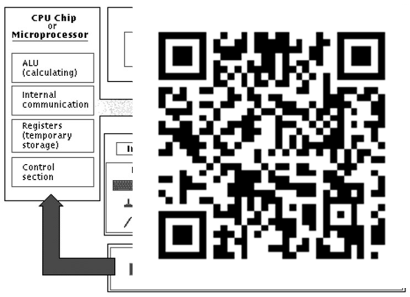
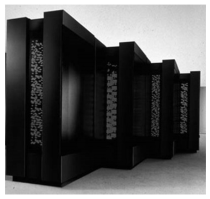
Learning; comprehension; & introspection

COMP25111
Operating Systems
Lectures 13:
Virtual Memory (3)



Dr Richard Neville
r.neville@manchester.ac.uk
Room: G12 Kilburn Building, Bottom floor



Week
8

NOTE: The **up-to-date** version of this lecture is kept on the associated web site – available [on-line] @ Blackboard select: COMP15111 Introduction to Computer Systems **www.manchester.ac.uk/portal**

http://www.cs.man.ac.uk/~neville/COMP25111/Lecture13/Lecture13.html

Learning; comprehension; & introspection

Where to find this Lecture 13 of the COMP25111 course?

First Go to Blackboard 9; then select: COMP25111 Operating Systems

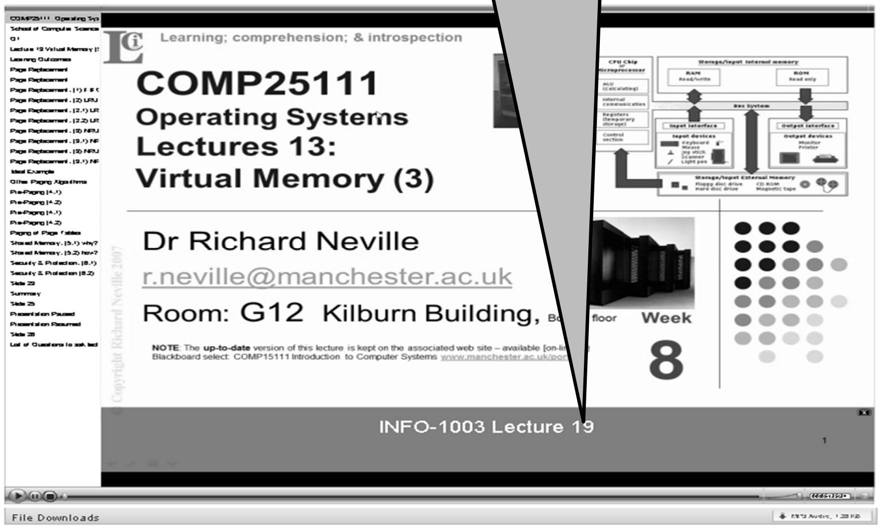
Then select: Week 8
This topic provides...
12: Memory management 3 (Virtual Memory (2)) by RN;
13: Memory management 4 (Virtual Memory (3)) by RN.

Do not worry about the number "19" – keep watching the lecture and you will see it is actually"13"!!!

Then select: Lecture 13 Information

Then select: Real Time Video of Lecture13

Then select:



COMP25111 Operating Sys
School of Computer Science
Q1
Lecture 13 Virtual Memory (3
Learning Outcomes
Page Replacement
Page Replacement
Page Replacement . (1) FIFO
Page Replacement . (2) LRU
Page Replacement . (2.1) LR
Page Replacement . (2.2) LR
Page Replacement . (3) NRU
Page Replacement . (3.1) NF
Page Replacement . (3) NRU
Page Replacement . (3.1) NF
Ideal Example
Other Paging Algorithms
Pre-Paging (4.1)
Pre-Paging (4.2)
Pre-Paging (4.1)
Pre-Paging (4.2)
Paging of Page Tables
Shared Memory . (5.1) why?
Shared Memory . (5.2) how?
Security & Protection . (6.1)
Security & Protection (6.2)
Slide 23
Summary
Slide 25
Presentation Paused
Presentation Resumed
Slide 28
List of Questions to ask lect

Learning; comprehension; & introspection
COMP25111
Operating Systems
Lectures 13:
Virtual Memory (3)
Dr Richard Neville
r.neville@manchester.ac.uk
Room: G12 Kilburn Building, Bottom floor
Week
8
NOTE: The up-to-date version of this lecture is kept on the associated web site – available [on-line] @ Blackboard select: COMP15111 Introduction to Computer Systems www.manchester.ac.uk/portal
INFO-1003 Lecture 19
File Downloads

Students Copy

Short Exam Questions **Q1**

1. Question

State the **three issues** Segmented Virtual Memory support with respect to the computer system as a whole.

ANSWER(S):

1)
2)
3)

Answer(s):

- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture.

Now on Blackboard:

Lab3 Covers: Lab3 Exercise MSWord document

Lab 3 Folder Contains:

1. MMUSim.java; and
2. skeleton MMU.java [Java files]

Lab 3 Hints Folder: [#H1] and [#H2]

Lecture 13 Virtual Memory (3)

Efficient Use of Main Memory

Towards Modern Operating Systems

Introduction to:

- Page Replacement Policies

Reading: MOS;

Reference: Modern Operating Systems (MOS) Andrew Tanenbaum,

Learning Outcomes

D-words

C-words

Be able to

1) Discuss the concept of 'Page Replacement'
2) Differentiate between First in First Out (FIFO) and Least Recently Used (LRU)
3) Explain Not Recently Used (NRU)
4) Describe Pre-Paging
5) State why and how Shared Memory is used
6) Assess Security & Protection issues related to memory

FOOTNOTE2: D-words: direction words. C-Words: content words.
Ref.: Michael J. Wallace (1980, 2004)
Study Skills in English, ISBN 9780521537520.
D-Words also aligned to; Ref.: Taxonomy of Educational Objectives:
The Classification of Educational Goals; pp. 201–207; B. S. Bloom (Ed.)
Susan Fauer Company, Inc. 1956.

Footnote
1: Discuss: aligned to describe, paraphrase, discuss & give example.
2: Differentiate: aligned to analyze, categorize, compare and contrast.
3: Explain: aligned to describe; discuss; give examples.
4: Describe: aligned to paraphrase, discuss & give example [pictorial or diagrammatic].
5: State: aligned to define, identify.
6: Assess: aligned to evaluate and interpret.

Reference: Bloom, B.S., *Taxonomy of Educational Objectives*. Handbook I: The Cognitive Domain. 1956: New York: David McKay Co Inc.

Page Replacement

- Recall the MMU operation:
  1. Given a Virtual Address;
  2. Extract Virtual page number;
  3. Look at corresponding page table entry;
  4. If page is in real memory, access it;
  5. else access the disk to read wanted page and find a place to put it in real memory. This will probably involve finding a page to reject to disk.

(p)	(i)
...	...
07	00
06	XX
05	03
04	02
03	XX
02	01
01	XX
00	XX

Start M E

Page Replacement

- So which page to reject?
  - This can have a dramatic effect on performance.
- If we could predict the page which would be accessed again furthest in the future (maybe never) this would be the ideal one to pick!
- But (in any real program with dynamic behaviour) this is impossible.

Page Replacement: (1) FIFO

- First in First Out (FIFO)
  - Identify the oldest page in real memory and get rid of that;
    - they can be tagged with a **sequence number**.
  - But just because it's old doesn't mean that it isn't wanted!
    - It may contain a critical bit of code which is continually in use.
  - Can be improved using **second chance** algorithm – see MOS4.4.

Learning; comprehension; & introspection

Page Replacement: (2) LRU

A few additions to notes are required.

- Least Recently Used (LRU)
  -

Inverse of LRU is:

- If something has been used recently then it probably will be again. And the converse.
  - Simplest implementation is to associate a timestamp counter (may need 64 bits) with every page, updated whenever the page is accessed.

timestamp is the naming convention used in the Lab exercise.

Footnote 1: Temporal locality: The concept that, likelihood of referencing a resource is higher if a resource near it was just referenced.

Learning; comprehension; & introspection

Page Replacement: (2.1) LRU Continued

LRU Page Replacement timestamp

- But, if needed on every access, cannot be done efficiently in memory.
- Instead add **counter** to each TLB entry

Translation Lookaside Buffer

- But this added **counter** is:
  - Expensive hardware and can triple size of each page table entry.

Reference Lecture 13 Self-study NOTES: Translation Lookaside Buffer notes.

Page Replacement: (2.2) LRU Example

Physical Memory

Analogues to: **Virtual Memory**

1) Real memory has space for 3 page frames only
2) 5 pages (say 1,2,3,4,5) initially on disk [not in memory]
3) Requests in the order: 1 2 3 4 1 2 4 3 5 4

Requested by 'Execution Order' say…

Requests order →

	1	2	3	4	1	2	4	3	5	4
3 page frames only	1	2	3	4	1	2	4	3	5	4
		1	2	3	4	1	2	4	3	5
			1	2	3	4	1	2	4	3
	M	M	M	M	M	M	H	M	M	H

LRU

Requests order

Time Since used ↓

M=miss

H = hit

Page Replacement: (3) NRU Not Recently Used

- Recently used pages kept in memory; this infers not recently used pages replaced.
- The R and M bits in a page table are used:
  - The R bit set when referenced.
  - The M bit set when modified (written to).
- How it works:
  1. At fixed intervals, the clock interrupt triggers and clears the referenced bit (R = 0) of all the pages.
  2. Referenced bit marks pages referenced in interval.
  3. So during interval if page referenced R=1 then it is **used**.
  4. If not R=0 then it is **NOT used**.
     - and at the end of interval is a candidate for replacement.

R bit page 0: **0**0001111
R bit page 1: **0**0000000

- Note: NRU is a crude approximation to LRU.

Reference: R bit is the referenced bit (read or written) – set if the page frame is referenced. Other bits in the page table are: Modified, Present/absent + set of protection bits (R,W,X).

Page Replacement: (3.1) NRU Not Recently Used - Continued

- When replacing a page, the operating system divides the pages into four classes:

Question: State what classes '0' and '3' represent?

Not Recently Used

Class	R bit (referenced)	M bit (modified)
0	0	0
1	0	1
2	1	0
3	1	1

Recently Used

- The NRU algorithm picks a random page from the lowest class for removal.
  - Algorithm implies that a referenced page is more important than a modified page.

Idealised page replacement algorithm

Ideal Example

Belady's optimal replacement algorithm

Order: 1 2 3 4 1 2 4 3 5 4

Look forward to 'furthest future' use

3 page frames only

1	2	3	4	1	2	4	3	5	4
1	2	3	4	4	4	4	4	4	4
	1	2	2	2	2	2	2	5	5
		1	1	1	1	1	3	3	3
M	M	M	M	H	H	H	M	M	H

Requests order

M=miss

H = hit

S Middle E

Other Paging Algorithms

- There are a number of refinements to the LRU principle resulting in other algorithms.
- To emphasise, because any page fault causes a disk access, it takes time (***mS***) compared to ***nS*** instruction execution.
- Minimising page faults by using sophisticated replacement algorithms is worthwhile.

Pre-Paging (4.1)

- If a swapped out process is brought back into memory, we could let it reload its new pages one by one.
- However, it will be much more efficient if we keep track of the process' **working set**
  - i.e. group of recently used pages, and
- load these in one go before re-starting the process.

Footnote 1: Pre-paging: A technique whereby the operating system in a paging virtual memory multitasking environment loads all pages of a process's working set into memory before the process is restarted.

Pre-Paging (4.2)

- We may also be able to keep track of groups of pages that get used together when a process is running.
- Rather than just load one at a time we can load a **group** together.
- Obviously need to displace others, so need to be careful. But used in practice (Linux 2.4 + kernel).

Paging of Page Tables

- Page tables are just locations in memory which exist on pages.
- They can themselves be paged out to disk.
- However, when a process is running, its page tables must be in memory.
- In practice, only those parts of the table which hold translations for the page tables!

Shared Memory: (5.1) why?

A few additions to notes are required.

- In some cases it is necessary for processes to share memory:
  1)
  2) Shared data space (e.g. Unix pipes); or
  3) Shared Library Code (dll's, i.e. dynamic link libraries).
- If each process sees only its own version of virtual memory space, how can this happen?

Shared Memory: (5.2) how?

- Page tables can be set up so that (possibly different) virtual addresses in multiple processes translate to the same real page.
- This would normally be organised by the OS.
- [But] Allowing user processes to alter their own page tables would clearly be a major security issue.

Security & Protection: (6.1) Reason

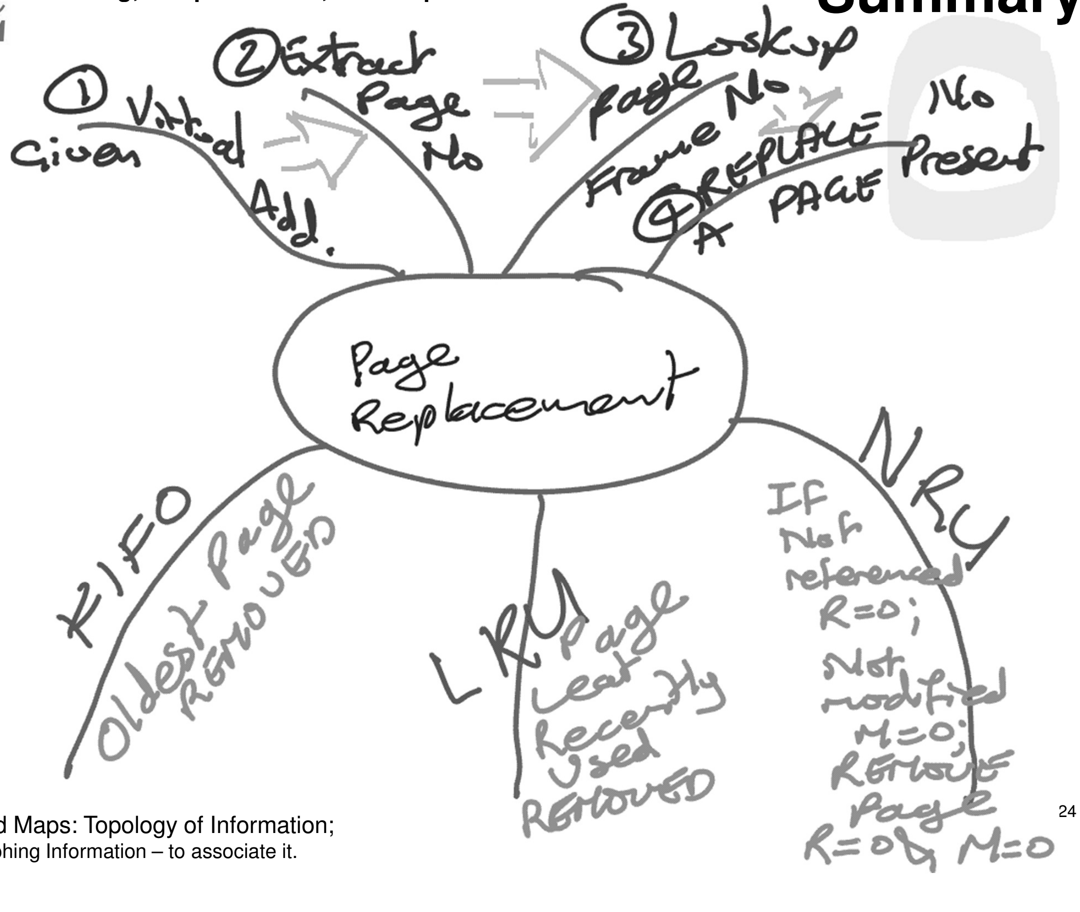
- We have seen that a page table entry can contain permission information (R,W,X).
- This can be used to control access;
- for example:
  1) Code can be marked 'read only'; (R,W,X)=(1,0,0)
  2) Page tables can be marked 'read only'; or (R,W,X)=(1,0,0)
  3) Data can be 'read/write' but **not** 'execute'. (R,W,X)=(1,1,0)
     - avoid accidental or malicious execution of data.

Security & Protection (6.2)

- But pages are there to help manage the address space.
- They are not natural units of protection. E.g.
  1) Code Area;
  2) Data Area; or
  3) Stack Area.
- Some (but not all) systems instead have an additional level of virtual memory.

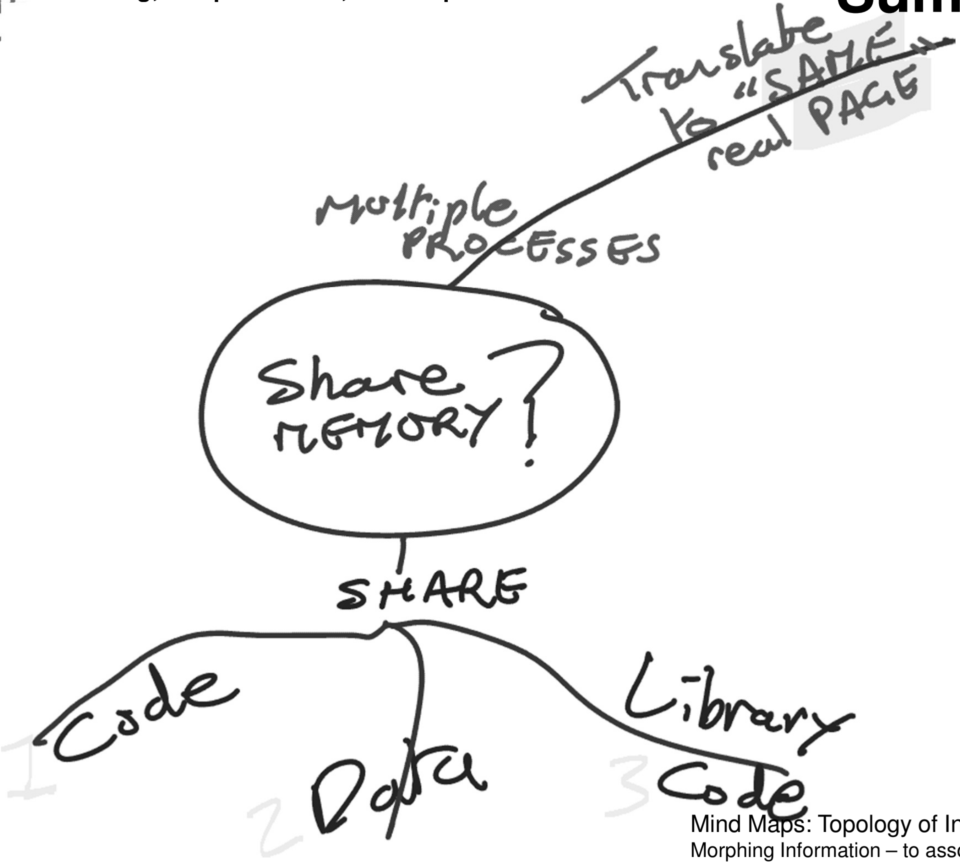
END

Summary

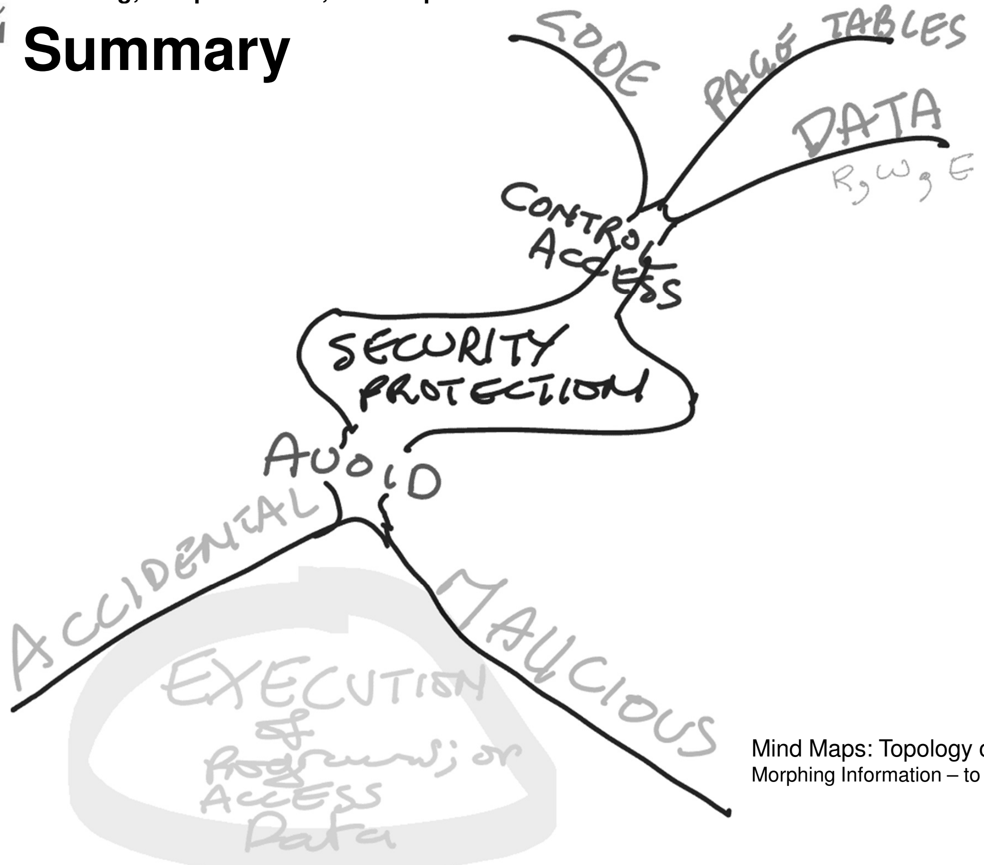


Mind Maps: Topology of Information;
Morphing Information – to associate it.

Summary



Summary



List of Questions to ask lecturer

- Before the 9a.m. start lecture the lecturer will be half an hour early and you can ask [any and all] questions in that half hour; before the lecture:

1.

2.

3.

4.

5.

Getting ready for next week
Do next week’s Q3’s NOW

- Once you have re-read the lecture notes; and listened to the audio recording [while stepping through the PPT] of the lecture again:
- Please have a think about next week’s Q3’s
  - on the next page
- If you try to answer the Q3’s now you will be in a much better position to recall the information.
- Once you have done this, transfer your answers to next weeks “Student [OWN answers] version” at the start of next weeks lecture.
  - YES this implies bringing the last weeks lecture notes to the next lecture …

Learning; comprehension; & introspection

Now on Blackboard:
Lab3 Covers: Lab3 Exercise MSWord document
Lab 3 Folder Contains:
MMUSim.java; and
skeleton MMU.java [Java files]
Lab 3 Hints Folder: [#H1] and [#H2]

THIS week's

Short Exam Questions Q3

1. Question
   State the **three issues** Segmented Virtual Memory support with respect to the computer system as a whole.

   Answer(s):

2. Question
   Give two reasons why 'segments' support the management of the execution of multiple processes in an operating system :

   Answer(s):

3. Question
   Give the names of the two fields the segment address is split into.

   Answer(s):

●

- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture

Learning; comprehension; & introspection

Next lecture [week's]

Short Exam Questions Q3

1. Question
   Name three page replacement policies?

   Answer(s):

2. Question
   What is meant by the term 'working set;' with respect to process memory pages?

   Answer(s):

3. Question
   What methodology can minimise page faults by the use of sophisticated replacement algorithms?

   Answer(s):

- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture

Glossary

- Why build a Glossary for each course unit you undertake?
- It is imperative that the correct terminology [keywords] are utilised in context in your exam answers; this is so important that the lecturer has added glossaries to each paper copy of your lectures. It is of such importance that in your notes [prior to the start of the glossary] the following advice is given:
  - Each module you undertake uses its own jargon.
  - This can be a problem for new students, whom are trying to comprehend the new domain knowledge attached to a particular new module.
  - One way to get to know the new jargon is to build your own GLOSSARIES for each course module.
  - The glossary on the next few pages is a starting point for this module [unit].
  - Please feel free to add to the glossaries throughout the unit…
  - The glossary is full of potential exam questions of the form "define X" or "briefly explain X."
- Please heed the advice in the future; even if your lecturers do not supply a glossary build your own as without knowledge of the appropriate terminology [keywords] when expounding your knowledge you will not be viewed as comprehending the details of any theory.

GLOSSARY

- Using the on-line resources and any other resources compile a glossary of the terms below:
- Page replacement →
- Virtual address →
- Page table →
- FIFO →
- LRU →
- Timestamp counter →
- Page frames →
- Page number →
- Working set →
- Translations →
- User code →
- Shared data space →
- Shared library code →
- Permission information →

Learning; comprehension; & introspection

Learning Resources 1

- **Descriptions [Theory] (in text books)**
- Remember the key issues, highlighted in GREEN, are the concepts to look for in any book:
  - Section on page replacement algorithm, first-in-first-out replacement algorithm, least recently used replacement algorithm in chapter 6 the Central processor unit operation in: Chalk BS, Carter AT, Hind RW (2004) Computer Organisation and Architecture: An introduction 2nd Edition, Palgrave, ISBN 1-4039-0164-3 .
  - Section on page replacement algorithm, FIFO, LRU, NRU, Pre-paging, Security, protection [memory] – in chapter 4 the Computer Systems Organization; & Parallel Computer Architectures in: Modern Operating Systems, 2/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, Publisher: Prentice.
- **Web resources:**
  - **Virtual Memory; available [on-line] @ http://courses.cs.vt.edu/~csonline/OS/Lessons/VirtualMemory/index.html**
  - **Cache Simulator Lab Session 1; available [on-line] @ http://myweb.lsbu.ac.uk/~chalkbs/research/CacheSimDescription.htm**
  - **Virtual Memory Page Replacement Algorithms [good set of references at the end]; available [on-line] @ http://people.msoe.edu/~mccrawt/resume/papers/CS384/mccrawt_cs384_virtual.pdf**
  - **MOS Free e-book [Low resolution (Not high quality graphics or printing – but readable)]:**
  - **Modern Operating Systems (MOS) 2nd Edition Andrew Tanenbaum, available**
  - **[on-line] @: http://www.freebookzone.com/fetch.php?bkcls=os_thry&bkidx=35**
  - Memory ***Replacement*** Policies **; available [on-line] @ *www.cs.uiuc.edu/class/fa08/cs241/lectures/24-Memory.ppt***

Learning; comprehension; & introspection

Questions

Introduction to Questions:

The set of questions are based on lecture 13.

Answer Sheet will be given later in year and will contain the answers to these questions.

- Remember to find detailed and comprehensive answer you should [also] reference associated text books in the library.
- A reasonable starting place for associated book titles are:

1) This units ‘module guide’; given to you in RN’s first lecture – or on the web [Blackboard];
2) Those books mentioned in ‘Background Reading;’
3) Those books [and web resources] mentioned in Learning Resources.

Questions

1. Question:

- Draw up a table that lists:
- 1) [Page] Replacement policy name; &
- 2) Brief description of how the policy works.

Questions

1. Answer:

Policy name	Description of how the policy works

Answer(s):

Questions

2. Question

State three reasons why it is necessary for processes to share memory.

Questions

2. Answer

Answer(s):

Questions

3. Question

   In the context of permission information: like read (R), write (W), & execute (X).

   What can the RWX permission information control access to; with respect to security and protection?

Questions

3. Answer

Answer(s):

4. Question

On a paged machine with 3 pages frames available for it, a particular process makes accesses to the following pages in the order given:

0, 7, 3, 1, 2, 3, 1, 7, 3

Show the contents of the 3 page frames and the cumulative total number of page faults after each memory access assuming that an LRU page replacement algorithm is in use and that the page frames are initially empty. The type of diagram you should draw up is depicted in figure below:

Access:	0	7	3	1	2	3	1	7	3		
Most recent:	X	X	X	X	X	X	X	X	X	)	Contents of
Second most :	X	X	X	X	X	X	X	X	X	)	page frames
Third most:	X	X	X	X	X	X	X	X	X	)	in memory
Total PFs:	X	X	X	X	X	X	X	X	X		

Typical diagram showing 3 page frames and the cumulative total number of page faults.

4. Answer

For 0, 7, 3, 1, 2, 3, 1, 7, 3:

Answer(s):

Questions

5. What is a page replacement algorithm?

Answer(s):

Revision Exercises

- Scan read Lecture 13’s Questions.
  - Answer Lecture 13’s Questions
    - Particularly those questions you had difficulties with when you first tried them.

Background Reading

- [1] Computer Organisation and Architecture: An introduction 2nd Edition, Palgrave, ISBN 1-4039-0164-3 chapter 6 the Central processor unit operation in: Chalk BS, Carter AT, Hind RW (2004).
  - Section on page replacement algorithm, first-in-first-out replacement algorithm, least recently used replacement algorithm.
- [2] Modern Operating Systems, 2/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, Publisher: Prentice
  - Section on page replacement algorithm, FIFO, LRU, NRU, Pre-paging, Security, protection [memory].
- {On Web} **4.4 First-In, First-Out (FIFO),** Ref. **Page replacement in operating system memory management, by Heikki Paajanen,** available on-line from [URL]: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/12528/URN_NBN_fi_jyu-2007775.pdf, (last visited [accessed]: 21-7-2011).
- {On Web} **Memory Management: Virtual Memory,** Ref. Understanding Operating Systems, Fifth Edition available on-line from [URL]: http://www.johnrouda.com/class/PDF/CPT%20257/01600_IM_ch03.pdf, (last visited [accessed]: 21-7-2011).
- {On Web} **Part 2: Paging Algorithms and Implementation Issues**, Ref. Chapter 4: Memory Management, on-line from [URL]: http://www.cs.pitt.edu/~mosse/cs1550/Slides/amer-memory2.pdf, (last visited [accessed]: 25-7-2011).
- {On Web} **4.4 PAGE REPLACEMENT ALGORITHMS,** Ref. MEMORY MANAGEMENT, CHAP. 4 , on-line from [URL]: http://www.lira.dist.unige.it/teaching/OS/files/sample-4.pdf, (last visited [accessed]: 25-7-2011).
- {On Web} **,** Ref. , on-line from [URL]: , (last visited [accessed]: 25-7-2011).

COMP25111 Exercise 3:

Simulation of Paging Behaviour

Contains a copy of the "traceA" file.

Before you start this PLEASE read the multimedia PPTs and audios on the Blackboard 9 portal- on the Blackboard 9 Web Site; select: COMP25111 Operating Systems
https://www.portal.manchester.ac.uk/; then
1/ Go to COMP25111 Operating Systems 2011-12 1st Semester;
2/ Then Lab3_RN
3/ Then Lab3 Hints
4/ Then read through as many hints as you need; remember the methods, processes, naming conventions, and implicit theory outlined in the hints will be useful to you in your final year projects where you should think about using these techniques…

Duration: 1 Session

1. Learning Outcomes

On completion of this exercise, a student will:

- Have implemented the functionality of a simple Memory Management Unit (MMU) for a paged virtual memory system in Java.
- Have exercised the MMU using a trace file of memory accesses using a Java program which initialises the MMU and calls methods within it to perform read and write actions.
- Have produced statistics, which show the behaviour of the MMU with particular reference to page faults and page rejections.

2. Introduction

Virtual memory is a technique for providing a process with apparent access to the whole addressable memory space of a processor in circumstances where the real memory available may be considerably smaller. One advantages of the scheme is the process can use larger code and data sizes than would otherwise be possible. Another is the run-time layout of code and data can be considerably simplified if it does not have to fit into a confined space. This is particularly true for dynamic data whose size is unknown until run-time.

Virtual memory relies on the fact that, in the majority of programs, the use of both code and data exhibits both *spatial locality* and *temporal locality*. That is that at any point in time, the program is usually only using a small fraction of its code and data and that, over a short time period, the usage will not change significantly.

A virtual memory system therefore tries to keep currently used code and data in the (limited size) real memory of the system while the rest (if it exists) is kept on background storage such as magnetic disk. In order to relieve the programmer from the complex task of working out which data should be where at any point in time, the system arranges to move the data between real memory and disk automatically as required.

A piece of hardware know as a Memory Management Unit (MMU), together with system software within the Operating System provide all the functionality needed to

do this. The purpose of this exercise is to implement the functionality of a MMU in order to gain an understanding of the principles.

3. Paged Virtual Memory

Devices such as disks usually store and retrieve data in blocks of hundreds or thousands of bytes. Accessing a block is often slow compared to CPU memory speeds but once accessed; the contents of that block can be read or written very rapidly. In these circumstances, it makes sense to move data in units which are larger than bytes or CPU words. Virtual memory systems therefore operate at the unit of a *page* when transferring data to and from memory and disk. A page does not necessarily correspond to a disk block size but is fixed for a particular MMU typically between 4 and 64 kilobytes. This will usually correspond to several disk blocks.

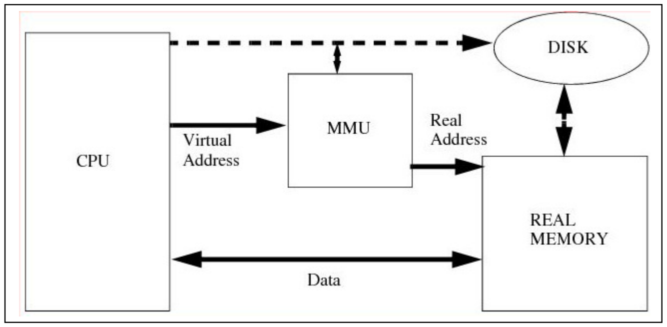


Figure 1 shows the basic structure of a paged virtual memory system.

The addresses issued by the CPU are virtual addresses which are the full width of the processor's addressing capability. In a modern 32 bit processor (for example ARM or x86) these will be a full 32 bits capable of addressing 4Gbytes of memory. The real memory will usually be smaller, typically 512 Mbytes on a laptop.

Each of these memory spaces is considered to be divided into pages. Note that this is not reflected in the internal structure of any memory, it is just an abstract division; the memory addresses are still binary values which cover a linear address space (usually at the unit of bytes). However, we can view any address as being divided into two sections, one which addresses the page and one which is an offset within the page which addresses individual units (bytes). By convention, we usually refer to the virtual address as having *virtual page numbers* (or just page numbers) and the real address as having *page frame numbers.* Figure 2 shows this pictorially.

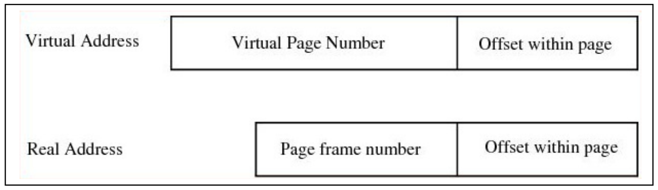


Figure 2 shows this pictorially view of *virtual page numbers* and *page frame numbers*.

The function of the MMU is to keep track of which virtual pages exist in real memory and, which on disk. It does this by keeping a *page table* of every possible virtual page number which contains the page frame numbers of any real copies. It is important to note that the offset is common to both virtual and real addresses. In order to find the data associated with a virtual address it is only necessary to take the virtual page number and replace it by the page frame number (if it exists). The page frame number is looked up in the page table to produce the real address which can then be used to access real memory. This is usually referred to as *address translation.*

If the table indicates that a virtual page does not exist in real memory this is called a *page fault.* It is necessary for the MMU to initiate a transfer of the page's data from the background storage (disk). The exact mechanism for this is not relevant here but it should be noted that this is a relatively slow process and the CPU itself can organise most of the transfer.

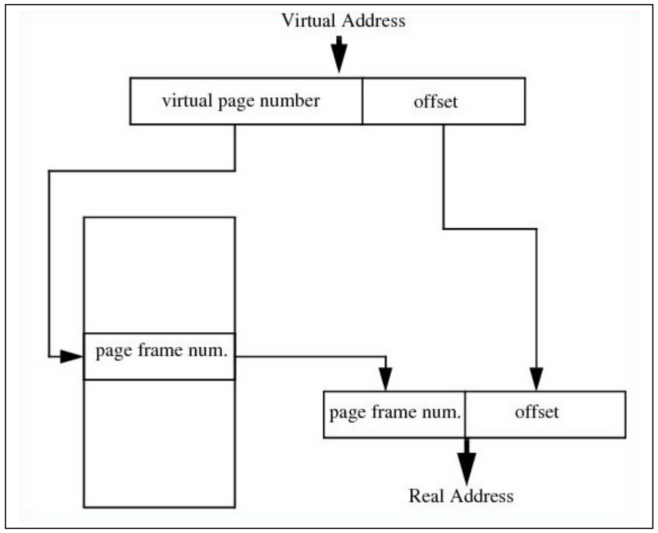


Figure 3 shows more detail of the MMU table and the address translation operation.

The page table is shown here as a single linear structure with the lookup done by the full page number. For large virtual address spaces, this may not be practical and more complex techniques may be required (this is covered in the course lectures).

4. Page Rejection

If there is a page fault, a place needs to be found in real memory for the page copy. Initially, assuming all real memory is not in use, it is possible to keep allocating page frames linearly through the address space. However, if all real memory is in use, we must find a page to reject. We must then write its contents back to disk before reallocating its page frame number to the page access, which has caused the fault.

We need to decide which page to choose using a *page rejection algorithm.* The most commonly used is an algorithm called LRU (least recently used). It works on the simple principle that the page, which has survived the longest time without being accessed, is a good candidate for replacement.
One useful optimisation is to note if a page has been written to since it was bought from disk into real memory. If not, there is no need to write it back to disk before reusing the real page frame. This is often true for pages which contain code.

5. The Exercise

The exercise is composed of two parts:

Part One: implements the majority of the code and the LRU page replacement algorithm. The majority of the (theoretical) advice below will aid you in this task;

after you have designed, developed, and implemented the program utilise the two trace files: **traceA** and **traceB**; to test your program.

Part Two: implements the a second algorithm the FIFO page replacement algorithm. The advice below will not aid you in this task. Nevertheless, it is expected that once you have implemented (and tested) the LRU algorithm you will have gained sufficient knowledge to enable you to undertake the FIFO design, development and testing on your own. After you have designed, developed, and implemented the program utilise the trace file: **traceLRUandFIFO**; to test your program. The FIFO algorithm has been covered in your lecture series and it is covered extensively in your course books; also see the Learning Resources (book & Web resources) as well as the Background Reading (book & Web resources) at the end of Lecture COMP25111 Operating Systems Lectures 14: Virtual Memory (3) for further reference material.

Introduction to Exercise

The purpose of the exercise is to write Java code to implement a simple MMU, which contains a page table and code to perform the necessary actions when read, or write accesses occur to a virtual memory address. Because the exercise is using a trace of addresses rather than executing a real program, there is no need to perform any real memory accesses. Instead, it is simply necessary to access the page table to determine if there is a page fault and, if so, find a free page frame to use and place it in the table. If, initially, there are unused page frames these can be allocated linearly until there are none free. At that point it will be necessary to find a page to reject and use its page frame for the newly required page.

You should implement a LRU algorithm to determine page rejection. To make this simple, you should have a 'timestamp' in each page table entry, which is updated appropriately, when a page is accessed.

You should also include a 'dirty' indication in the page table implementation, which is set on a write. This indicates if the page needs to be written back to disk if it is rejected.

Note that a practical page table will usually also contain page access information which determines the types of access which are allowed to a particular page (for example read only). There is no need to include such information for the purposes of this exercise.

In appropriate Blackboard subfolder, you will find a program harness which, after creating a MMU object, reads the trace of store accesses and makes calls to read and write methods within that object until the trace file ends. It then prints out some statistics:

Total Accesses
Total Instruction Fetches
Total Page Faults
Total Page Rejections (i.e. ignoring startup page faults)
Total Page Writebacks (to disk)

The first two are included in the given program; the last three need to be calculated by your MMU implementation. (A skeleton MMU.java file is provided in the appropriate Blackboard subfolder)

The trace file format is a line for each access. The first number indicates the access type (0=read, 1=write, 2=fetch) and the second is a (hex) 24 bit memory address. Within the program, the address is considered to be a 12-bit page number and a 12 bit offset representing 4k pages each of 4k bytes, i.e. 16Mbytes in total of virtual memory

space. This is deliberately small so that your implementation can use a simple single level page table. Although obviously not representative of modern systems, this is typical of a virtual memory system in computers of the 1960s when virtual memory was invented (in Manchester University on the Atlas computer [1][2]).

An example of 'traceA' file is given below:

```
2 400000
0 600000
2 400001
1 600001
2 400002
0 700000
2 500000
0 700001
```

The first line "2 400000" is decoded as follows:

First number	Second number
2	400000
2=fetch	0100 0000 0000 0000 0000 0000_{HEX}
This is a fetch access	24 bit hexadecimal memory address

The following decode the first number of each of the eight lines in traceA:

First number	Type of access
2	2=fetch
0	0=read
2	2=fetch
1	1=write
2	2=fetch
0	0=read
2	2=fetch
0	0=read

The program given makes a single call to the method which performs the trace simulation using a very very small real memory (2 [real] page frames, i.e. 8k bytes [in MMUSim.java the real memory sizes (in pages) is set to "rmem_pages = 2;"). This should be run with the file traceA when initially debugging your implementation. This file contains only 8 store accesses which have been devised so that you should be able to predict what statistics your MMU should produce.

When you think this is working, you should modify the program to perform simulations using both 32 and 64 real page frames on traceB. This represents 128k and 256k bytes of real memory, again realistic in the 1960s.

6. Results

You should be prepared to demonstrate the results of the runs on traceA and traceB when your exercise is marked. To help you determine if you have got a correct implementation, the total number of page faults for 32 pages and traceB should be 227.

You should also be prepared to describe your code and draw relevant conclusions from the results.

Assessment

COMP20051 Exercise 3: Developing Simulation of Paging Behaviour

Demonstrators please fill in the student's full detail [below] before starting the assessment; may be the best way to do this is to get the students to fill this in themselves:

Students full name	Student Number	Email address	Course studied

At the deadline, at the end of the Session, the marks awarded are as given in table 1.
Demonstrators please assign marks for each of the eight questions in boxes provided.

At the deadline, at the end of Session 5, the marks awarded are as follows:

Mark awarded for:	Exercise 3	
1. Show Correct page table [data] structure; plus comment on how you [the student] developed the data structure.	1	
Demonstrating an overall [software engineering] correct approach; of the algorithmic and combined data structure: **2 – 5**.		
2. State how REQUIREMENT development was undertaken.	1	
3. State how [if any] ABSTRACT DESIGN was undertaken.	1	
4. State how IMPLEMENTATION was undertaken.	1	
5. Demonstrating a methodology for TESTING the program; and comment on: **interpretation of results**.	1	
6. Showing the generation of an appropriate OUTPUT trace file: **traceA** files [displayed in real-time on command [Prompt] screen] (using LRU).	1	
7. Showing the generation of an appropriate OUTPUT trace file: **traceB** files [displayed in real-time on command [Prompt] screen] (using LRU).	2	
8. Showing the generation of an appropriate OUTPUT trace file: **traceLRUandFIFO** files [displayed in real-time on command [Prompt] screen] (using LRU and FIFO).	2	
TOTAL mark	10	

Remember to save these exercises in a directory nominally named …/COMP20051/ex3.

Do this exercise [or save this exercise] in a directory named COMP251111/ex3 directory, save your downloaded template [Skeleton(s)] code:-

MMUSim.java; and
MMU.java.

as well as the trace files:

traceA;
traceB; and
traceLRUandFIFO

… in COMP25111/ex3.

Remember to save your trace files of your solution.

The three files traceA; traceB; and traceLRUandFIFO; as well as the template [Skeleton(s)] MMUSim.java & MMU.java are downloadable from Blackboard in Folder Lab3.

References

[1] One-Level Storage System, T. Kilburn, D.B.G. Edwards, M.J. Lanigan, F.H. Sumner, IRE Trans. Electronic Computers April 1962

[2] Tom Kilburn (1956), The Atlas, School of Computer Science: information on Tom Kilburn's [computing] effort known as the MUSE (microsecond) computer , available [on-line] @ http://www.computer50.org/kgill/atlas/atlas.html, [Last accessed 12/10/09, 11:09].

7. Decoding the COMP20051 Laboratory Exercise 3; and '*supporting advice*'

The previous six pages document is a typical laboratory that you are normally set in the School of Computer Science. The first task is to decode [work out] what explicitly is required. As you have read the document once now reread section 5; this time highlighting or extracting the main requirements of the exercise.

Hint 1: [#H1]

[#H1] If you require a hint, download them from Blackboard. 'Hint No 1' hints at the requirements you should have data mined from reread section 5.

After reading document 'H1.doc' to validate you have extracted the relevant requirements you next step is to start the design phase that fulfils the requirements [#R1 to #R5] of the exercise. Then once designed the MMU class can be implemented.

Hint 2: [#H2]

[#H2] If you require a hint, download them from Blackboard. After you have compiled a list of requirements, you should now think about designing the MMU class. 'Hint No 2' suggests an approach you could take.

Reading document 'H2.doc' may help you move forward in the design phase or validate your design.

You should now be able to implement, test, and run MMUSim utilising traceA. Once you have reached this stage you can compare your program's trace output with a typical trace A output [<u>res</u>ult] available on Blackboard; the file is named 'traceAres.'

One of the final requirements of the exercise is to modify the program [MMUSim] to perform simulations using both 32 and 64 real page frames on traceB. To do this you should consider altering the number of real memory pages.

Finally, you should consider the best way to describe your code; when asked. In addition, you should think about drawing relevant conclusions from the results.

The ">>" operator

Final point: what does the operator ">>" actually do?

In the code snippet:

int addr = access.addr;

int vpage_no = addr >> 24 - vmem_bits; // assumes a 24 bit virt addr
*given vmem_bits = 12. Hence the actual value of '24 - vmem_bits' is "**addr >> 12**."*

In "doSimul(…)" in class MMUSim.Java. The operator ">>" is utilised. If we look at what happend

Values:
addr = 4194304
vpage_no = 1024

So what has happened?

The java operator "**>>**"

>> shift bits right with sign extension

It implies [with two objects '**x**' & '**y**']:

x >> y

Means: Shift Right - Signed

Explicitly means: Shift **x** to the right by **y** bits. Low order bits are lost.
Same bit value as sign (0 for positive numbers, 1 for negative) fills in the left bits.

Given the example addr = 4194304
which is specified for addr an interger (int) which is base (or radix) 10.

Converting 4194304_{10}
Converting to hexadecimal:
00400000_{16}
or radix 2 it is:
$0000\ 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000_2$

Hence given ***addr >> 12**."* After shifting right 12 places this is:

$0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 0000\ 0000_2$
which is:
00000400_{16}
or radix 2 it is:
1024_{10}

QED