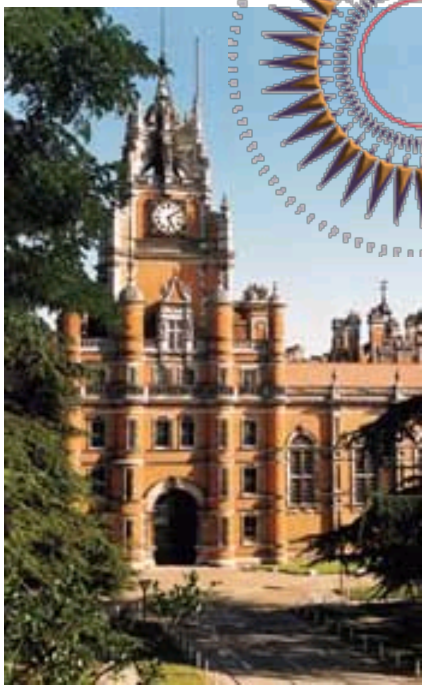


Lecture 6 – Paging Algorithms

Karl R. Wilcox
Karl@cs.rhul.ac.uk

Administration

- **There WILL be a lecture this Friday**
  - **In MFLT, 11:00**

- **Lecture Notes**
  - **Are available at http://www.cs.rhul.ac.uk/~karl**
  - **Formats**
    - **Original Powerpoint 2000 slides**
    - **PDF files, A4 2 slides per page, colour**
    - **Let me know if you would like other formats**

Objectives

- In this class we will discuss:
  - Page replacement algorithms
  - Belady's Anomaly
  - Further memory management
  - Page size optimisation

Page Replacement Algorithms

- When a page fault occurs the operating system has to *choose* a page to remove from memory to make room for the newly brought in page
- If the page has been modified (while in memory) it has to be rewritten on to the disk (update)
- If the page was not modified it can be just overwritten by the new one (**Modified bit**)
- The algorithm which is used by the OS to choose a page which will leave the memory is called the **Page Replacement Algorithm (PRA)**

The Optimal PRA

- What would an optimal PRA be?
- We want the page to be removed from the memory to be that one which will not be used for longest time
  - Each page (from the set in memory) will be referenced after some number of instructions
  - Assign each page a label, the number of instructions that will be executed before that page is nexct referenced
- The optimal PRA will remove the page with the highest label
- Unfortunately, the optimal PRA is not realizable

The Not-Recently-Used PRA

- Every page is given two status bits, *R* and *M*
  - *R* bit is set whenever the page is referenced (r/w)
  - *M* bit is set when the page is modified (w0
- Can be set by hardware or simulated in software
- The NRU algorithm uses these bits to divide the pages into four categories:
  - Class 0: *R* = 0, *M* = 0
  - Class 1: *R* = 0, *M* = 1
  - Class 2: *R* = 1, *M* = 0
  - Class 3: *R* = 1, *M* = 1
- The *R* bit is cleared on every clock interrupt
- Remove a page at random from the lowest numbered non-empty class

The FIFO PRA

- The OS maintains a list of all pages currently in memory
- The page at the head of the list is the oldest one
- The page at the tail is the most recently arrived
- The first-in, first-out algorithm removes the head and adds the newly loaded page at the tail

- This algorithm is rarely used
  - Tends to swap out commonly used pages

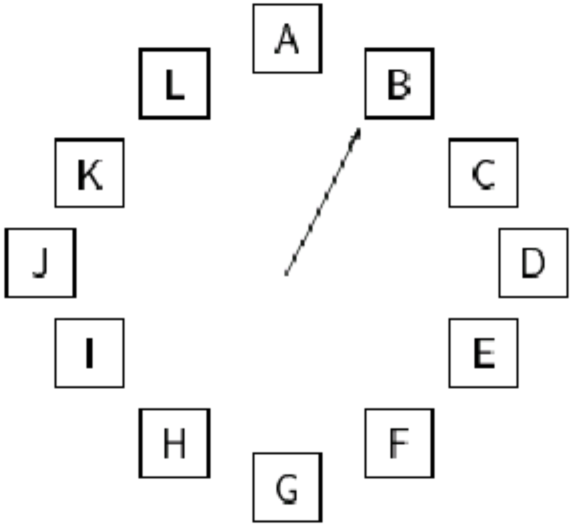
Royal Holloway University of London

The Second Chance PRA

- This is a simple modification to FIFO, it avoids swapping out heavily used pages
- Pages are kept in a list (as FIFO)
- Before removing the oldest page, the algorithm checks the *R* bit:
  - If *R* = 0, the page is removed
  - if *R* = 1, the *R* bit is cleared and the page is moved from the head to the tail of the list and the search continues
- There is a cost in moving pages around

The Clock PRA

- This is further modification of the second chance algorithm
  - No need to move the pages



The Least Recently Used PRA

- **Observation: Pages that have been heavily used in the last few instructions will probably be heavily used again in the next few**
- **Basic idea: When a page fault occurs, throw out the page that has been unused for the longest time**
- **This is the *Least Recently Used* (LRU) paging**
  - **Expensive to implement, need to update timestamp on each access**
  - **May have hardware support**
  - **Can be simulated in software: the *Not Frequently Used* (NFU) algorithm**

Belady's Anomaly

- One would assume that the more page frames the memory has, the fewer page faults a program will get
- This is not always the case!
- Example, using the FIFO PRA – Belady's anomaly
  - Assume 5 virtual pages: 0 1 2 3 4
  - Referenced in order: 0 1 2 3 0 1 4 0 1 2 3 4
  - Use FIFO PRA with 3 page frames and count page faults
  - Use FIFO PRA with 4 page frames and count page faults
  - Which has the most page faults?

The Reference String

- **Assume we have one process running on a machine**
  - **Each process's memory access can be characterised by an ordered list of page numbers**
  - **The list is called the *Reference String***
- **A paging system can be characterised by three items:**
  - **The reference string of the executing process**
  - **The page replacement algorithm**
  - **The number of page frames available in memory, *m***

Abstract Interpreter

- **The abstract interpreter works as follows:**
  - **To keep track of the memory *M*, an internal array is maintained**
  - **It has *n* elements, *n* = number of virtual pages**
  - **The top *m* elements of *M* are the pages in memory**
  - **The bottom part contains pages, that have been referenced once, but which are not in memory now**
  - **Initially *M* is empty**
- **If a page fault occurs (i.e. the page number is not in the top *m* elements) the page is just added (in the beginning) of the PRA is invoked**
- **Top and bottom part can be separately rearranged**

Try An Example

- 6 virtual pages: 0 1 2 3 4 5
- 3 physical page frames: $m = 3$
- Reference string: 0 2 1 5 4 2 3 1 5 3 5 5 1 4 5
- Page replacement algorithm: LRU

Stack Algorithms

- Let *r* be an index into the reference string
- *M(m, r)* is the set of pages in the top part of *M* after *r* memory references
- Algorithms that have the property

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

are called stack algorithms
- They do not suffer from Belady's anomaly

Distance String

- The distance string is the list of distances (1..∞) for each referenced page
- For each page reference the item in the distance string is the distance of the called page from the top of the array M
  - Note: the distance string depends on the reference string AND the PRA
- How many page frames should a process be given?

Design Issues

- Some issues to consider when designing paging systems
  - Demand paging – load pages only when they are referenced
  - Working set model – page references tend to be localised to the working set
    - (Pre)Load all the pages in the working set to avoid future page faults

Design Issue – Page Size

- Goal: determine the optimum page size
- Issues:
  - A randomly chosen text segment (code) will not fill an integral number of pages
    - Implies small pages are best
  - On average, half of the final page will be empty
  - Many pages mean a large page table
    - Implies large pages are best
- We will look at this in more detail in the assessment exercise (on Friday)

Summary

- Most modern operating systems used paged, virtual memory
  - To allow processes to have a virtual address space bigger than physical memory
  - To load only part of the memory required by each process
- Choosing which page to swap out when a page fault occurs can have a significant impact on overall performance
  - We have looked at various algorithms and an abstract model to examine them
- Page size is a trade off between variables

Next Lecture

- On Friday, MFLT
- We will talk about I/O and hardware management
- Lecture Notes: http://www.cs.rhul.ac.uk/~karl