Operating Systems

Lecture 8

Virtual memory

Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

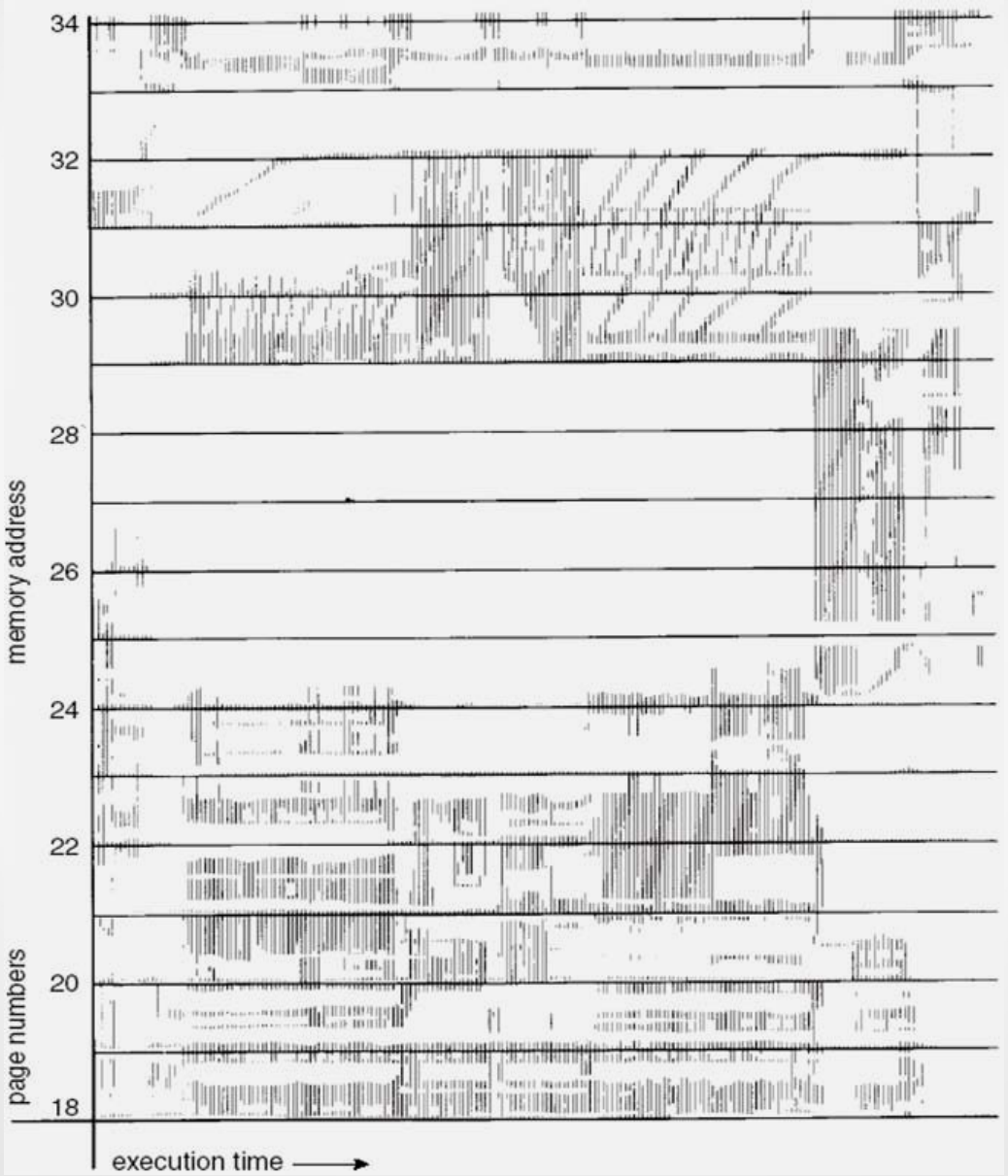
Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

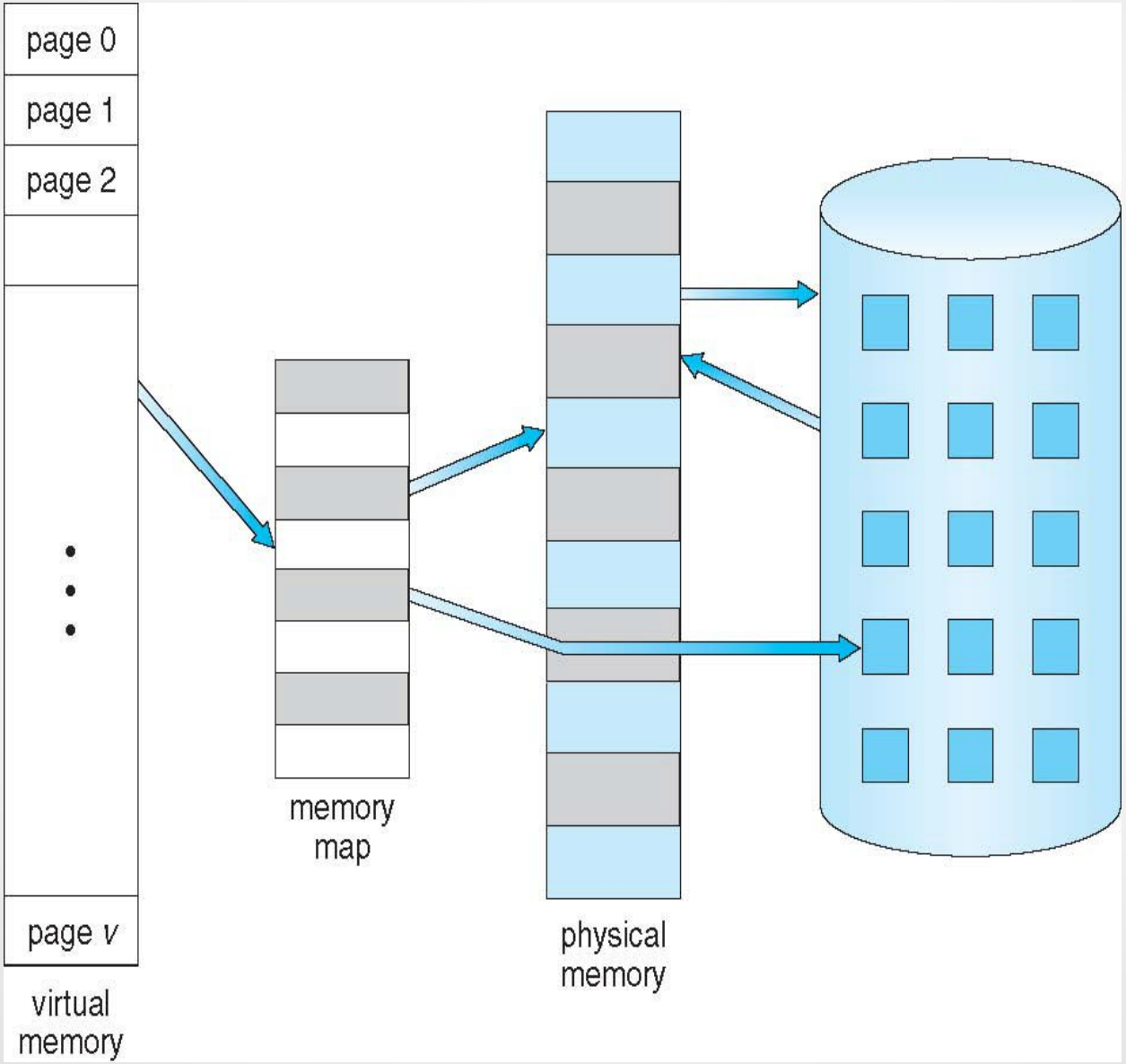
Virtual memory

- Virtual memory separates physical memory from the logical memory.
- The main observation – usually the whole process doesn't have to reside in the main memory to operate – at a given time, processes do not need all their memory but only fragments.
  - All the unused parts of process' memory can be stored somewhere outside the main memory, for instance on the disk.
- Virtual addresses are remapped to physical memory addresses. But the available (potential) address space is usually bigger than the available RAM memory in the system.
  - MMU must be extended so it can tell, if given logical address corresponds to data stored in physical memory, and if not – where the corresponding page is.
  - Virtual memory is an extension of segmentation/paging mechanisms.
  - Virtual memory allows for easy implementation of *shared memory*.

Exemplary process memory reference track



Virtual memory larger than physical one

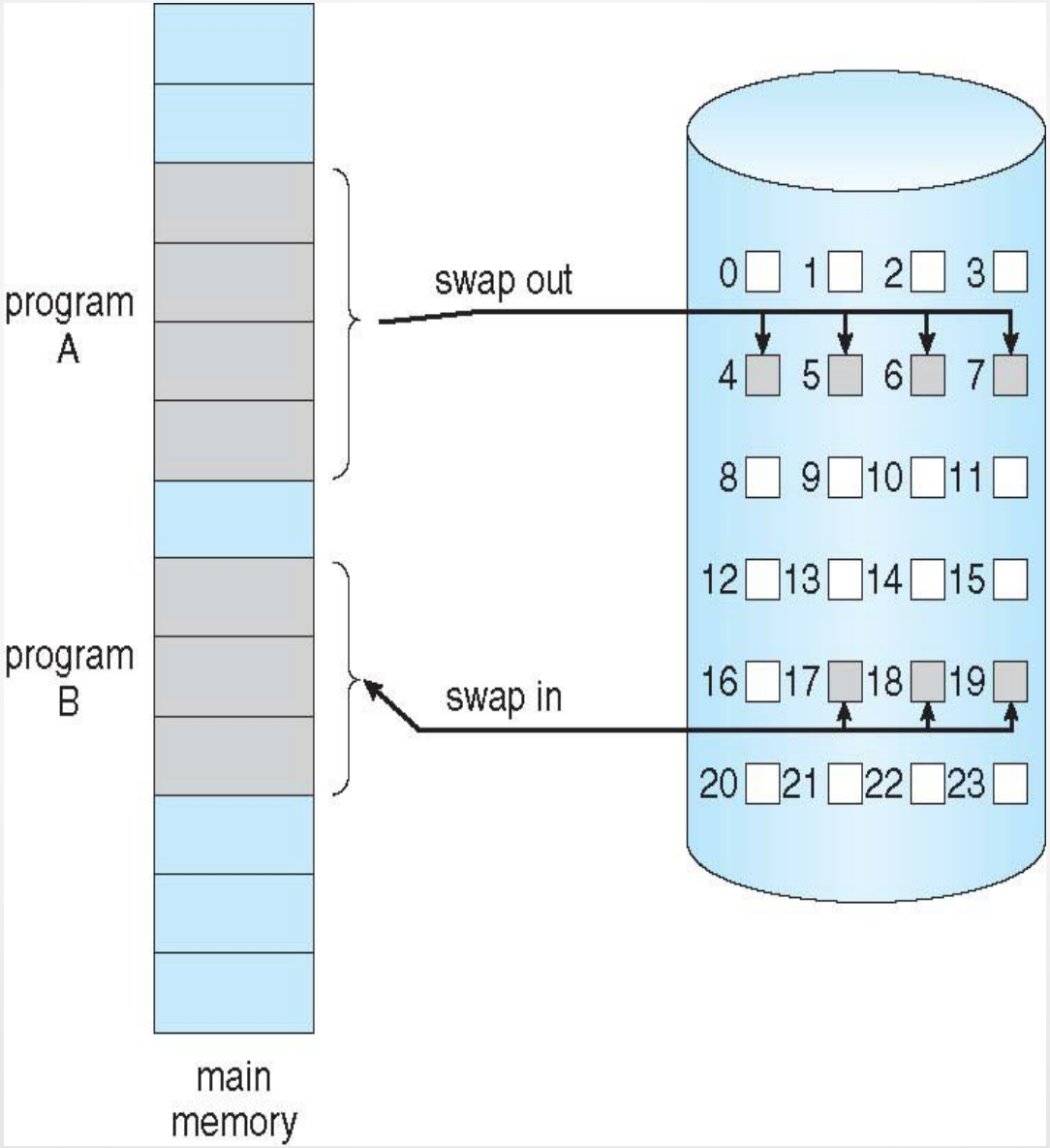


Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

Demand paging

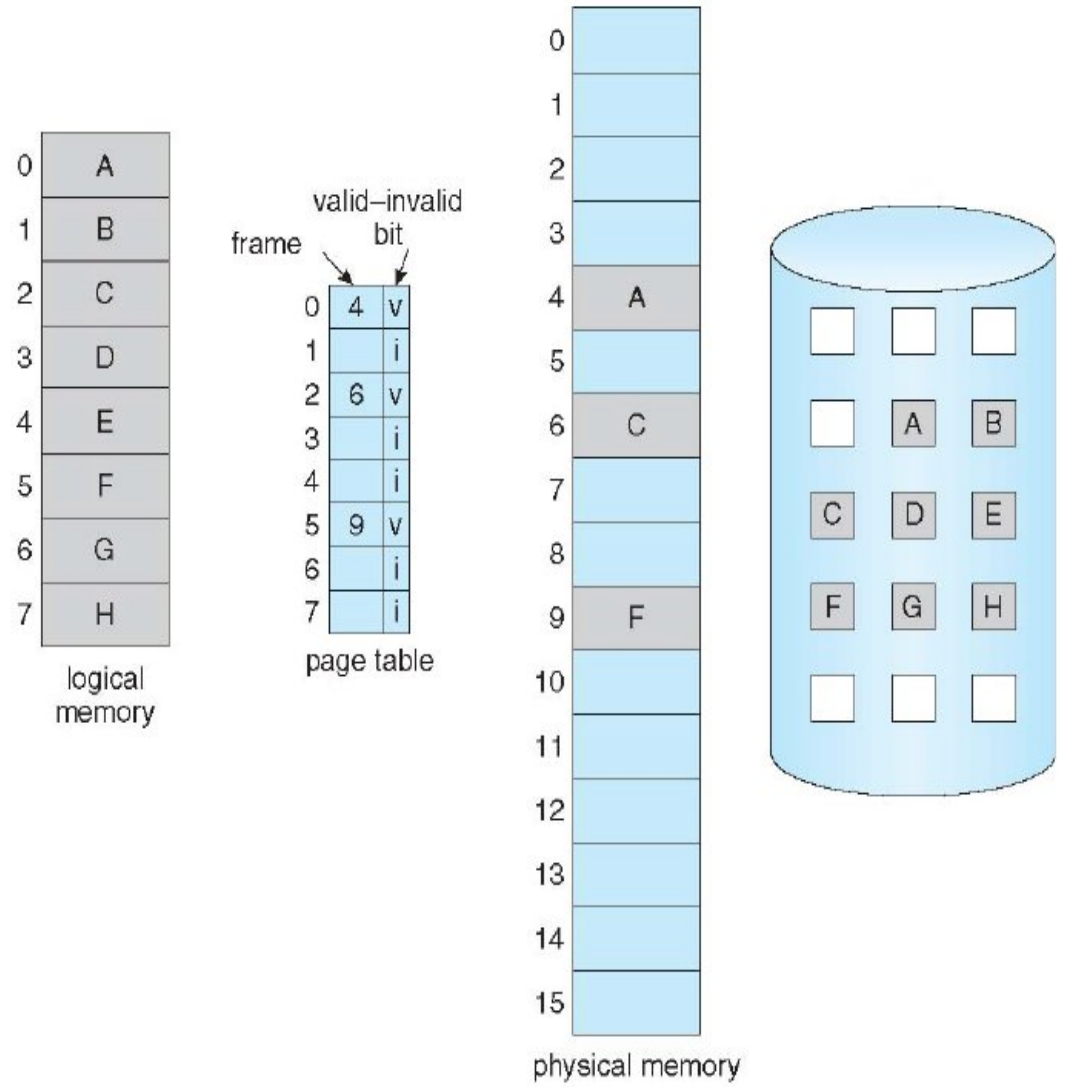
- **Demand paging** consists in fetching a memory page from the disk only when it's needed.
  - Sometimes called a *lazy paging procedure*.
  - It reduces the number of I/O operations – no unneeded pages are transferred.
- When a process issues a memory access request, three thing can happen:
  - Access is invalid
  - Access is valid and the page is in the main memory
  - Access is valid, but the page is not in the main memory.


program
A
program
B
swap out
swap in
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
main
memory

Demand paging

- Determining a page state must be done by hardware – otherwise it's unefficient.
- It can be supported by the **availability bit**, associated with every page entry in the page table.
  - If this bit is set, call is valid and page is in the primary memory.
    - Such request may be fulfilled.
  - If this bit is not set, then the page is not in the memory or reference is invalid.
    - Such request may not be immediately fulfilled – a **page fault** is generated and must be further processed.

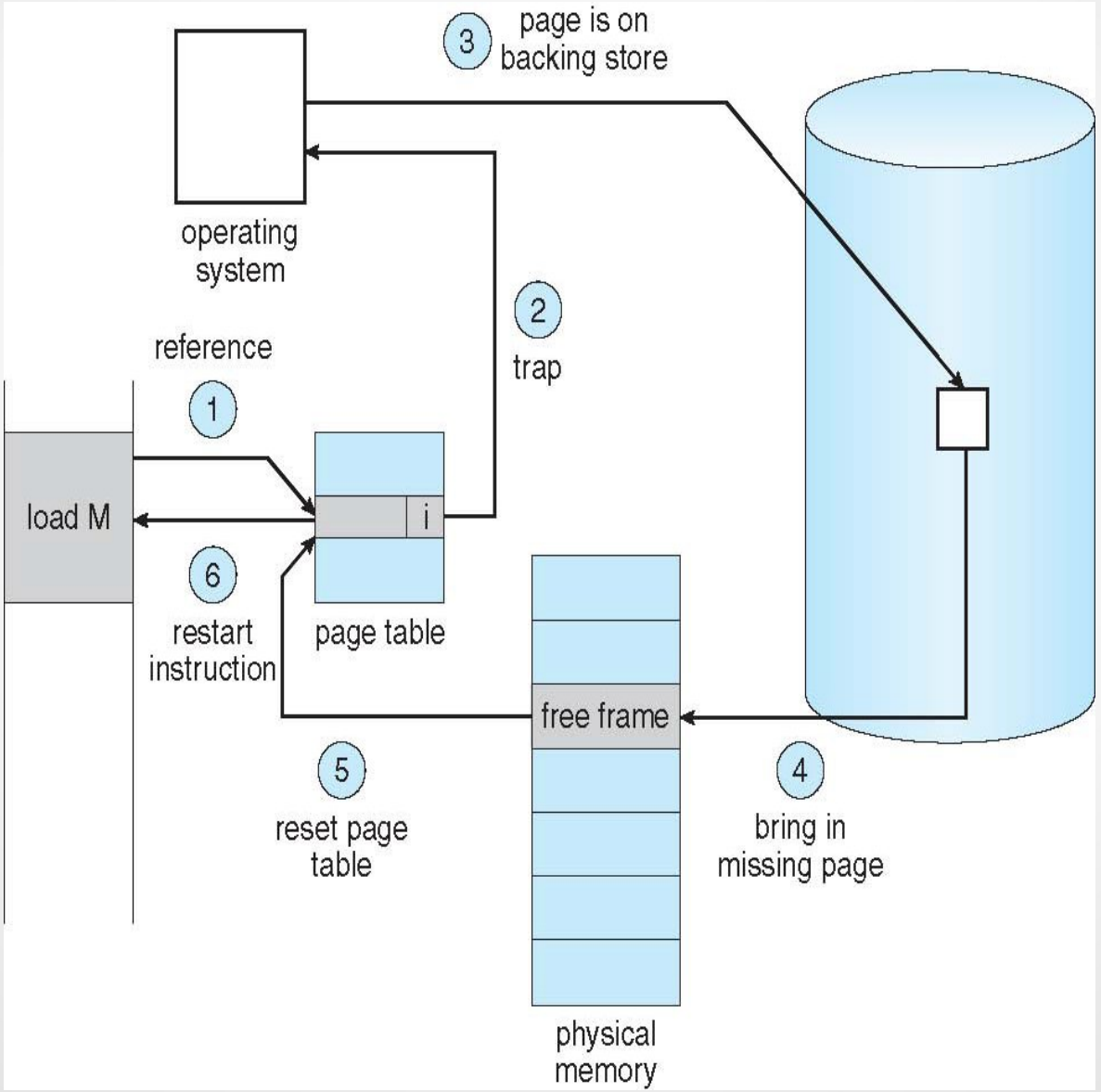
Availability bit



Page fault handling

- When a page fault happens, the following steps are taken:
  - Check in an internal array, if the call was valid. If not – return an access error.
  - Find a free page in the physical memory.
  - Fetch the missing page from the disk.
  - Update the page table.
  - Resume the instruction, which was interrupted by the page fault.

Steps in handling a Page Fault



How much time does a page fault consume?

- **EAP** (*Effective Access Time*) – a measure, which allows to estimate the average access time under the probability of page faults.
  - Assume, that *p* is the probability of page fault, when a program is running.
  - EAT is defined as follows:

    EAT = (1-*p*) * (standard primary memory access time) +
    p * (page fault overhead + [page write time] +
    + page read time + process resume overhead +
    + standard primary memory access time)
  - If the page is in the memory (probability 1-*p*), only standard memory access time is considered. If the page is not in the memory, we must also include all other activities.

How much time does a page fault consume?

- Example:
  - Standard memory access time: 2ns
  - Half of replaced pages need to be written to the disk
  - Disk access time: 5ms (5000000 ns)
  - We neglect overheads of interrupt handling and process resuming for they are very small, comparing to disk access).
  - EAT = (1-*p*) * 2 + *p* * (50% * 5000000 + 5000000) ns =

    2 + *p* * 7499998 ns
- EAT is directly proportional to probability of page fault. For EAT to be acceptable, *p* must be very small.

Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

Page substitution

- What to do, when there are no available free pages in the memory and we have to fetch one from the disk?
  - A system must select one page from the memory and send it to the disk. So it is substituted by the page from the waiting request.
  - Page substitution includes:
    - Find the requested page on the disk
    - Find a *victim* page in the main memory, which will be sent to the disk.
    - Store the victim page to the disk, if it is not there (if it is new or was modified).
    - Unset availability bit for this page
    - Read the requested page to the available memory area.
    - Set its availability bit.

Page substitution

- What to do, when there are no available free pages in the memory and we have to fetch one from the disk?
  - A system must select one page from the memory and send it to the disk. So it is substituted by the page from the waiting request.
  - Page substitution includes:
    - Find the requested page on the disk
    - Find a *victim* page in the main memory, which will be sent to the disk.
    - Store the victim page to the disk, if it is not there (if it is new or was modified).
    - Unset availability bit for this page
    - Read the requested page to the available memory area.
    - Set its availability bit.

How to find a victim page

- A selection algorithm should minimise the probability of page faults.
- At run time, we do not know the future reference requests. It means, that a system can only use a heuristics.
  - FIFO,
  - Optimal algorithm,
  - LRU,
  - Second chance algorithm,
  - Counting algorithms.

Test case

- Let's assume, that we have 4 physical pages.
- We consider the following sequence of page access requests

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

FIFO

- In the **FIFO** heuristics, the page which stays in the memory for the longest time is selected as a victim. So the pages are brought and removed from the memory in the FIFO order.
- This algorithm has a very small overhead and is easy to implement.

Time	**1**	**2**	**3**	**4**	5	6	**7**	8	**9**	10	**11**	**12**
Ref.	1	2	3	4	1	2	5	3	1	5	2	4
Page 1	**1**	1	1	1	1	1	**5**	5	5	5	5	5
Page 2		**2**	2	2	2	2	2	2	**1**	1	1	1
Page 3			**3**	3	3	3	3	3	3	3	**2**	2
Page 4				**4**	4	4	4	4	4	4	3	**4**

Optimal algorithm

- In the **optimal algorithm**, a system selects this page, which will be referenced as the last in the future.
- This theoretical – a system cannot know the future.
- This algorithm gives optimal order of page selections.

Time	**1**	**2**	**3**	**4**	5	6	**7**	8	9	10	11	**12**
Ref.	1	2	3	4	1	2	5	3	1	5	2	4
Page 1	**1**	1	1	1	1	1	1	1	1	1	1	1
Page 2		**2**	2	2	2	2	2	2	2	2	2	2
Page 3			**3**	3	3	3	3	3	3	3	3	3
Page 4				**4**	4	4	**5**	5	5	5	5	**4**

Last Recently Used (LRU)

- The LRU heuristics selects this page, which was recently referenced as the last one in the past (looking back).
- The main idea – if the page was recently used, then it will probably be needed soon, so let's replace the page to which the reference was done earlier.
- Implementation needs changing of reference timestamp with every reference, so hardware support is needed for fast working.

Time	**1**	**2**	**3**	**4**	5	6	**7**	**8**	9	10	11	**12**
Ref.	1	2	3	4	1	2	5	3	1	5	2	4
Page 1	**1**	1	1	1	1	1	1	1	1	1	1	1
Page 2		**2**	2	2	2	2	2	2	2	2	2	2
Page 3			**3**	3	3	3	**5**	5	5	5	5	5
Page 4				**4**	4	4	4	**3**	3	3	3	**4**

Second chance algorithm

- The second chance algorithm is a modification of FIFO algorithm, extended towards some characteristics of LRU heuristics (if the page was recently used, let's give it a second chance).
- Each page has additional bit assigned, which is set to 1 with every reference to this page.
- Whenever a page must be replaced, this algorithm traverses the queue, until the victim page is found:
  - If the first page in the queue has its bit set to 1, it is put at the end of the queue with this bit reset.
  - If the first page in the queue has its bit set to 0, it is selected as the victim page.

Time	**1**	**2**	**3**	**4**	5	6	**7**	8	**9**	10	**11**	**12**
Ref.	1	2	3	4	1	2	5	3	1	5	2	4
Page 1	**1** 1	1 1	1 1	1 1	1 1	1 1	5 1	5 1	5 1	5 1	5 1	5 0
Page 2		**2** 1	2 1	2 1	2 1	2 1	2 0	2 0	**1** 1	1 1	1 1	1 0
Page 3			**3** 1	3 1	3 1	3 1	3 0	**3** 1	3 1	3 1	3 0	**4** 1
Page 4				**4** 1	4 1	4 1	4 0	4 0	4 0	4 0	**2** 1	2 1

Counting algorithms

- Counting algorithms use additional counters assiciated with each page. They must have a hardware support – each page access equals counter incrementation.
  - Least Frequently Used (LFU) – replace the page with the smallest number of references (such page is not important for others we used more times).
  - Most Frequently Used (MFU) – replace the page with the biggest number of references (page with small reference counter was probably recently loaded and will be needed soon).
- These heuristics are not popular for they need additional hardware support and their performance far from the one of an optimal algorithm.

Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

Page assignment

- In a system with many processes running simultaneously, it is important, how many pages are assigned to a given process. This number may be fixed or dynamic.
  - **Local assignment** – each process is assigned a fixed number of pages and must work using only this set.
  - **Global assignment** – a number of pages assigned to a process may be changed (both increased and decreased).

Page allocation strategies

- **Fixed assignment** – a process has a fixed number of pages assigned. If there are pages left in a system, processes may apply for them. Initial assignment may be equal or proportional.
- **Priority assignment** – each process has its priority. If a page fault happens in a process, a system selects as a victim page one of the pages belonging to processes with lower priority.
- **Page fault counting** – if a page fault frequency in a given process falls below a certain threshold, it looses one of the pages assigned. On the other hand, if this frequency exceeds another threshold, such process receives one more page.

Summary

- This lecture shows the problem of virtual memory:
  - Introduction.
  - Demand paging
  - Page substitution strategies
    - FIFO
    - The optimal algorithm
    - Last Recently Used (LRU)
    - Second chance
    - Counting algorithms
  - Page assignment
  - Working set

Working set

- If a program is well written, a process running it will use only a small number of pages at a time and will reference them frequently.
- A working set is a set of pages, which were referenced during the last *I* instructions (for a fixed *I*, for instance *I*=100000).
- The knowledge of working sets for processes in a system allows to „fairly” distribute the whole physical memory between all the processes, proportionally to their working set size.

Optimization example

- Program structure
  - int data[1024,1024];
  - Each row is stored in one page (4kB pages)

- Program 1

```
for (j = 0; j <1024; j++)
    for (i = 0; i < 1024; i++)
        data[i,j] = 0;
```

- 1024 x 1024 = 1048576 page faults

- Program 2

```
for (i = 0; i < 1024; i++)
    for (j = 0; j < 1024; j++)
        data[i,j] = 0;
```

- 1024 page faults

Implementation examples

- Windows XP
- Solaris

Windows XP

- Uses demand paging with clustering. Clustering brings pages surrounding the faulting page
- Processes are assigned working set minimum and working set maximum limits
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may have assigned pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum

Solaris

- Maintains a list of free pages to assign faulting processes
- Lotsfree – threshold parameter (amount of free memory) to begin paging
- Desfree – threshold parameter to increasing paging
- Minfree – threshold parameter to being swapping
- Paging is performed by a pageout process
- Pageout scans pages using modified clock algorithm
- Scanrate is the rate at which pages are scanned. It ranges from slowscan to fastscan
- Pageout is called more frequently depending upon the amount of free memory available
- Priority paging gives priority to process code pages

Thank You