)
-------------	--

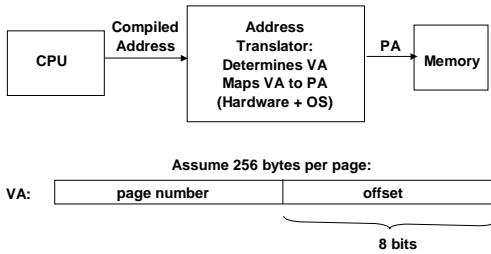
CS3204 - Arthur

Virtual Address to Physical Address Mapping



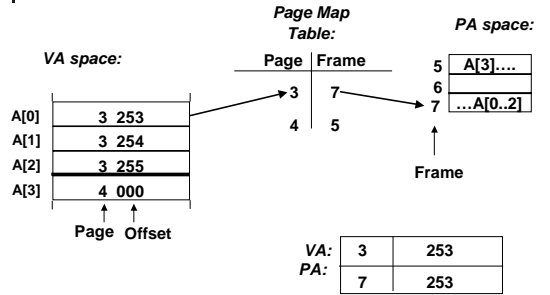
CS3204 - Arthur

Key Idea in Paging Systems



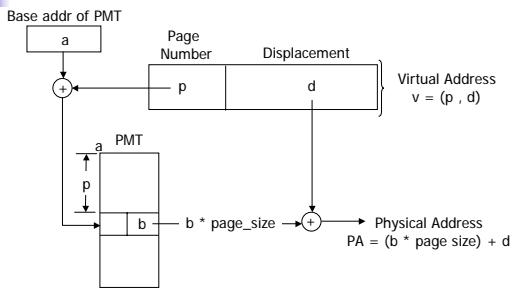
CS3204 - Arthur

Key Idea in Paging Systems...



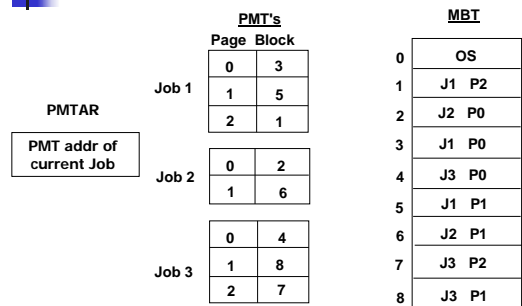
CS3204 - Arthur

Addressing Scheme



CS3204 - Arthur

Paging Mapping Example



CS3204 - Arthur

Page Management

Page Map Table (PMT):

Contains VA page to PA block mapping

Page	Block
0	7
1	2
2	13

1 PMT / job
1 Entry / page

CS3204 - Arthur

Page Management

Page Map Table Address Register (PMTAR):

Length of program in pages (# PMT entries)	Base address of current PMT
--	-----------------------------

Points to PMT for currently executing job

1 PMTAR / System

CS3204 - Arthur

Page Management ...

Memory Block Table (MBT)

Maps each block of main memory either to a process id and page number or to "free"

1 MBT / System
1 Entry / Block

CS3204 - Arthur

Page Management ...

Process (Job) Control Block (PCB)

Contains information about all jobs in the system

Stores: Job Size
Location of PMT

1 PCB / system
1 entry / job

CS3204 - Arthur

Page Addressing - Let's get REAL

VA = < page, offset >

PA = block size * block + offset

Assume:

1 word PMT entries;

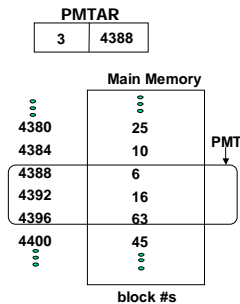
byte addressable MM

Example:

page & block size = 4 K bytes

VA = < 1, 1234 >

PA = $4096 * 16 + 1234$



CS3204 - Arthur

Determining Virtual Address <Page , Offset> from the Compiled Address

Compiled Address (relative to 0) : 18087

Page size: 2K (2048 bytes)
Memory is **byte** addressable

Virtual Address:

Page = Div (Compiled Address, Page Size)
Offset = MOD (Compiled Address, Page Size)

< 8 , 1703 >

DIV => "Shift right", 11 bits ($2048 = 2^{11}$)
MOD => "AND" 21 High-order bits with 0

CS3204 - Arthur

Review Questions

Assume:

2 bytes PMT entries; byte addressable MM
page & block size = 4 K bytes

- 1) What is the maximum size for any program?
- 2) What PA corresponds to compiled address 220945?
- 2) What is the MBT length if MM size is 80M?
(Assume MBT entries are 2 bytes long.)
- 3) What is the PMT length if compiled size = 300K?

CS3204 - Arthur

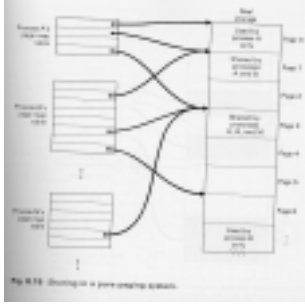
Allocating Pages

```

procedure allocation (int Size) {
    WS
    Word size in bytes
    P
    Page size in Bytes
    Size
    Size of program in bytes
    NPPgm
    Num of pages needed for pgm
    NPPmt
    number of pages needed for PMT
    (1 word / entry)
    NPTot
    Total number of pages needed
    MaxBlocks
    main memory size, in blocks
}
    NPPgm := ceiling( Size / P );
    NPmt := ceiling( (NPPgm * WS) / P );
    NPTot := NPPgm + NPmt;
    If ( NPTot > MaxBlocks )
    Then ERROR
    Else If ( NPTot blocks are not free in MBT )
    Then Add job to HOLDQ;
    Else {
        Allocate pages to blocks;
        Update MBT, PCB;
        Create, initialize PMT;
    }
}
    
```

CS3204 - Arthur

Sharing Pages of Reentrant Code or Data Between Processes



CS3204 - Arthur

Pros/Cons of Paging

Advantages:

- Efficient memory usage
- Simple partition management due to discontinuous loading and fixed partition size
- No compaction necessary
- Easy to share pages

CS3204 - Arthur

Pros/Cons of Paging...

Disadvantages:

- Job Size \leq Memory Size
- Internal fragmentation (half the page size on the average)
- Need special hardware for address translation
- Some main memory space used for PMT's
- Address translation lengthens memory cycle times

CS3204 - Arthur

Demand Paging

Jobs are paged, but not all pages have to be in memory at the same time

VIRTUAL MEMORY

- The operating system creates the illusion of more memory
- Job size can exceed main memory size
- Pages are only brought in when referenced (on demand)
- Often page 0 is loaded initially when a job is scheduled

CS3204 - Arthur

Demand Paging Motivation

	PMT's			MBT	
	Page	Block			
Job 1	0	3	0	OS	
	1	5	1	J1 P2	
	2	1	2	J2 P0	
Job 2	0	2	3	J1 P0	
	1	6	4	J3 P0	
Job 3	0	4	5	J1 P1	
	1		6	J2 P1	
	2				

1. What happens if job 3 references page 1?
2. What does the CPU do while J3P1 is being read?

CS3204 - Arthur

Terminology

Page fault:

Interrupt that arises upon a reference to a page that is not in main memory.

Page swapping:

Replacement of one page in memory by another page in response to a page fault.

CS3204 - Arthur

When a Page Fault Occurs

- Select a page to be removed
- Copy it to disk if it has been changed **
- Update old page's PMT **
- Copy new page to main memory
- Update new page's PMT
- Update MBT **

Thrashing occurs when a job continuously references pages which are not in main memory

CS3204 - Arthur

Demand Page Management

Page Map Table (PMT)

Maps page to blocks
 Status: Pointer to
 Main Memory Block
Indicator
Main/Secondary Memory

File Map Table (FMT)

Maps a job's pages to secondary memory

PMT for the Disk

1 FMT / job

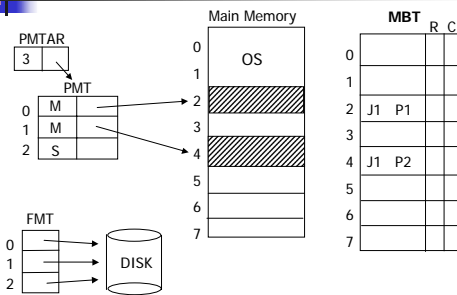
Memory Block Table (MBT)

Maps block to page
 Contains: Job/Page Number
Reference bit
Change bit

1 entry / page

CS3204 - Arthur

Demand Paging Schematic



CS3204 - Arthur

Demand Paging Data Structures

	PMT			FMT		MBT
	Page	Block	In_Mem	Disk		
Job 1	0	3	Yes	Dsk Addr		0 OS
	1	5	Yes	Dsk Addr		1 J1 P2
	2	1	Yes	Dsk Addr		2 J2 P0
Job 2	0	2	Yes	Dsk Addr		3 J1 P0
	1	6	Yes	Dsk Addr		4 J3 P0
Job 3	0	4	Yes	Dsk Addr		5 J1 P1
	1		No	Dsk Addr		6 J2 P1
	2		No	Dsk Addr		7

Referenced Changed

1. What happens if job 3 references page 1?

CS3204 - Arthur

Summary of Data Structures

- Page Map Table (PMT):** Maps page to block
Fields: - page number (which page in memory)
 - In_Memory <--- *New!*
- Memory Block Table (MBT):** Maps block to either process id and page number or to "free"
Fields: <--- *New!*
 - Reference Count
 - Change Bit
- File Map Table (FMT):** Maps a job's pages to secondary memory (like a PMT for the disk) <--- *New!*
 1 FMT / job, 1 entry / page

CS3204 - Arthur

Page Replacement

Now we consider the decision of selecting which page to replace upon a page fault.

Local versus Global Page Replacement

Local Requires that each process remove a page from its own set of allocated blocks

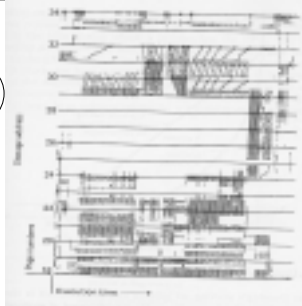
Global A replacement page may be selected from the set of all blocks

CS3204 - Arthur

A Program's Execution Profile

Question:

Does a program need *all* its pages in main memory at all times?



CS3204 - Arthur

The Principle of Locality

At any time, the *locality* of a process is the set of pages that are actively being used together

Spatial There is a high probability that once a location is referenced, the one after it will be accessed in the near future

Sequential code, Array processing, Code within a loop

Temporal A referenced location is likely to be accessed again in the near future

Loop indices, Single data elements

CS3204 - Arthur

More on Locality

Does a linked list help or hurt locality?

Does a recursive function display spatial or temporal locality?

CS3204 - Arthur

Working Set Theory (Formalizes "Locality")

- A process' **working set** is the number of pages currently being referenced during $(t, t+\Delta)$ for some small Δ .
- The working set size is an estimate of degree of locality
- A job should not be scheduled unless there is room for its entire working set
 - Why?

CS3204 - Arthur

Idea Behind Working Set

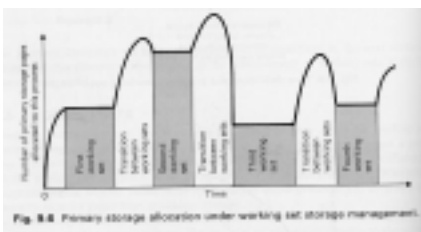


Fig. 8.6 Primary storage allocation under working set storage management.

CS3204 - Arthur

Motivation : Page Replacement Algorithms

Which page replacement rule should we use to give the *minimum* page fault rate?

Page fault rate = # faults / # refs

CS3204 - Arthur

Page Replacement Algorithm: Optimal Replacement

- Replace the page which will not be used for the longest period of time
- Lowest page fault rate of all algorithms
- Requires knowledge of the future

Example:

MM has 3 blocks containing 3,5,2.

Current and future refs:

4, 3, 3, 4, 2, 3, 4, 5, 1, 3, 4

fault

OPT replaces 5

CS3204 - Arthur

Optimal Replacement Algorithm

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												

Page Faults =

Page Fault Rate =

CS3204 - Arthur

Replacement Algorithm: FIFO

- Replace the "oldest" page
- A frequently used page may be swapped out

Belady's Anomaly:

For some page replacement algorithms, the page fault rate may increase as the number of blocks increase

CS3204 - Arthur

FIFO Page Replacement

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												

Page Faults =

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												
Block Number 3												

Page Faults =

CS3204 - Arthur

Replacement Algorithms: Least Recently Used (LRU)

- Uses the recent past as an approximation of the near future
- Stack algorithm
 - Does NOT suffer from Belady's Anomaly
- Hardware / Overhead intensive

CS3204 - Arthur

Least Recently Used (LRU)

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												

Page Faults =

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												
Block Number 3												

Page Faults =

CS3204 - Arthur

Replacement Algorithms: LRU Approximation

- Uses reference bits in the MBT and a static reference pointer (RP)
- The reference pointer is not reinitialized between calls to LRU Approximation
- Set referenced bit to 1 when loading a page
- Set referenced bit to 1 on a R/W
- Set referenced bit to 0 if currently a 1 and scanning for a replacement page
- Replace page with reference bit = 0

CS3204 - Arthur

LRU Approximation Algorithm...

Initially: RP ← -1

```
begin
  RP := (RP + 1) mod MBTSize;
  While (MBT[RP].Referenced = 1) Do
    Begin
      MBT[RP].Referenced := 0
      RP := (RP + 1) mod MBTSize;
    End
  return(RP);
```

Note: referenced bit is set to 1 when a page is
(a) referenced, and
(b) when first loaded into memory
RP always points to last page replaced

CS3204 - Arthur

LRU Approximation

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												

Page Faults =
Page Fault Rate =

CS3204 - Arthur

Replacement Algorithms: Least Frequently Used (LFU)

- Keep a reference count, select page with lowest count
- Reference count is number of times a page has been referenced over its **current** stay in memory, **not** over the lifetime of the program

Page Trace:	0	1	2	3	0	1	4	0	1	2	3	4
Block Number 0												
Block Number 1												
Block Number 2												

Page Faults =

CS3204 - Arthur

Pros/Cons of Demand Paging

⊕ Advantages:

- Can run program larger than physical memory
- Allows higher multiprogramming level than pure paging
- Efficient memory usage
- No compaction is required
- Portions of process that are never called are never loaded
- Simple partition management due to discontinuous loading and fixed partition size
- Easy to share pages

CS3204 - Arthur

Pros/Cons of Demand Paging...

⊖ Disadvantages:

- Internal fragmentation
- Program turnaround time increases each time a page is replaced, then reloaded
- Need special address translation hardware

CS3204 - Arthur