Review

- Demand Page and Page Fault
- Free-Frame List
- Performance of Demand Page with Page Fault
- Swapping with Paging
- Copy-on-Write between Parent and Child
- Page Table Entries
- Page Replacement Algorithms
  - Optimal Algorithm
  - Not Recently Used
  - First In First Out
  - Second Change
  - The Clock Page Replacement
  - Least Recently Used

Preview

- Belady's Anomaly
  - Structure of Stack Algorithm
  - Stack Algorithm
- Design Issues for Paging System
  - Local versus Global allocation
  - Load Control
  - Page Size
- Segmentation
  - Hardware for Segmentation Implementation
  - Advantage and Disadvantage with Segmentation
- Segmentation with Paging (case study MULTICS)

Modeling Page Replacement Algorithm (Belady's Anomaly)

Belady's Anomaly (by **László Bélády)**

- Intuitively, it might seem that the more page frames the process has, the fewer page faults a process will get. This is not always the case – Belady's Anomaly
- **László Bélády** shows the case with FIFO cases more page frames the memory has, the more page faults a program.

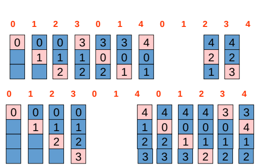
Modeling Page Replacement Algorithm (Belady's Anomaly)

László Bélády

Born April 29, 1928, Budapest, Hungary

Professional Experience:
- IBM Corp.: T.J. Watson Research Center, 1961-1981,
- program manager for software technology, IBM Headquarters, 1981-1983;
- manager of software engineering, Japan Science Institute (now Tokyo Research Laboratories), 1983-1984;
- vice president and director of the software technology and advanced computer technology programs, Microelectronics and Computer Technology Corp. (MCC), 1984-1991;
- chairman, Mitsubishi Electric Research Laboratories, Cambridge, Mass., 1991

Modeling Page Replacement Algorithm (Belady's Anomaly)



9 page faults with 3 page frame

10 page faults with 4 page frame

Modeling Page Replacement Algorithm (Stack Algorithm)

A paging system can be characterized by three items

1. The reference string of the executing process
2. The page replacement algorithm
3. The number of page frames available for a process in memory,

Modeling Page Replacement Algorithm
(Stack Algorithm)

Model for Stack Algorithm

- Maintains an internal array M that keep track of the state of memory.
- M has as many as virtual memory pages *n*.
- Top *m* entries contain all the pages currently in the memory (page frames).
  - n: # of pages, m: # of page frames
- Bottom *n* - *m* entries contains all the pages that have been referenced once but have been page out and are not currently in memory

Modeling Page Replacement Algorithm
(Model for Stack Algorithm)

Properties of the model

1. When a page is referenced, it is always moved to the top entry in M.
2. If the page referenced was already in M, all pages above it move down one position.
3. A transition from within the box to outside of it corresponds to a page being evicted from the memory.
4. The pages that were below the referenced page are not moved.

Modeling Page Replacement Algorithm
(Property for Stack Algorithm)

Property of Stack Algorithm

- If any page replacement algorithm has following property, we call it is stack algorithm

$$M(m,r) \subseteq M(m+1,r)$$

- If we increase memory size by one page frame and re-execute the process, at every point during the execution, all the pages that were present in the first run are also present in the second run, along with one additional page
- If a replacement algorithm is a stack algorithm, it do not suffer from Belady's Anormaly.

Modeling Page Replacement Algorithm
(Stack Algorithm)

LRU algorithm with the model

1. The reference strings: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1
2. The page replacement algorithm: LRU
3. The number of page frames available in memory:
   1. Virtual spaces: 8 pages
   2. Physical spaces: 4 page frames or
   3. Physical spaces: 5 page faranes

Modeling Page Replacement Algorithm
(Stack Algorithm)

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P	P	P	P	P	P	P		P					P			P						P	

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P	P	P	P	P	P	P		P								P							

LRU with a reference string

0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1

Modeling Page Replacement Algorithm
(Stack Algorithm)

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3												
		0	2	1	3	5	4	6	6	7	4	7												
			0	2	1	3	5	4	4	6	6	4												
				0	2	1	3	5	5	5	5	6												
					0	2	1	1	3	3	3	5												
						0	2	2	1	1	1	1												
							0	0	2	2	2	2												
									0	0	0	0												

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	2	3	4	1
	0	2	1	3	5	4	6	3	7															
		0	2	1	3	5	4	6	3															
			0	2	1	3	5	4	6															
				0	2	1	3	5	4															
					0	2	1	1	5															
						0	2	2	1															
							0	0	2															
									0															

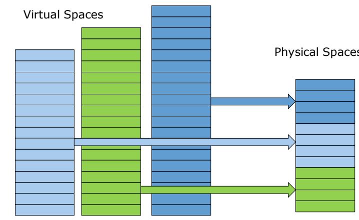
FIFO with same reference string

Design Issues for Paging System
(Local vs. Global Allocation)

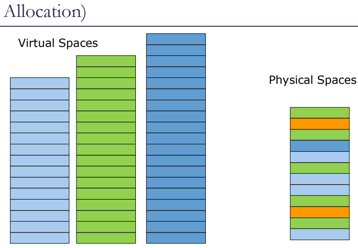
Local versus Global allocation Policies

- With multiple processes competing for frames, we can classify page replacement algorithms into two categories: global replacement and local replacement.
  - Global replacement allows a process to select a replacement frame from the set of all frames.
  - Local replacement requires that each process select from only its own set of allocated frame

Design Issues for Paging System
(Local Allocation)



Design Issues for Paging System
(Global Allocation)



Design Issues for Paging System
(Load Control)

Load Control

- Even though we use best replacement algorithm with global allocation policies, if the combined working sets of all processes exceed the capacity of memory (if system support high degree of multiprogramming with small main memory), thrashing can be expected.
- **Thrashing**: two or more processes accessing a shared resource repeatedly such that serious system performance degradation occurs because the system is spending a disproportionate amount of time just *accessing* the shared resource.
- Possible concern is reducing the degree of multiprogramming – swap out some of processes into the disk.

Design Issues for Paging System
(Page Size)

- Smaller page size smaller internal fragmentation.
- Larger size will cause more unused program to be in memory than a small page size.
- Small pages means that programs will need many pages (hence a large page tables).
- Many small page, wasting time for the seek and rotation.
- Some system need load the page table into the hardware registers to execute. – many small page, need more page table loading time.
- Overhead of paging system (memory overhead)
  - Page table
  - Internal fragmentation loss

Design Issues for Paging System
(Page Size)

Mathematical Analysis

S: average size of process (byte)
P: the size of page (byte)
E: Each page table entry needs (byte).

$\frac{S}{P}$: *Average* number of pages per process

$\frac{S}{P} \times E$: Average page table space

$\frac{P}{2}$: the wasted memory in the last page of the process

Design Issues for Paging System (Page Size)

Total overhead by page table and internal fragmentation loss is

$\frac{S}{P}$: *Average* number of pages per process

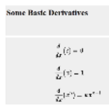
$\frac{S}{P} \times E$: Average page table space

$\frac{P}{2}$: the wasted memory in the last page of the process

$$Overhead(P) = \frac{SE}{P} + \frac{P}{2}$$

$$Overhead'(P) = -\frac{SE}{P^2} + \frac{1}{2} = 0$$

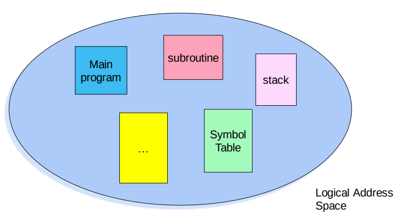
$P = \sqrt{2SE}$: *optimal* page size



Segmentation

- Segmentation is a logical entity, which the programmer knows and uses as a logical entity.
- Segmentation is a memory management scheme that supports the user's view of memory!!

Segmentation (User's View of Memory)



Segmentation

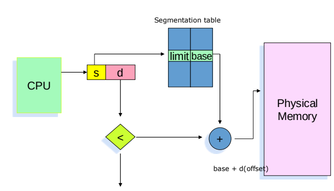
- A logical address space is a collection of segments. Each segment has name and a length.
- Each segment might have different size.
- A segment might contain a procedure or an array or a stack or a collection of scalar variables, but usually it does <u>not contain a mixture of different</u> types
- Normally, when the user program is compiled, <u>the compiler automatically constructs segments reflecting the input program.</u>
- Segmentation table save <u>size of segment</u> and <u>base address</u>(beginning of physical address) for each segment

Segmentation

Ex) any compiler might create separate segments for following:

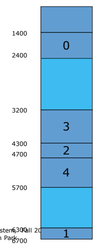
- The global variables;
- The procedure call stack, to store parameters and return address
- The code portion of each procedure or function
- The local variables of each procedure and function

Segmentation Implementation Hardware



Segmentation Implementation Hardware

	limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



Ex) A reference to segment 3 to byte 852 is mapped to

3200 + 852 = 4052

Ex) A reference to segment 0 to byte 1222

would result in error

Advantages with Segmentation
(Protection)

Protection

- Because the segments represent a semantically defined portion of the program, it is likely that all entries in the segment will be used for same way.
- It is relatively easy to protect a segmentation

Advantages with Segmentation
(Protection)

Ex1)
- In a modern architecture, instructions are non-self-modifying, so instruction segment can be defined as a read only or execute only.
- The memory mapping hardware check the protection bit associated with each segment table entry to prevent illegal accesses to memory.

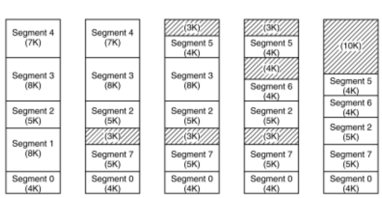
Ex2)
- Big array is placed in its own segment
- The memory-management hardware will automatically check that array indexes are legal and do not stray outside the array boundary's.

Advantages with Segmentation
(Sharing)

A Segment can be shared by several process without allocating segment space for each process!!

Ex) Shared Library
- Modern workstations that run advanced window systems have extremely large graphical libraries compiled into nearly every program.
- Graphical libraries can put into a segment and can be shared by several processes.

Disadvantages with Segmentation
(External Fragmentation)



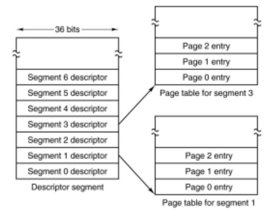
Segmentation with Paging

Reason for the segmentation with paging
- If the segments are large, to keep them in the physical memory might be wasting memory space.
- If a segment's virtual space is larger than physical space, it is not even possible to keep them in the physical memory.
- A solution is the segmentation with paging – each segment is divided into pages.

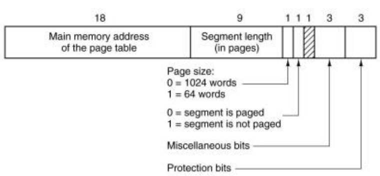
Segmentation with Page (MULTICS)

- For each program a virtual memory of up to 2^{18} segments (more than quarter million) are allowed.
- Each program has segmentation table
- Since there are potentially 2^{18} entries in the segmentation table, a segment table itself is a segment with page table.

Segmentation with Page (MULTICS)

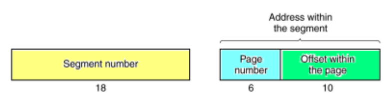


Segmentation with Page (MULTICS)



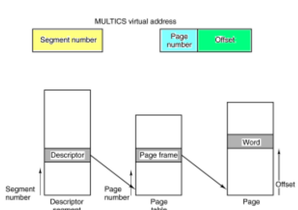
Segmentation table entry

Segmentation with Page (MULTICS)



34 bit Virtual Address

Segmentation with Page (MULTICS)



Segmentation with Page (MULTICS)

When a memory reference is occurs followings are processes

1. The segment number is address for the segment descriptor in the segmentation table
2. Check whether segment's page table is in or not. If it is not in, segmentation fault occurs (OS need to handle it). If it is in go to next
3. Check whether the page is in the memory or not. If the page is not in, page fault (OS need to handle it). If the page is in memory, go to next step
4. The offset in the virtual memory address is added with page frame number. It is sent through address bus to main memory (physical space)
5. Read/write