OPERATING SYSTEMS

LECTURE #3: MODERN PARADIGMS OF MEMORY MANAGEMENT

Written by David Goodwin
based on the lecture series of Dr. Dayou Li
and the book *Understanding Operating Systems* 4^{th} *ed.*
by *I.M.Flynn and A.McIver McHoes* (2006)

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY,
UNIVERSITY OF BEDFORDSHIRE.

OPERATING SYSTEMS, 2013

11th FEBRUARY 2013

Outline

1. Introduction
2. Paged Memory
    - Displacement
3. Demand Paging
    - PMT
    - Hardware instruction processing
    - Page fault handler
    - Replacement Policies
        - FIFO
        - LRU
        - MRU
        - LFU
4. Segmented Memory
5. Virtual Memory
6. Cache Memory
7. summary

Introduction
Paged Memory
Displacement
Demand Paging
PMT
Hardware instruction processing
Page fault handler
Replacement Policies
FIFO
LRU
MRU
LFU
Segmented Memory
Virtual Memory
Cache Memory
summary

INTRODUCTION

Review

- Last lecture, we looked at some simple memory allocation schemes
    - Fixed partitions
    - First-fit
    - Best-fit
    - Deallocation
- Each of these required the memory manager to store the entire program in main memory in contigious locations
- They created problems such as fragmentation, or the overheads of relocation/compaction
- This lecture we will follow memory allocation schemes that remove the restriction of storing programs contigiously, and remove the requirement that the entire program reside in memory during execution

Introduction
Paged Memory
  Displacement
Demand Paging
  PMT
  Hardware instruction processing
  Page fault handler
  Replacement Policies
    FIFO
    LRU
    MRU
    LFU
Segmented Memory
Virtual Memory
Cache Memory
summary

PAGED MEMORY

PAGED MEMORY ALLOCATION

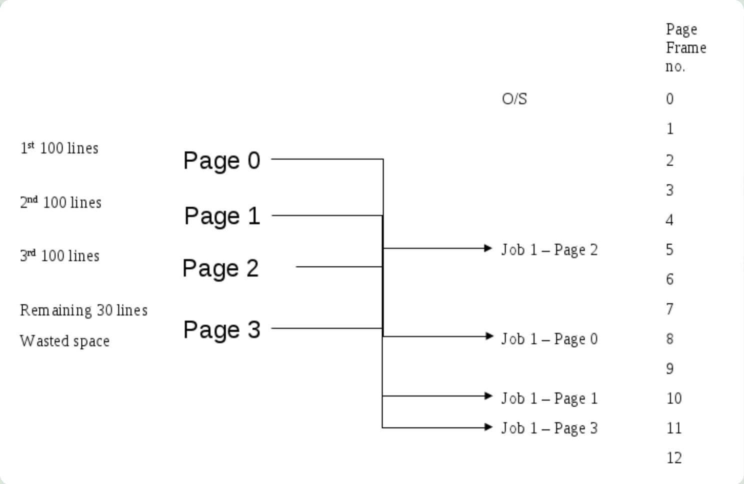
- Concept
  - Each incoming job is divided into pages of equal size
  - Main memory is also divided into a number of page frames
  - In some O/Ss, sizePage = sizeDiskSector = sizeMemoryPageFrame to ensure efficiency
    - Sections of disk where jobs are stored are called **sectors** (or blocks)
    - Sections of main memory are called **page frames**
  - One sector will hold one page of job instruction and fit into one page frame of memory
  - When loading a job into main memory, the Memory manager needs to
    1. Calculate the number of pages in the job
    2. Locate enough free page frames in main memory
    3. Load all pages into the page frames

PAGED MEMORY ALLOCATION

- Analysis
  - Advantages:
    - Pages do not need to store in the main memory contiguously (the free page frame can spread all places in main memory)
    - More efficient use of main memory (comparing to the approaches of early memory management) - no external/internal fragmentation
  - New problem brought:
    - The memory manager has to *keep track of all* the pages
    - Enlarging the size and complexity of OS (overhead)
    - When a job is to be executed, the entire job must be loaded into memory

PAGED MEMORY ALLOCATION

- Example

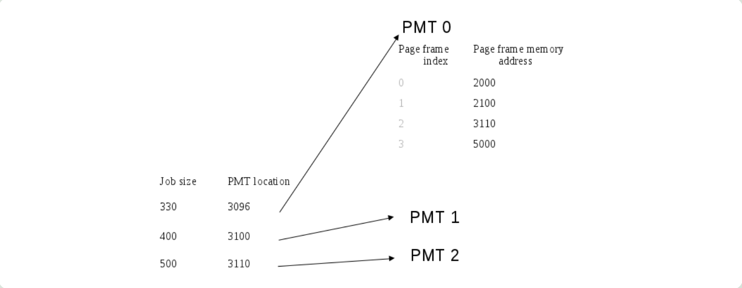


1st 100 lines — Page 0
2nd 100 lines — Page 1
3rd 100 lines — Page 2
Remaining 30 lines
Wasted space — Page 3

Page Frame no.	
0	O/S
1	
2	
3	
4	
5	Job 1 – Page 2
6	
7	
8	Job 1 – Page 0
9	
10	Job 1 – Page 1
11	Job 1 – Page 3
12	

PAGED MEMORY ALLOCATION

- Mapping jobs with page frames
  - Job table (JT) contains two entities:
    - Job size
    - PMT location
  - Page map table (PMT) contains
    - Page index (hidden)
    - Page fame memory address (the first address of a page frame)
  - Memory map table (MMT)
    - Status of each page frame



PMT 0

Page frame index	Page frame memory address
0	2000
1	2100
2	3110
3	5000

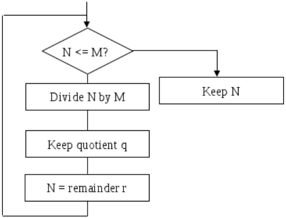
Job size	PMT location
330	3096
400	3100
500	3110

PMT 1

PMT 2

PAGED MEMORY ALLOCATION

- Displacement (or offset) of a line:
  - A factor used to locate each lines of a page within its page frame
  - It is relative to the location of line 0 of a page
  - Algorithm
    - N – page size
    - M – line no to be located
    - Quotient – page number (M/N)
    - Remainder – displacement
  - Example:



PAGED MEMORY ALLOCATION

- The division algorithm is carried out by the hardware, the OS is responsible for maintaining the tables
- This gives the location of the line with respect to the job's pages
- The algorithm needs to be expanded to find the exact location of the line in main memory

PAGED MEMORY ALLOCATION

. . . we need to correlate each job's pages with its page frame number via the Page Map Table

1. determine the page number and displacement of line
    1. Page number = integer quotient from division algorithm (job space address / page size)
    2. Displacement = remainder from the page number division
2. Refer to this job's PMT and find which page frame contains the job's page we are interested
3. Get the address of the beginning of the page frame by multiplying the page frame number by the page frame size
4. Now, add the displacement (from step 1) to the starting address of the page frame to compute the precise location in memory of the line of interest

PAGED MEMORY ALLOCATION

- It can be seen that this is a lengthy process
    - Every time an instruction is executed, or a data value is used; the OS must translate the job space address, which is **relative**, into its physical memory, which is **absolute**
    - This process of resolving the address is called **address resolution** (or address translation)
- The key to success of this scheme is the size of the page
    - too small - very long PMT
    - too large - excessive internal fragmentation

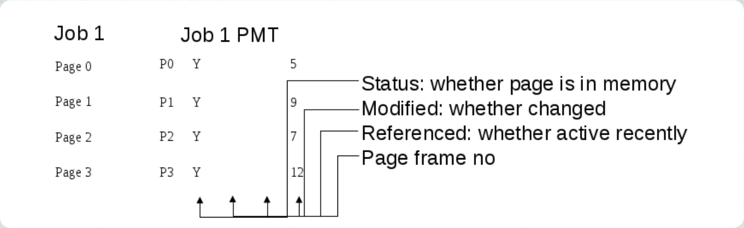
DEMAND PAGING

DEMAND PAGING

- Fist scheme that removed the restriction of having the entire job in memory from beginning to end of processing
- Takes advantage of the fact that programs are written sequentially i.e. not all pages are necessary at once
- Concept
    - Allowing to load only a part of the program into memory for processing
    - Some pages in memory need to be removed to give rooms for other pages that are requested
    - PMT contains more entries to support different predefined policies for replacing pages
        1. to determine if the page begin requested is already in memory
        2. to determine if the page contents have been modified
        3. to determine if the page has been referenced recently
- The key to successful implementation of this scheme is to use high-speed direct access storage device (hard drive) that can work directly with the CPU

DEMAND PAGING

- PMT Example:



Job 1	Job 1 PMT		
Page 0	P0	Y	5
Page 1	P1	Y	9
Page 2	P2	Y	7
Page 3	P3	Y	12

Status: whether page is in memory
Modified: whether changed
Referenced: whether active recently
Page frame no

- If a page is already in memory, the system will be spared time of bringing the page from secondary storage
- If the page has not been modified, it doesn't need to be rewritten to the secondary storage
- Page-swapping policy schemes determine which pages should remain in memory (relatively active ones) or which should be swapped out (relatively inactive ones) to make room for other pages being requested

Hardware instruction processing algortihm

Algorithm 1 Hardware instruction processing algortihm

```
Start processing instruction
Generate data addresses
Compute page number
if page is in memory then
    get data and finish instruction
    advance to next instruction
    return to step 1
else
    generate page interrupt
    call page fault handler
end if
```

Page fault handler algortihm

- When the previous test fails, the OS software takes over with a section called **page fault handler**

Algorithm 2 Page fault handler algortihm

```
if there is no free page frame then
    select page to be swapped out using page removal algorithm
    update job's Page Map Table
    if content of page has been changed then
        write page to disk
    end if
end if
Use page number from step 3 from the Hardware Instruction Processing Algorithm to get disk address where the requested page is stored (the file manager uses the page number to get the disk address)
Read page into memory
Update job's Page Map Table
Update Memeory Map Table
Restart interrupted instruction
```

DEMAND PAGING

- Where there is an excessive amount of **page swapping** between main memory and secondary storage, the operation becomes inefficient. This is called **thrashing**.
- Ideally, a demand paging scheme is most efficient when pragrammers are aware of the page size used by their operating system, and design their programs to keep page faults to a minimum (not really feasible).

Example:

- we could have such a situation where a loop is split over two pages:

```
① for(j=1:j<100;++j)
  {
  k=j*j;
② m=a*j;
  }
```

DEMAND PAGING

- Replacement policies
  - First-in First-out (FIFO): the page which has been in memory the longest will be removed to give room for a page requested
  - Example
    - A program consists of 4 pages A, B, C, D
    - These pages are requested in the sequence of ABACABDBACD
    - There are only two free page frames available
    - A page interrupt is denoted by an asterisk (*)

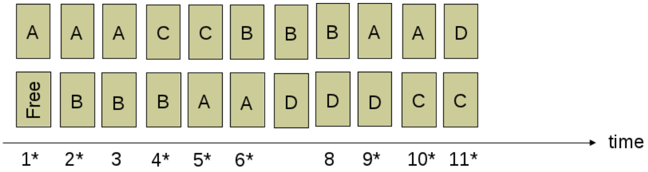
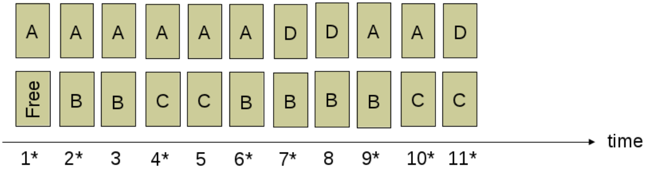


Figure : The success rate of this example is 2/11, or 18%, and the failure rate is 82%.

DEMAND PAGING

- Least recent used (LRU): the page that shows the least recent used will be removed to give room for a page requested
    - Referenced column of PMT is checked for this purpose
    - Example
        - A program consists of 4 pages A, B, C, D
        - These pages are requested in the sequence of ABACABDBACD
        - There are only two free page frames available



1*	2*	3	4*	5	6*	7*	8	9*	10*	11*
A	A	A	A	A	A	D	D	A	A	D
Free	B	B	C	C	B	B	B	B	C	C

Figure : The success rate of this example is 3/11, or 27%, and the failure rate is 73%.

DEMAND PAGING

- Most recent used (MRU): the page that shows the most recent active will be removed to give room for a page requested
  - Last referenced time is checked for this purpose
  - Example
    - A program consists of 4 pages A, B, C, D
    - These pages are requested in the sequence of ABACABDBACD
    - There are only two free page frames available

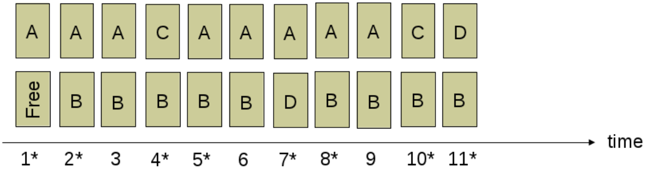


Figure : The success rate of this example is 4/11, or 36%, and the failure rate is 66%.

DEMAND PAGING

- Least frequently used (LFU): the page that shows the least frequently used will be removed to give room for a page requested
  - Time when loaded is checked for this purpose
  - Example
    - A program consists of 4 pages A, B, C, D
    - These pages are requested in the sequence of ABACABDBACD
    - There are only two free page frames available

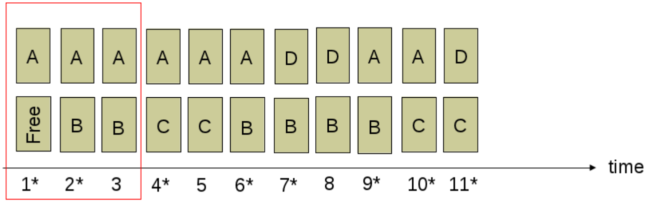


Figure : The success rate of this example is 3/11, or 27%, and the failure rate is 73%.

SEGMENTED MEMORY

SEGMENTED MEMORY ALLOCATION

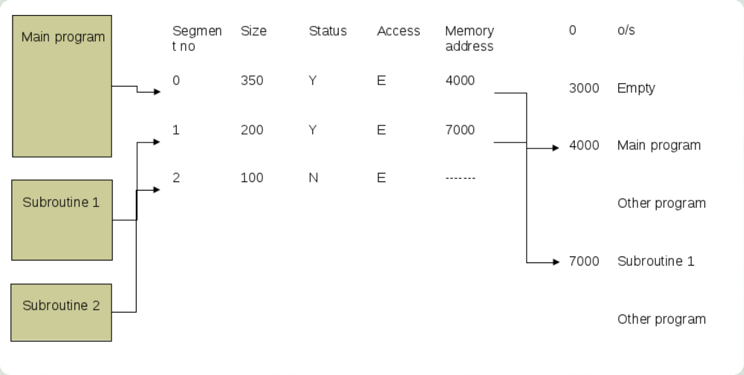
- Problem in paging
    - Pages have the same and fixed size, like fixed partitions, can still cause memory waste via internal fragmentation
- Concept of segmented memory allocation
    - A job is divided into segments according program's structural modules (logical groupings of code)
    - A segment map table (SMT) is generated for each job
    - These segments can have different size
    - Memory is allocated in a dynamic manner (something like dynamic partitions)
- designed to reduce page faults that result from having a segment's loop split over two or more pages
- a **subroutine** is an example of one such logical group

SEGMENTED MEMORY ALLOCATION

- Segment Map Table contains:
    1. segment numbers
    2. segement lengths
    3. access rights
    4. status
    5. location in memory
- The memory manager needs to keep track of of segments in memory.
- This is done with three tables combining the aspects of both dynamic partitions and demand paging:
    1. Job Table lists every job in process
    2. The Segment Map Table lists details about each segement
    3. The Memory Map Table monitors the allocation of main memory

SEGMENTED MEMORY ALLOCATION

- To access a specific location within a segment we can perform an operation similar to the one used for paged memory management
- The only difference is we work with segemnts instead of pages
- Example



Segment no	Size	Status	Access	Memory address
0	350	Y	E	4000
1	200	Y	E	7000
2	100	N	E	-------

0	o/s
3000	Empty
4000	Main program
	Other program
7000	Subroutine 1
	Other program

SEGMENTED MEMORY ALLOCATION

- The disadvantage of any allocation scheme in which memory is partitioned dynamically is the return of external fragmentation.
- Therefore, the relocation/compaction algorithms must be used at predefined times
- The difference between paging and segmentation is conceptual:
  - Pages are physical units that are invisible to the user's program, and are fixed sizes
  - Segements are logical units that are visible to the user's program, and are variable sizes

VIRTUAL MEMORY

VIRTUAL MEMORY

- Concept
    - Virtual memory is a technique which allow pages or segments moving between main memory and secondary storage
    - Virtual memory is based on paging and/or segmentation
    - Information sharing can be implemented without a large I/O overhead
        - Multiple users, each has a copy of shared information
        - With virtual memory, a single copy is loaded on demand
    - Virtual memory works well in a multiprogramming environment

VIRTUAL MEMORY

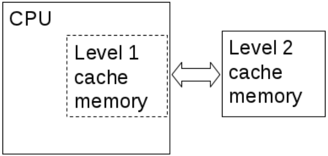
Virtual memory with paging	Virtual memory with segmentation
Allows internal fragmentation	No internal fragmentation
No external fragmentation	Allows external fragmentation
equal sized pages	unequal sized segements
Absolute address needed	Absolute address needed
Requires PMT	Requires SMT

- There are some disadvantages, although outweighed by the advantages. The disadvantages are:
  1. Increased processor hardware costs
  2. Increased overhead for handling page interrupts
  3. Increased software complexity to prevent thrashing

CACHE MEMORY

CACHE MEMORY

- Concept
  - Traditionally, when a program is executed, instructions of the program are loaded to a register in CPU one after another from main memory
  - In most programs, the same instructions are used repeatedly and loading these instructions again and again wastes CPU time
  - Cache memory stores these instructions within CPU
  - Structure



CACHE MEMORY

- Measure of efficiency
    - Hit ratio
      $h = \left(\frac{\text{requests found in cache}}{\text{total requests}}\right)$
    - Hit ratio is usually expressed as a percentage (multiplied by 100 above gives the hit ratio as a percentage)
    - $(1 - h) = m$, miss ratio
- Average memory access time
  $t_a = t_c + (1 - h) * t_m$
  $t_c \equiv$ average cache access time
  $t_m \equiv$ average memory access time

main memory transfer algorithm

Algorithm 3 main memory transfer algorithm

```
CPU puts the address of a memory location in the memory address register and requests
data or an instruction to be retrieved from that address
A test is performed to determine if the block containing this address is already in a cache
slot:
if address is already in cache slot then
    transfer the information to the CPU register - DONE
else
    Access main memory for the block containing the requested address
    Allocate a free cache slot to the block
    perform these in parallel: 1) transfer the information to CPU 2) load the block into
    slot; - DONE
end if
```

Introduction
Paged Memory
Displacement
Demand Paging
PMT
Hardware instruction processing
Page fault handler
Replacement Policies
FIFO
LRU
MRU
LFU
Segmented Memory
Virtual Memory
Cache Memory
summary

SUMMARY

SUMMARY

- Paging
- Segment
- Virtual memory
- Cache

Key terms

address resolution

associative memory

cache memory

demand paging

displacement

FIFO policy

Job Table

LRU policy

Memory Map Table

Key terms

page

page fault handler

page frame

Page Map Table

page replacement policy

page swapping

paged memory allocation

sector

segement

Segment Map Table

segmented memory allocation

subroutine

thrashing

virtual memory