Virtual memory

Tran, Van Hoai

Faculty of Computer Science & Engineering
HCMC University of Technology

E-mail: hoai@hcmut.edu.vn
(partly based on slides of Le Thanh Van)

Outline

Outline

Role of virtual memory

There needs a mechanism to load partially a program into physical memory for execution

Why do we need to load those into memory ?

- Code to handle unusual error conditions
- Array allocated largely, but seldomly used fully

Partially loaded programs may provide some advantages

- Programmers can write program for extremely large virtual space
- User program takes less physical memory $\Rightarrow$ multiprogramming increases without any sacrifice of response/turnaround time
- Less I/O needed to load/swap programs into memory

Role of virtual memory

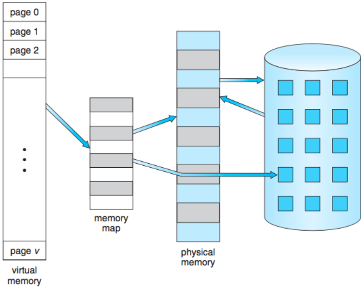
There needs a mechanism to load partially a program into physical memory for execution

Why do we need to load those into memory ?

- Code to handle unusual error conditions
- Array allocated largely, but seldomly used fully

Partially loaded programs may provide some advantages

- Programmers can write program for extremely large virtual space
- User program takes less physical memory $\Rightarrow$ multiprogramming increases without any sacrifice of response/turnaround time
- Less I/O needed to load/swap programs into memory

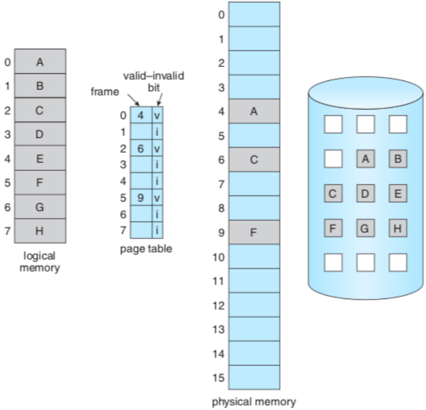


Virtual memory involves the separation of logical memory as perceived by users from physical memory

Demand paging

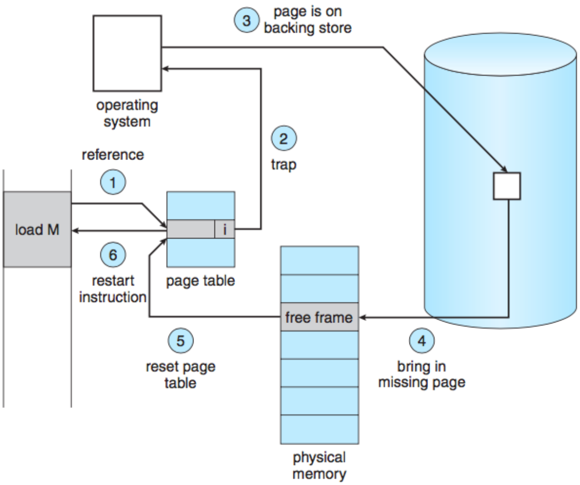
Idea

Demand paging is a strategy to load pages only as they are needed (pager is not swapper)



Valid-invalid bit is used to check memory resident of a page

Steps in demand paging



Pure demand paging = only bringing pages into memory as required

Hardware support

- Page table: marking valid-invalid of a page
- Secondary memory (disk): known as swap device or swap space

An instruction may have page fault anywhere, restarting the whole instruction is needed. Consider three-address instruction ADD A, B, C.

1. Fetch and decode the instruction (ADD)
2. Fetch A
3. Fetch B
4. Add A and B
5. Store sum to C

Hardware support

- Page table: marking valid-invalid of a page
- Secondary memory (disk): known as swap device or swap space

An instruction may have page fault anywhere, restarting the whole instruction is needed. Consider three-address instruction ADD A, B, C.

1. Fetch and decode the instruction (ADD)
2. Fetch A
3. Fetch B
4. Add A and B
5. Store sum to C

What happens if A, B, C cannot be in main memory at the time of instruction ADD executed ?

Performance of demand paging

Denote p be probability of a page fault

$$\begin{aligned} \text{Effective access time} &= (1-p) \times \text{memory access time} \\ &\quad + p \times \text{page fault time} \end{aligned}$$

Page fault time generally consists of

1. Service the page-fault interrupt
2. Read in the page
3. Restart the process

Performance of demand paging

Denote p be probability of a page fault

$$\begin{aligned}\text{Effective access time} &= (1-p) \times \text{memory access time} \\ &\quad + p \times \text{page fault time}\end{aligned}$$

Page fault time generally consists of

1. Service the page-fault interrupt
2. Read in the page
3. Restart the process

If page fault time = 8ms and memory access time = 200ns, then

$$\text{Effective access time (in ns)} = 200 + 7,999,800 \times p$$

If page-fault rate $p = 1/1000$,
then effective access time = 8.2 microseconds ($\approx 40 \times$ 200ns).

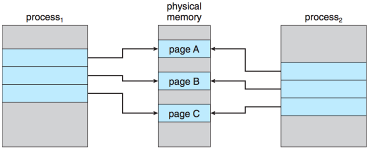
Outline

Copy-on-write

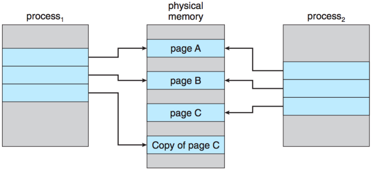
Copy-on-write

- Rapid process creation
- Minimizing the number of new pages allocated to newly created processes

Before



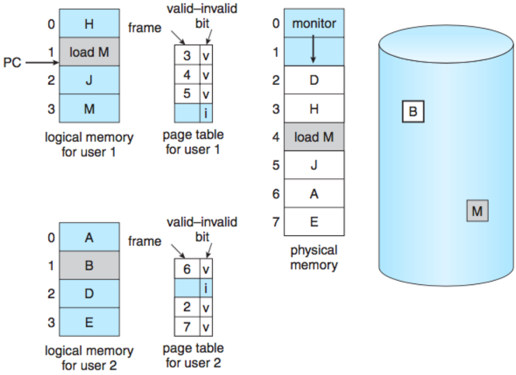
After process 1 modifies page C



- Applied for `fork()`
- `vfork()` has no copy-on-write (more efficient), and parent process is suspended

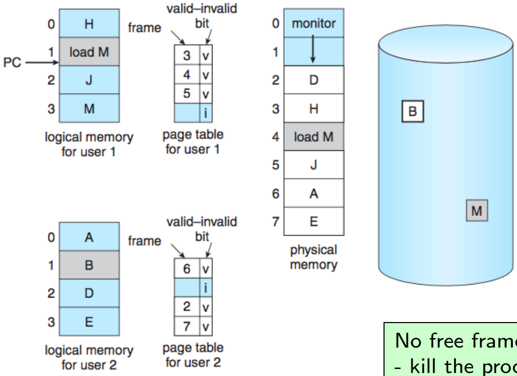
Page replacement

Some processes can utilize less pages than demanding
$\Rightarrow$ In order to increase multiprogramming degree, OS could over-allocating memory (when increasing multiprogramming degree)



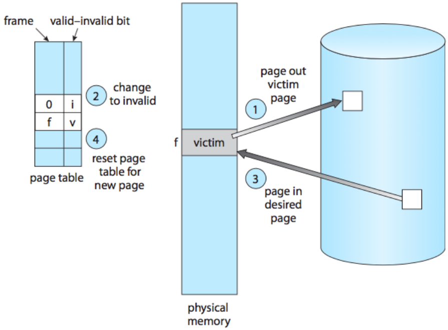
Page replacement

Some processes can utilize less pages than demanding
$\Rightarrow$ In order to increase multiprogramming degree, OS could over-allocating memory (when increasing multiprogramming degree)



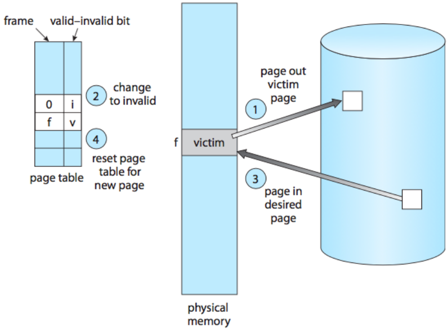
No free frames. Options:
- kill the process
- swap out other process
- **page replacement**

Basic page replacement



1. Find location of desired page on disk
2. Find a free frame
   a. if there is a free frame, use it
   b. if there is no free frame, call page-replacement algorithm to select a victim frame
   c. Write victim to disk and change status of related data structures
3. Read desired page into frame, change data structures
4. Continue the process where page fault occurred

Basic page replacement



1. Find location of desired page on disk
2. Find a free frame
   a. if there is a free frame, use it
   b. if there is no free frame, call page-replacement algorithm to select a victim frame
   c. Write victim to disk and change status of related data structures
3. Read desired page into frame, change data structures
4. Continue the process where page fault occurred

Minor tunnings

Many stuffs in above scheme can be modified to get better performance, such as

- modify bit *(no need to write pages without any modification).*
- Read-only pages can be discarded when desired.

Outline

Page replacement algorithm

Questions

- How many frames allocated for a process ?
- Which frame to be replaced ?
- Different programs have different memory references (namely reference string)

- Recording an address sequence as follows

  0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0609, 0102, 0105

- 100 bytes per page, then address sequence converted to reference string

  1, 4, 1, 6, 1 , 6, 1, 6, 1, 6, 1

FIFO page replacement

Let's try the following reference string for three-frame memory

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1$$

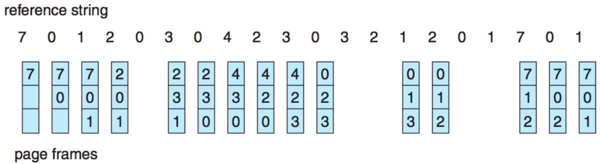
FIFO page replacement

Let's try the following reference string for three-frame memory

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1$$

FIFO page replacement

Choose next page to be replaced in FIFO scheme

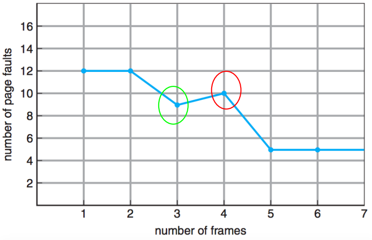


15 page faults

Optimal replacement algorithm

Bélády's anomaly for FIFO

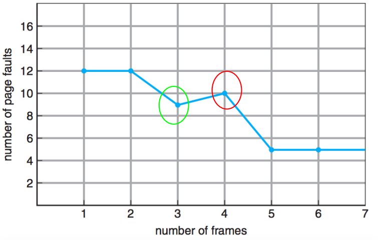
Number of available frames increases, but page fault does increase *(proved to be unbounded (2010))*



Optimal replacement algorithm

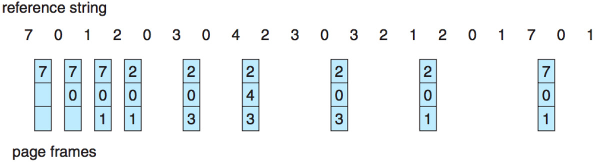
Bélády's anomaly for FIFO

Number of available frames increases, but page fault does increase *(proved to be unbounded (2010))*



Optimal (MIN/OPT) page replacement

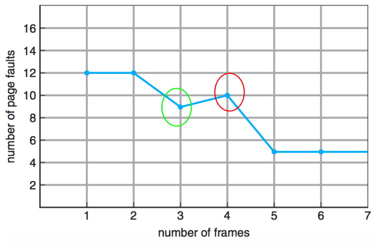
Replace the page that will not be used for the longest period of time



Optimal replacement algorithm

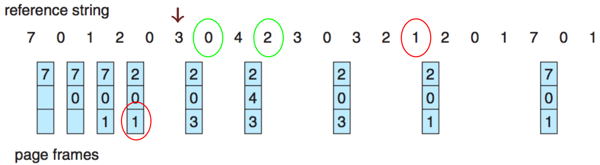
Bélády's anomaly for FIFO

Number of available frames increases, but page fault does increase *(proved to be unbounded (2010))*



Optimal (MIN/OPT) page replacement

Replace the page that will not be used for the longest period of time

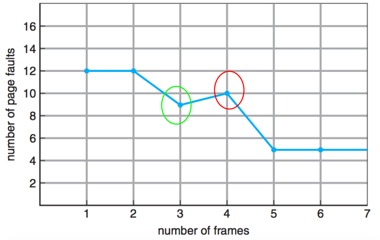


9 page faults

Optimal replacement algorithm

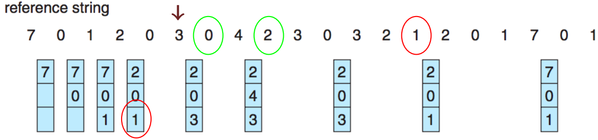
Bélády's anomaly for FIFO

Number of available frames increases, but page fault does increase *(proved to be unbounded (2010))*



Optimal (MIN/OPT) page replacement

Replace the page that will not be used for the longest period of time



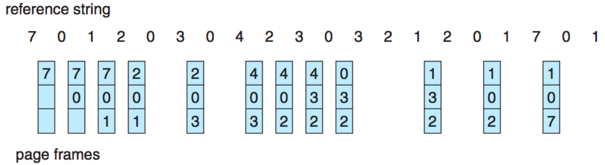
9 page faults

There is no information of the future

LRU replacement algorithm

LRU page replacement

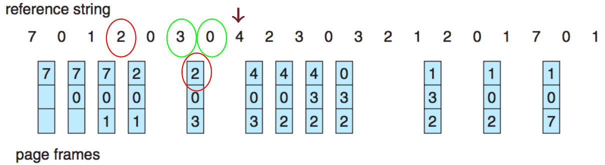
LRU (Least recently used) chooses the page that has not been used for the longest period of time



LRU replacement algorithm

LRU page replacement

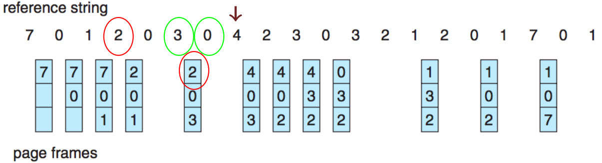
LRU (Least recently used) chooses the page that has not been used for the longest period of time



LRU replacement algorithm

LRU page replacement

LRU (Least recently used) chooses the page that has not been used for the longest period of time



We need hardware support to implement LRU algorithm.

- Counters: an additional field in each page-table entry to store time-of-use (clock or counter). Referencing a page → copying storing clock register to time-of-use field
- Stack: using the idea of a stack, "Last In First Out"

Other algorithms

- LRU-approximation page replacement
  - Additional-reference-bits algorithm
  - Second-chance algorithm
  - Enhanced second-chance algorithm
- Counting-based page replacement
  - Least frequently used (LFU)
  - Most frequently used (MFU)
- Page-buffering algorithm

Outline

Minimum number of frames

Minimum number of frames is defined by computer architecture

IBM MVC instruction needs

a) 6 bytes (can straddle 2 pages)
b) source location (2 pages)
c) destination location (2 pages)

$\rightarrow$ 6 pages totally

Number levels of indirection should be limited

Allocation algorithms

There are m frames

- Equal allocation: n processes $\rightarrow \lfloor \frac{m}{n} \rfloor$ for each process
- Proportional allocation: virtual memory size of p_i is s_i

$$a_i = m \times s_i / \sum s_i$$

a_i possibly depends on a combination of size and priority of the process

Allocation algorithms

There are m frames

- Equal allocation: n processes $\rightarrow \lfloor \frac{m}{n} \rfloor$ for each process
- Proportional allocation: virtual memory size of p_i is s_i

$$a_i = m \times s_i / \sum s_i$$

a_i possibly depends on a combination of size and priority of the process

Performance issues

- Global vs. local allocation
  - Global replacement: replacement frame is any in the set of all frames
  - Local replacement: replacement frame is in its own set of allocated frames
- Non-uniform memory access (NUMA)

Outline

Thrashing

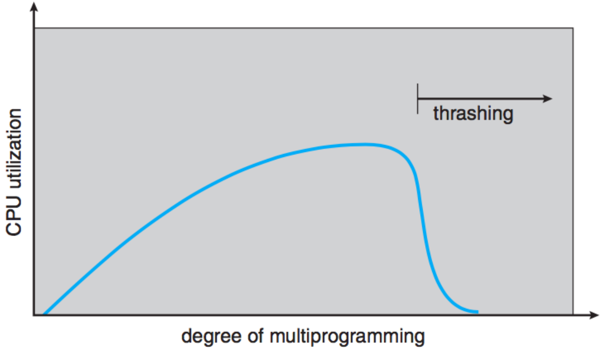
A process has no enough frames. Page faults occur frequently. Time for paging is more than that for execution.

Thrashing

A process has no enough frames. Page faults occur frequently. Time for paging is more than that for execution.

Thrashing

High paging activity is called thrashing



Working with thrashing

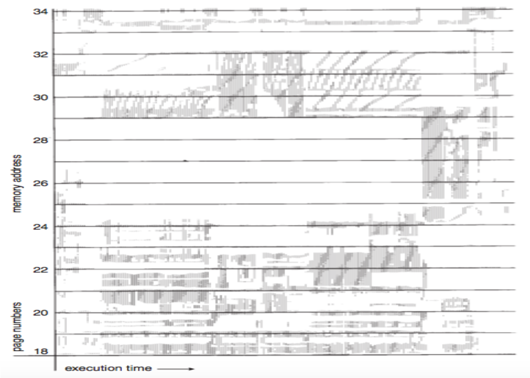
We can limit effects of thrashing by

- Using local replacement algorithm, not steal frames from another process
- Working-set strategy: using a locality model (locality = set of pages that are actively used together)

A function has its own

locality which consists of

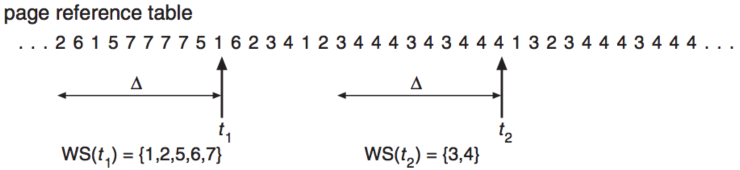
- its local variables
- subset of global variables



Working-set model

- Δ: working-set window
- Working set: set of pages in the most recent Δ pages references

$\Delta = 10$

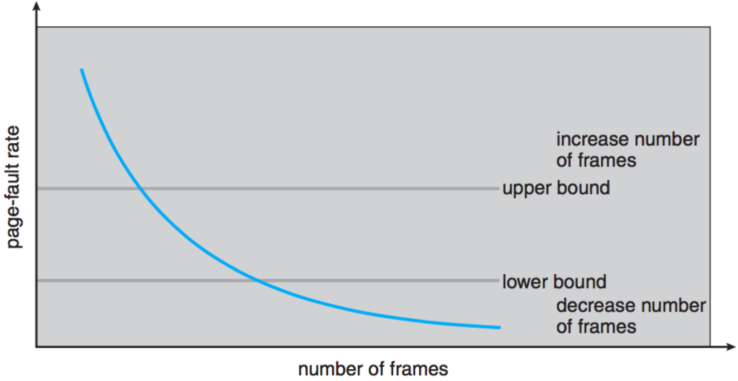


Denote WSS_i be working-set size of process i. Total demand is

$$D = \sum WSS_i$$

We can monitor D and respond accordingly when $D > m$

Page-fault frequency (PFF)



Outline

- Prepaging: prevent large number of page faults that occur when a process is started
- Page size: choice of page sige is often performed in OS design
- TLB reach
- Inverted page tables
- Program structure

```
int i, j;
int data[128][128];

for( j = 0; j < 128; j++ )
  for( i = 0; i < 128; i++ )
    data[i][j] = 0;
```

```
int i, j;
int data[128][128];

for( i = 0; i < 128; i++ )
  for( j = 0; j < 128; j++ )
    data[i][j] = 0;
```

- Prepaging: prevent large number of page faults that occur when a process is started
- Page size: choice of page sige is often performed in OS design
- TLB reach
- Inverted page tables
- Program structure

```
int i, j;
int data[128][128];

for( j = 0; j < 128; j++ )
  for( i = 0; i < 128; i++ )
    data[i][j] = 0;
```

```
int i, j;
int data[128][128];

for( i = 0; i < 128; i++ )
  for( j = 0; j < 128; j++ )
    data[i][j] = 0;
```

If OS allocates fewer than 128 frames of 128 words for entire program, the left code has $128 \times 128 = 16,384$ page faults. The right code possibly has only 128 page faults.