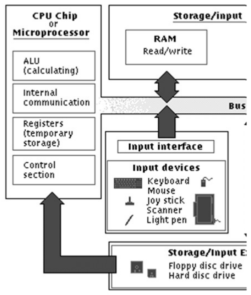
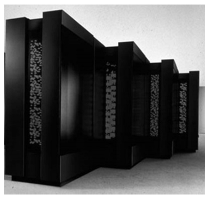
Audio visual - James Maclean [54038] 0161 306 4038

COMP25111
Operating Systems
Lectures 10
Memory Management (1)



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



Week
5

NOTE: The **up-to-date** version of this lecture is kept on the associated web site – available [on-line] @ Blackboard select: COMP25111 Introduction to Computer Systems **www.manchester.ac.uk/portal**

https://stream.manchester.ac.uk/Play.aspx?VideoId=9018

References:
1: Nova 840 (courtesy of Carl Friend) , available [on line] @ http://simh.trailing-edge.com/photos.html
2: Computer System, available [on line] @ http://encarta.msn.com/media_461532731/Computer_System.html
3: Connection Machine CM5, available [on line] @ http://hardware.localhost.nl/?ct=nth&showdir=pictures/2001-2003/Supercomputing

SkyDrive Real Time Video on
COMP25111 Lecture 11; "Intro To COMP25111 First Half" (Of the course), available on line @ WindowsLive_SkyDrive URL:
https://skydrive.live.com/#cid=802FD99601DE4751&id=802FD99601DE4751!279

Learning; comprehension; & introspection

First a Big Thank you to all of you whom submitted feedback that:

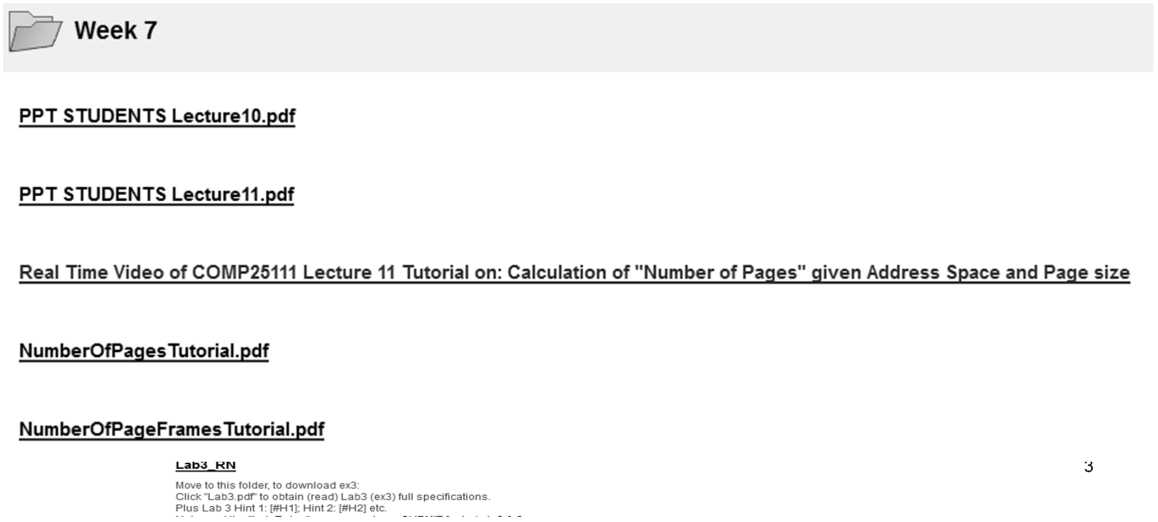
- I would like to thank all of you - on my behalf and on the behalf of all future students.
- YOUR FEEDBACK HAS HAD AN EFFECT;
- PLEASE KEEP GIVING "CS" YOUR THOUGHFUL FEEDBACK;
- TOGETHER, we can change CS for the better – THANKS.

COMP25111; URL; or HTTP address

- Web Site: **Blackboard select:**

  COMP25111 Operating Systems
  <u>www.manchester.ac.uk/portal</u>

- Will contain the overhead slides and aux. information
- Reference Textbooks covered in Unit guide, web, and the school web site.



Week 7

PPT STUDENTS Lecture10.pdf

PPT STUDENTS Lecture11.pdf

Real Time Video of COMP25111 Lecture 11 Tutorial on: Calculation of "Number of Pages" given Address Space and Page size

NumberOfPagesTutorial.pdf

NumberOfPageFramesTutorial.pdf

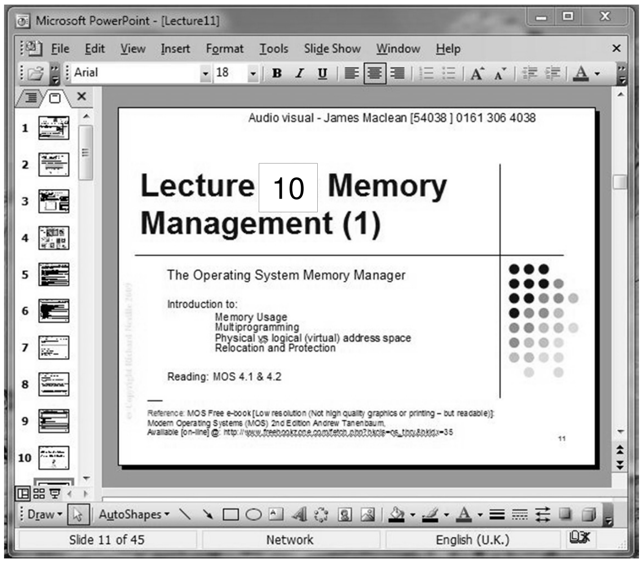
Lab3_RN

Move to this folder, to download ex3:
Click "Lab3.pdf" to obtain (read) Lab3 (ex3) full specifications.
Plus Lab 3 Hint 1: [#H1]; Hint 2: [#H2] etc.

Multimedia Notes & TuTorials

Click the PPT and Audio

PLEASE Listen to the multimedia PPTs and audios on: Blackboard.
They will make revision easier.


Microsoft PowerPoint - [Lecture11]
File Edit View Insert Format Tools Slide Show Window Help
Arial
18
Draw
AutoShapes
Slide 11 of 45
Network
English (U.K.)

Audio visual - James Maclean [54038] 0161 306 4038

Lecture 10 Memory Management (1)

The Operating System Memory Manager

Introduction to:
Memory Usage
Multiprogramming
Physical vs logical (virtual) address space
Relocation and Protection

Reading: MOS 4.1 & 4.2

Reference: 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

11

Then you can step through the PPT synchronously with the audio recording.

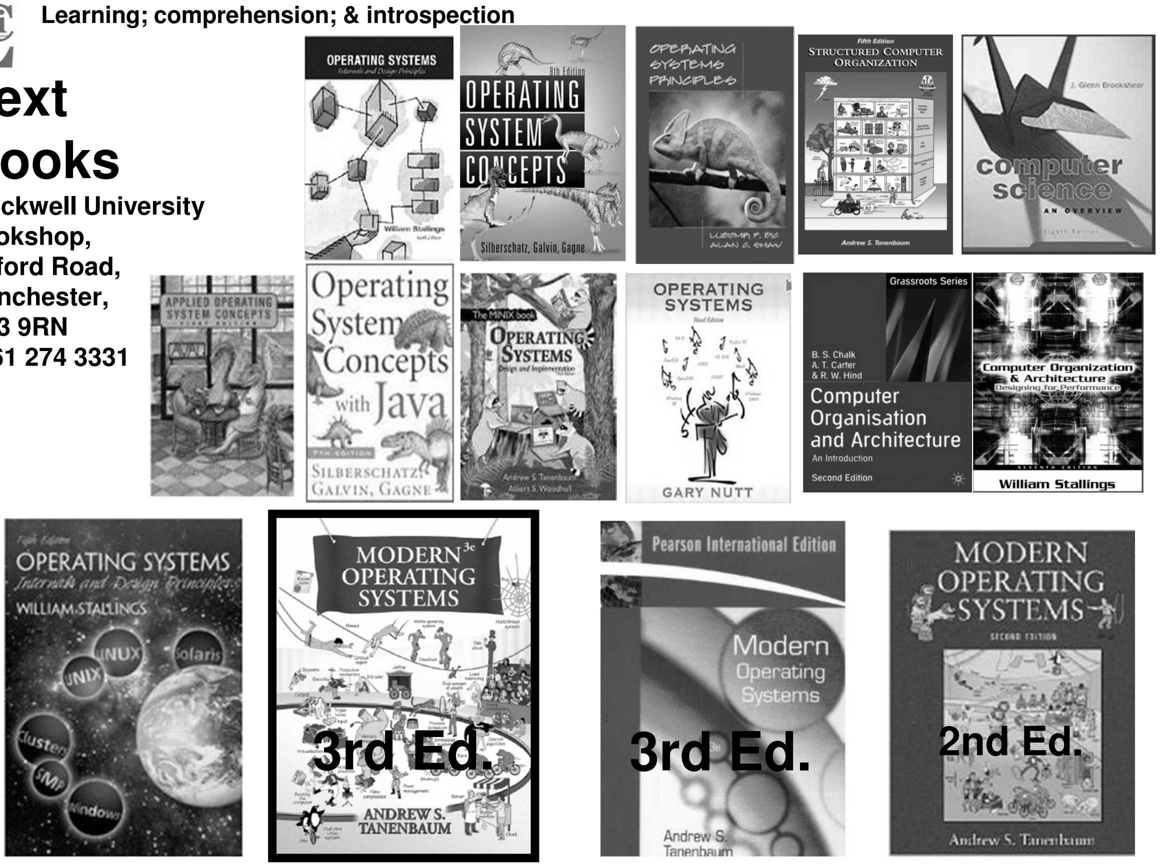
You can pause [the audio recording] at any time.

Best time to go through PPT &
audio is when you need to understand a concept in more
detail.

Use the multimedia PPTs & audios for REVISION too…

Text Books

Blackwell University Bookshop,
Oxford Road,
Manchester,
M13 9RN
0161 274 3331


OPERATING SYSTEMS
OPERATING SYSTEM CONCEPTS
OPERATING SYSTEMS PRINCIPLES
STRUCTURED COMPUTER ORGANIZATION
computer science AN OVERVIEW
APPLIED OPERATING SYSTEM CONCEPTS
Operating System Concepts with Java
The MINIX book OPERATING SYSTEMS Design and Implementation
OPERATING SYSTEMS GARY NUTT
Computer Organisation and Architecture An Introduction
Computer Organization & Architecture William Stallings
OPERATING SYSTEMS Internals and Design Principles WILLIAM STALLINGS
MODERN OPERATING SYSTEMS ANDREW S. TANENBAUM
3rd Ed.
Pearson International Edition Modern Operating Systems Andrew S. Tanenbaum
3rd Ed.
MODERN OPERATING SYSTEMS SECOND EDITION Andrew S. Tanenbaum
2nd Ed.

Feedback #1

"The feedback I received on my work (studies) was helpful" is the goal of this modules' real time and coursework feedback.

Real time feedback [**Level Three]:**

This type of feedback is two-way dialogue where learning can occur. Examples include the following times – when real time feedback is given [in real time]:

Time #1: Before the lecture starts;

Time #2: During the lecture break [half way through the lecture] - if time allows;

Time #3: At the end of the lecture; &

Time #4: Specific feedback periods specified by the lecturer.

Time slots #1 to #3 are normally in the lecture room specified [scheduled for the module].

Time slot #4 may be in the scheduled lecture room or in the lectures office.

Feedback #2

“The feedback I received on my work (studies) was helpful” is the goal of this modules’ real time and coursework feedback.

Types of feedback**:**

The feedback is normally pedagogical[1] and relates to several areas:

- Discussion of relevant [related] topics in lecture;
- Misunderstanding of theory – be it mathematical, concept, or basic;
- Reinforcement of theoretical beliefs;
- Reviewing a concept of theory discussed during the lecture – in a different way;
- Comprehension of process;
- Question answers discussion – either process or exact steps taken to evolve (calculate) answer;
- Lab work requirement;
- Lab work process;
- Lab work general questions.

Any other topic, concept, or theory relating to the module can also be discussed in a feedback session.

[1] **Pedagogical (pronounced: ped·a·gog·i·cal)**: relating to, or befitting a teacher (or student) or education (of student).

Problem logging into module web site

- Problem logging into module web site
- **contact ARS (**Action Request System) **[fill in web form]:**
- Action Request System http://csis.cs.man.ac.uk/
-
- **Was: dutyoffice@cs.man.ac.uk**
-
- **IMMEDIATELY**
-
- **e.g. if the web site [Blackboard] goes down the server may have crashed …**
- **The** Action Request System should be contacted
- **NOT the lecturer...**

Learning; comprehension; & introspection

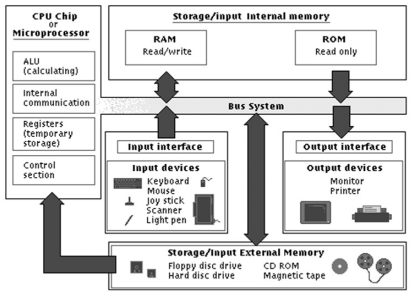
Copies of past exams

Copies of past exams

- 1/ Are at the end of the <u>module guide</u>;
- 2/ Are on the module web site at the end of the *.pdf version of the module guide – on the web site…

- You will be reminded of this when [if] you email me [a few days before the exam] and ask *where are the past papers?*

Audio visual - James Maclean [54038] 0161 306 4038



COMP25111

Operating Systems

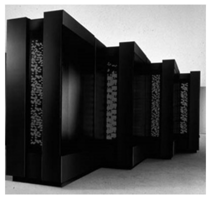
Lectures 10

Memory Management (1)

Dr Richard Neville

r.neville@manchester.ac.uk

Room: G12 Kilburn Building, Bottom floor



Week

5

NOTE: The **up-to-date** version of this lecture is kept on the associated web site – available [on-line] @ Blackboard select: COMP25111 Introduction to Computer Systems **www.manchester.ac.uk/portal**

References:
1: Nova 840 (courtesy of Carl Friend) , available [on line] @ http://simh.trailing-edge.com/photos.html
2: Computer System, available [on line] @ http://encarta.msn.com/media_461532731/Computer_System.html
3: Connection Machine CM5, available [on line] @ http://hardware.localhost.nl/?ct=nth&showdir=pictures/2001-2003/Supercomputing

SkyDrive Real Time Video on
COMP25111 Lecture 11; "Intro To COMP25111 First Half" (Of the course), available on line @ WindowsLive_SkyDrive URL:
https://skydrive.live.com/#cid=802FD99601DE4751&id=802FD99601DE4751!279

Learning; comprehension; & introspection

Where to find this Lecture 10 of the COMP25111 course?

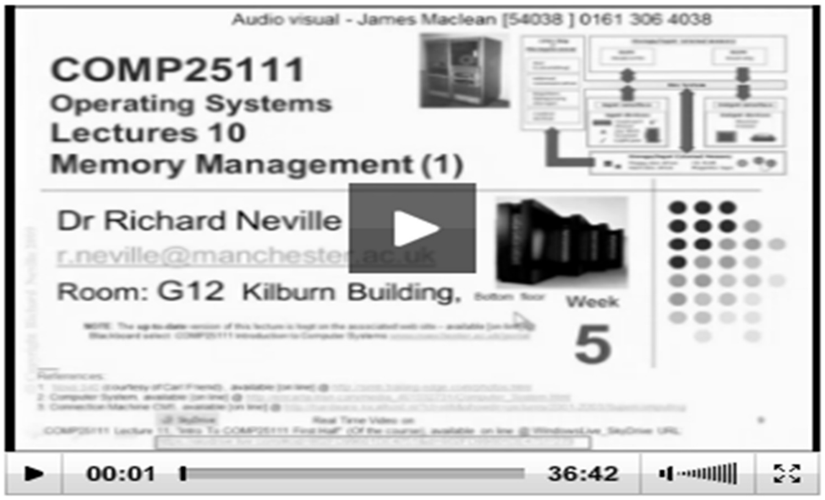
First Go to Blackboard 9; then select: COMP25111 Operating Systems

Then select: **Week 7**
This topic provides...
10: Memory management 1 (Introduction to basics) by RN;
11: Memory management 2 (Virtual Memory (1)) by RN;

Then select: **Lecture 10 Information**

Then select: **Real Time Video of Lecture 10**

Then select:



Audio visual - James Maclean [54038] 0161 306 4038

COMP25111
Operating Systems
Lectures 10
Memory Management (1)

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

Week 5

00:01 36:42

Learning; comprehension; & introspection

THIS week's Short Exam Questions Q1

Have a go answering Q1 question below:

1. Question
   Name, the three components that make up a <u>memory hierarchy</u> as used in computer systems:
   ANSWER(S):
   The three components are:
   1.
   2.
   3.

Answer(s):

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

Lab 3 RN

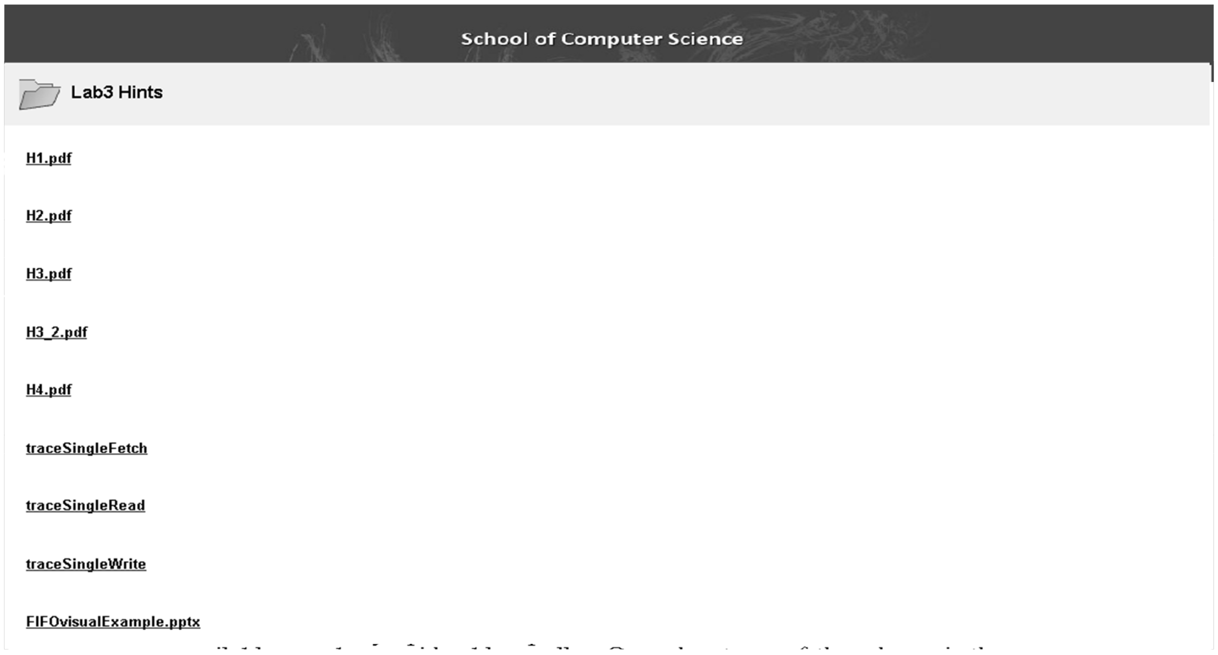
- If you are attending RNs Lectures L12 etc;
- You will be expected to undertake:
- Lab 3 RN:
- Time of lab:

Sem 1 A w3+ Lab LF31 Fri 09:00 - 11:00 F
Sem 1 A w3+ Lab LF31 Fri 11:00 - 13:00 I
Sem 1 A w3+ Lab G23 Mon 13:00 - 15:00 G

Lab 3 RN, cont. 1.

- All the information about the lab is available on Blackboard 9;
- Web Site: **Blackboard select:**

COMP25111 Operating Systems
www.manchester.ac.uk/portal


School of Computer Science

Lab3 Hints

H1.pdf

H2.pdf

H3.pdf

H3_2.pdf

H4.pdf

traceSingleFetch

traceSingleRead

traceSingleWrite

FIFOvisualExample.pptx

Lab 3 RN, cont. 2.

- Where to Find NetBeans on the lab machines:

1) Boot into Windows 7;
2) Click “Start;”
3) Click “All programs;”
4) Click the “NetBeans” folder [icon];
5) Click the “NetBeans IDE 7.0.1” icon.

SCENE SETTING – CONTEXT:

Problems:

Large Programs,
Large Data,
Large Libraries;

BUT:
“One advantage is that RAM is ~200 times faster than [hard] disk storage,
And typically enables :
Data access 50 to 100 times quicker”..

REASON:
“RAM has no moving parts and can access data with only a [file’s] memory
Address.
When data is on a spinning disk, … , the operating system must issue an interrupt, wait for the disk head to align [to correct sector & track], bring the data over [from] the controller [MMU] … and load data into RAM.”

CPU were:
32-bit systems provided ~4Gbytes of addressable memory…
Today’s 64-bit systems provide 2^{64} or ~18Ebytes (10^{18} Exa E)

Reference: Using In-memory analytics to quickly crunch big data, By Lee Garder, in IEEE Computer, October 2012, p. 16-21.

Audio visual - James Maclean [54038] 0161 306 4038

Lecture 10: Memory Management (1)

The Operating System Memory Manager

Introduction to:
- Memory Usage
- Multiprogramming
- Physical vs logical (virtual) address space
- Relocation and Protection

Reading: MOS 4.1 & 4.2

PLESE NOTE
Method Of assessment: 70% exam + 30% lab
Exam: coursework = 70:30
Exam is in semester 1. A number of lab's, worth 30%.

Reference: Modern Operating Systems (MOS) Andrew Tanenbaum,

Learning; comprehension; & introspection **Short [& Long] Exam Questions**

Learning Outcomes

D-words C-words

Introduction to Memory Management.

After the lecture student will be able to:

1) Describe the role of the memory allocation in an OS;
2) State where a simple single program fit in memory;
3) Discuss the issues with a single program;
4) Explain multiprogramming – fixed partitions;
5) Demonstrate an understanding of Relocation;
6) Differentiate between 'Relocation' and 'Relocation + Protection';
7) Discuss virtual addresses; &
8) Illustrate what is meant by 'Swapping'.

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: Describe: aligned to paraphrase, discuss & give example [pictorial or diagrammatic].
2: State: aligned to define, identify.
3: Discuss: aligned to describe, paraphrase, discuss & give example..
4: Explain: aligned to describe; discuss; give examples.
5: Understand(ing): aligned to outline; discriminate; categories; compare & contrast.
6: Differentiate: aligned to analyze, categorize, compare and contrast.
7: Illustrate: aligned to demonstrate [meaning of].

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

The Memory Allocation Role of the OS

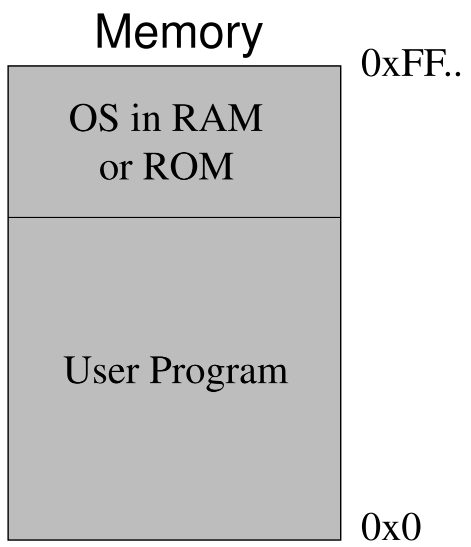
- Main RAM memory is an important resource in computer systems.
- Using memory efficiently is an important part of the OS function.
- Using memory safely (protection) is also an important part of the OS function.
- A program must be loaded to memory and given space for its data before it can execute.

A [simple] Single User Program: where does it fit in memory

Uniprogramming

- OS like KOMODO or original MSDOS undertake the following <u>steps</u> when a program is loaded into memory:

1) OS given a file name;
2) Loads it from disc;
3) Jumps to start of program;
4) Executes the program; may do I/O etc via OS;
5) Returns to OS when done.



A **Loader program** can produce relocatable code while it is loading a program binary into main memory.
Reloadable code may have a code base address of 0x0000 but be relocated to 0x2000; this is not the case [here]; as this example locates to 0x000.

Single Program Issues

A few additions to notes are required.

- Loads always to a fixed address;
  - (e.g. starts from 0).
- So, absolute addresses can exist in the binary image on file.
-
- QUESTIONS:
  - What if the program does a lot of slow I/O?
    - CPU could be doing something useful!
  - What if the memory is too small?

Where to find this Lecture 10 Multiprogramming And Fixed Partitions Tutorial one of the Cam Coder Tutorials for the COMP25111 course?

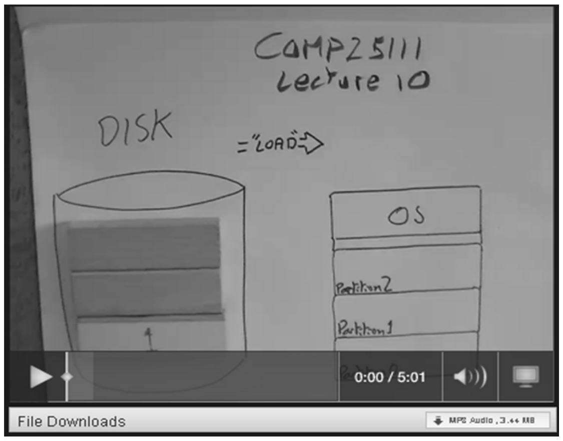
First Go to Blackboard 9; then select: COMP25111 Operating Systems

Then select: Week 7
This topic provides...
10: Memory management 1 (Introduction to basics) by RN;
11: Memory management 2 (Virtual Memory (1)) by RN;

Then select: Lecture 10 Information

Then select: Cam Coder Tutorials Lecture 10 Partitions Tutotial also see:

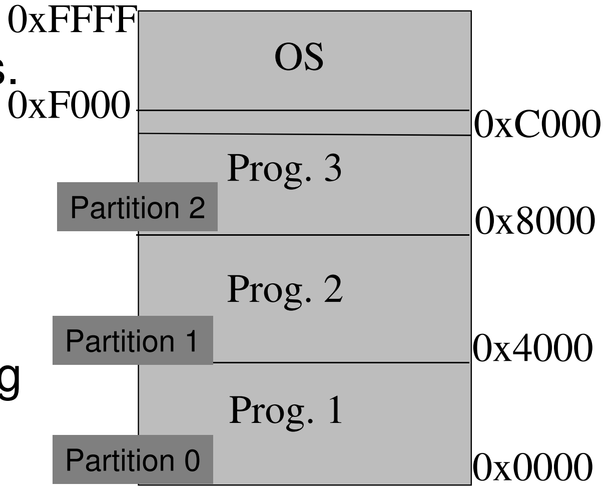
Then select: Cam Coder Lecture 10 Multiprogramming And Fixed Partitions Tutorial for a variation …



Multiprogramming – fixed partitions

1) Used in the past.
2) Load multiple programs.
3) Switch between them.
   - e.g. on I/O, or at regular intervals [the OS switches].
4) When one finished bring in a new one.

Multiprogrammed operating system: Partitions



Partitions == frames

Multiprogramming – fixed partitions

It is the ability of code to be relocated

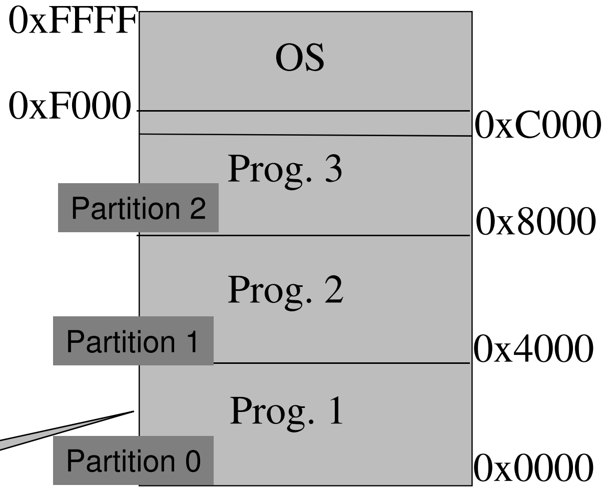
(i.e. position-independent code)

that enables a process to be swapped out of Partition 0 and later swapped back in to Partition 1.

Hence, it is the ability to relocate that enables swapping.

If partitions fixed size; this leads to: position-independent code

Multiprogrammed operating system: Partitions



Partitions == frames

Multiprogramming (fixed partitions) Issues

- Now code needs to be ‘relocated’;
  - i.e. absolute addresses depend on partition.
- Now programs could potentially write to each other’s memory;
  - protection needed.
- Partitions are (even) smaller than main memory.
- Provide illusion of *n* concurrent programs but limited to *n* (3 in this case)

Relocation; simplistic solution

- One way to solve the addresses problem is to have a loader[1] which ‘fixes up’ addresses
- i.e. an instruction of the form JSR 100 can be detected (its binary form) as it is being loaded and have the address modified.
- E.g. JSR (100+0) for partition 0

  JSR (100+4000) for partition 1

  JSR (100+8000) for partition 2

Reference: Reading: MOS (Ed 2) section Memory; subsection relating to figure 1-9; use of ‘base-limit [register] pair.

FootNote **1**: A **Loader program** can produce relocatable code while it is loading a program binary into main memory. Reloadable code may have a code base address of 0x0000 but be relocated to 0x4000 or 0x8000; this is the case [here]; as this example locates JSR in partition1 by adding 4000 and JSR in partition 2 by adding 8000, in partition 0 no change is necessary…

Relocation + Protection

‘fixes up’ addresses scheme

- Previous scheme does not solve any protection problem – programs can write to any memory [address].
- A simple (once used) solution is to introduce ‘base’ & ‘limit’ registers loaded before a program executes.

Concept used in MMUs; Dynamic relocation.

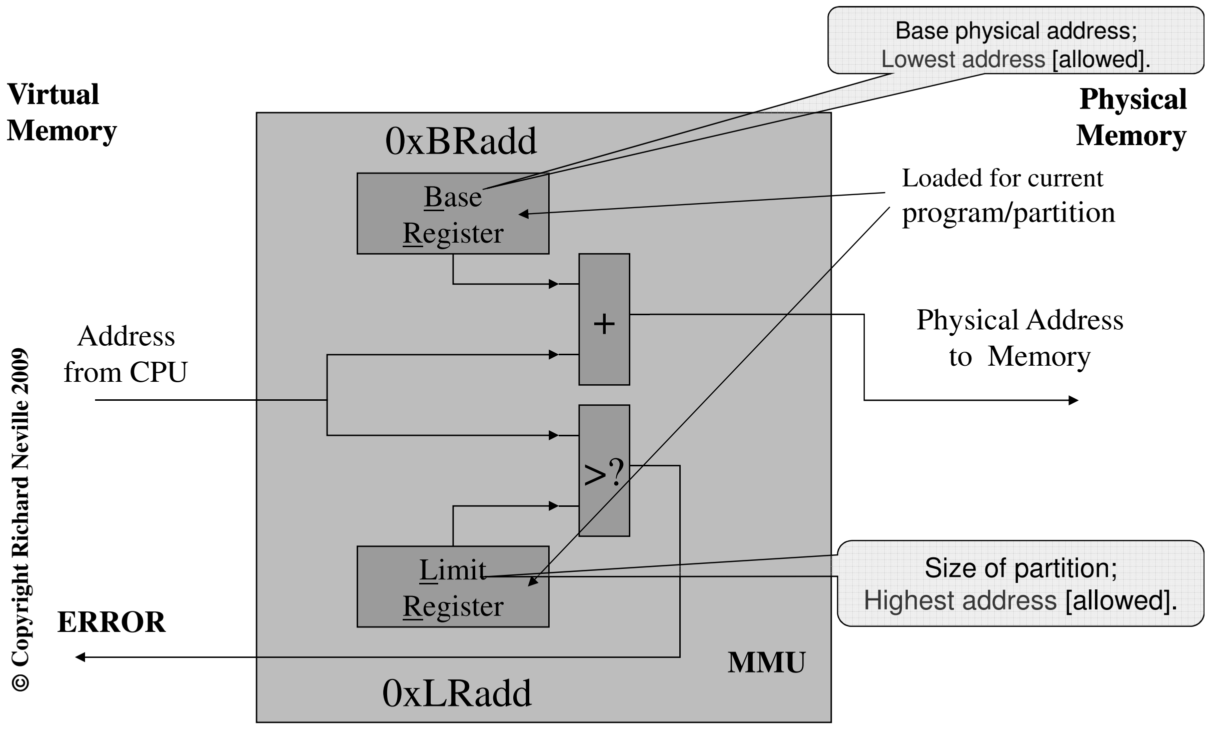
- Address added to the base but must be less than the limit.

Dynamic relocation – “*map each process’ address space onto different parts of physical memory.*”

Quotation

References:
1: Quote form: Modern Operating Systems, 3/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, ISBN 0-13-813459-6, ISBN 978-0-13-813459-4, Publisher: Prentice (person International Edition).

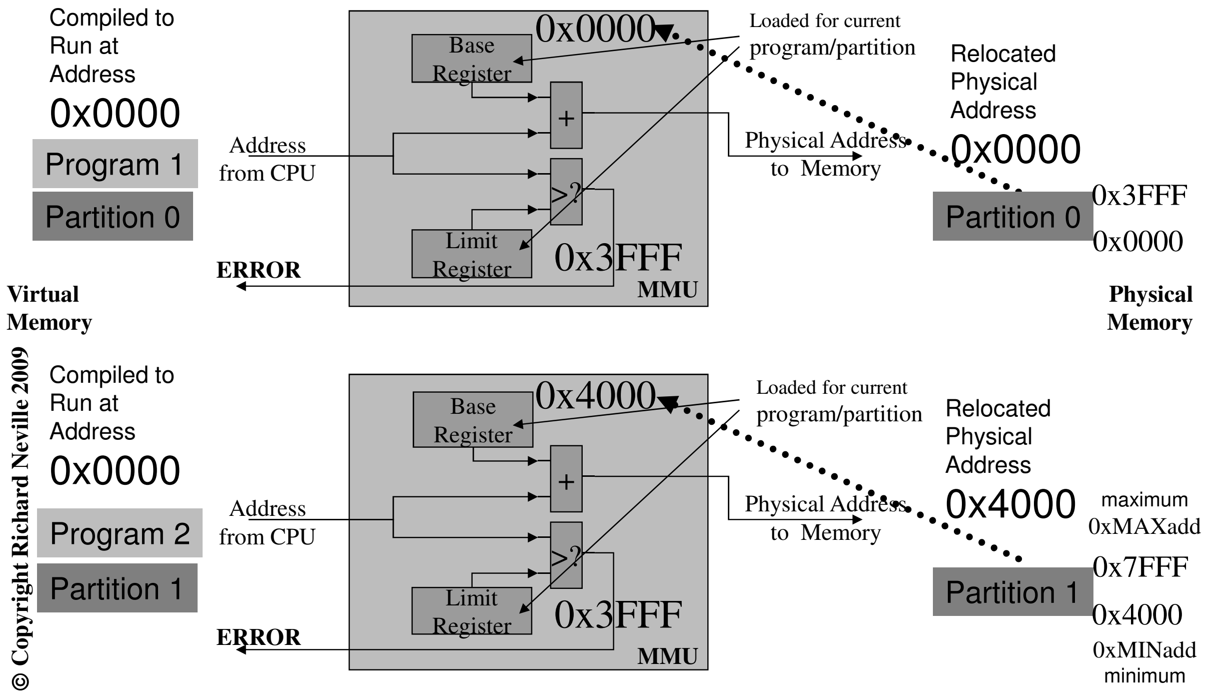
Relocation & Protection H/W (1)



S Middle E

Learning; comprehension; & introspection

Relocation & Protection H/W (2)



Learning; comprehension; & introspection

Quotations

1. Base register – "*When a process is run, the base register is loaded with the physical address where the program begins in memory…*"

2. Limit register – " … *the limit register is loaded with the length of the program..*"

3. Base register usage sequence – "*Every time a process references memory, either fetch an instruction or read or write a data word, the CPU hardware [MMU with Base and limits registers] automatically adds the base value to the address generated by the processor before sending the address out on the memory bus.…*"

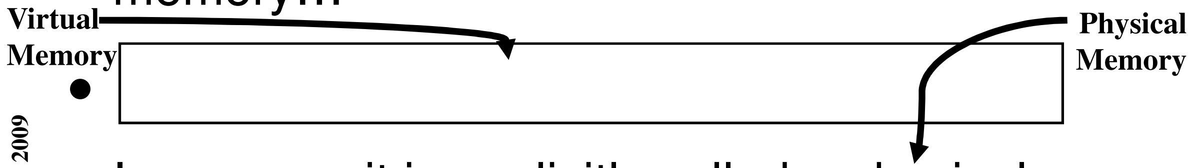
4. Limits register usage sequence – "*Simultaneously, it checks if the address offered is … greater than the value in the limits register; in which case a fault (or exception is thrown) is generated and access aborted.…*"

References:
1: Quote form: Modern Operating Systems, 3/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, ISBN 0-13-813459-6, ISBN 978-0-13-813459-4, Publisher: Prentice (person International Edition).

Virtual Addresses

A few additions to notes are required.

- Converting an address generated by a program[1] to the actual address used in memory…



-

- In memory it is explicitly called a physical address.

- This process is undertaken by the MMU (Memory management unit).

FootNote **1**: note the program's binary image is created by compiler - the program will be compiled to a specific 'base address' - this is the programs actual 'virtual address' - nominally this is a virtual base address of 0x0000 – it is NOT the actual address used in memory; its 'physical address.'

Power of Virtual Addresses

- The concept of a Virtual Address is a powerful one.

- We will see later how, by using more complex address translation, we can not only solve the relocation and protection problems but provide highly extended memory spaces.

Learning; comprehension; & introspection

Swapping (1)

- The fixed partition approach allows limited multiprogramming;
- QUESTION
  - But what if we want say 20 programs running at once?
- OPTIONS:

1) We can't keep dividing up the memory; or
2) Instead we could keep moving memory images (code + data) in and out to disc.

- **Swapping** allows the OS to moving memory images (code + data) in and out to disc.

Learning; comprehension; & introspection

Swapping (2) [cont.]

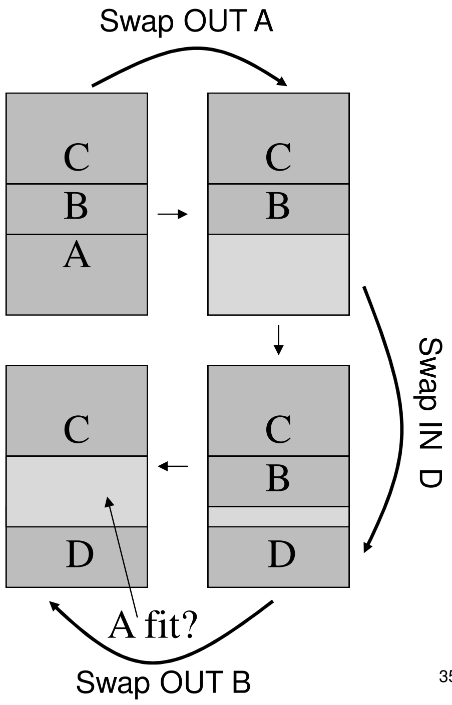
- We would still need more than one program in memory to ensure efficiency (i.e. exploit I/O delays).
- For flexibility, we should probably change to variable size partitions.
- Base & Limit approach makes this relatively easy.

Swapping (3) [cont.]

[With] variable size partitions…
Q: What happens?

- But can get fragmentation – may need to compact memory – messy.
- Real programs have dynamic memory allocation (new objects) – size unknown.

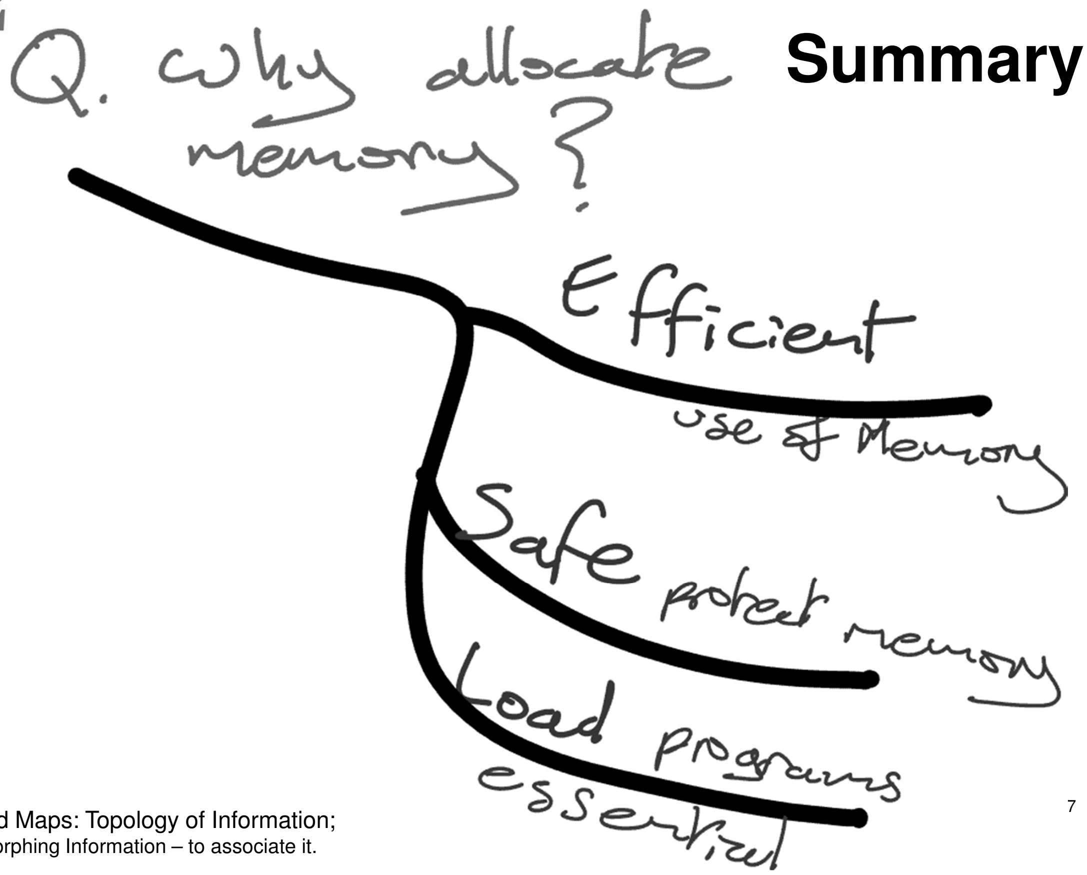
If required to:
'**Swap IN A**'
Back into memory;
QUESTION:
Will A fit?



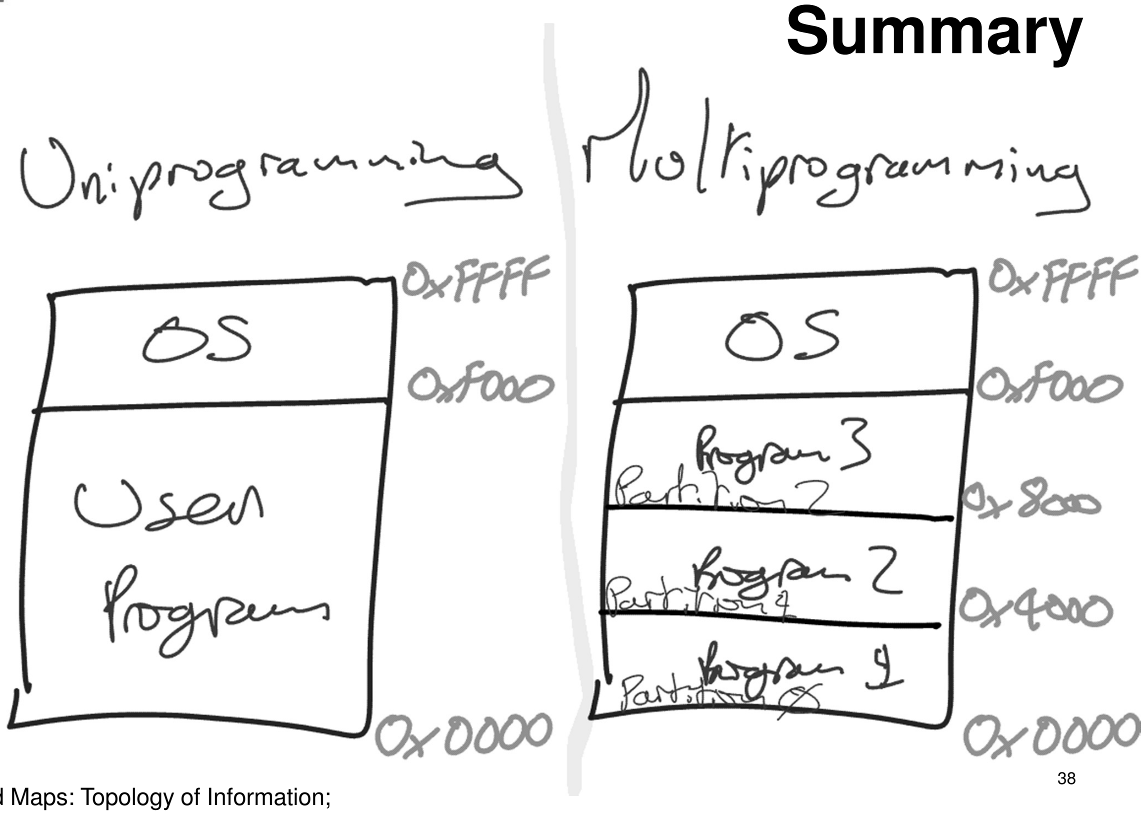
Summary

- Simple single program allocation is fairly easy but inflexible and wasteful;
- Multiple program allocation needs both relocation and protection – hardware supported address translation can do both.
- True multiprogramming needs more dynamic mechanisms and these introduce difficult 'housekeeping' issues.
- We need something better.

Summary

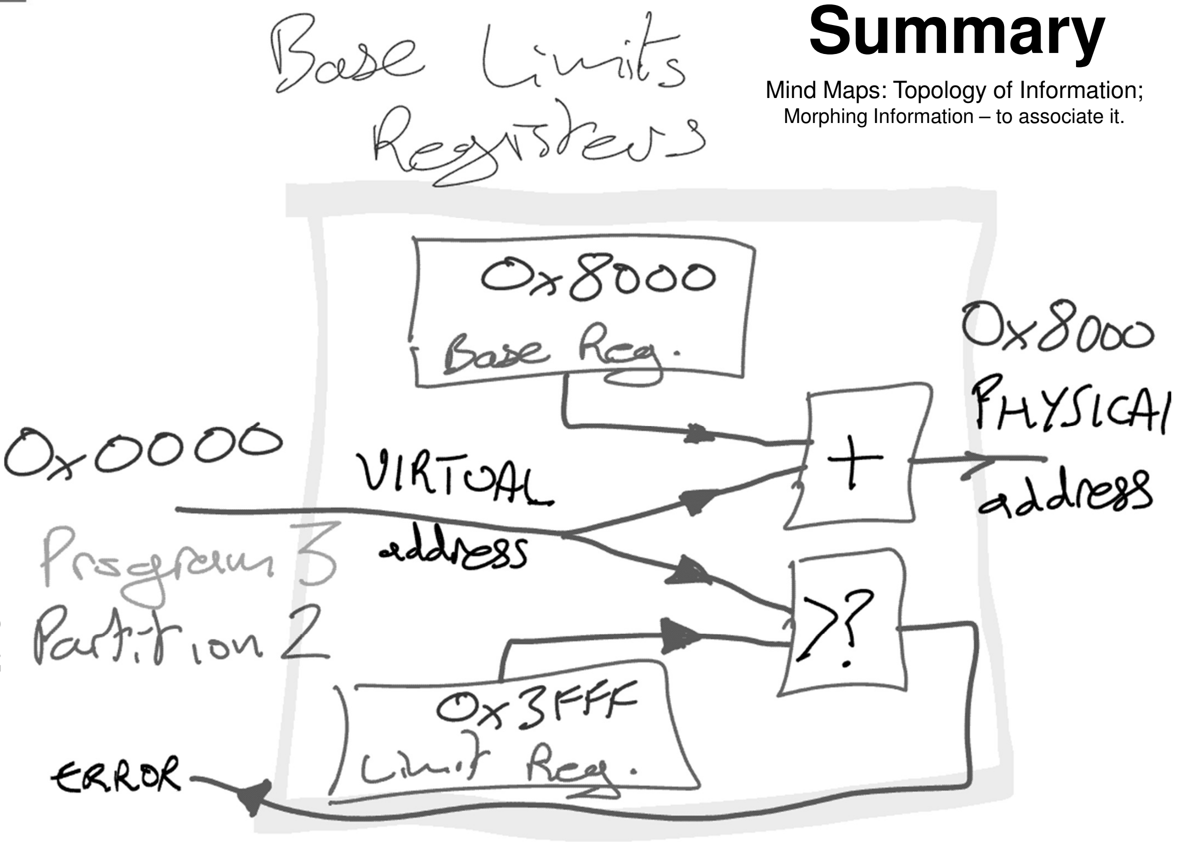


Summary



Summary

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



END

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

THIS week's

Have a go answering all Q3 questions below

Short Exam Questions **Q3**

1. Question
   Name, the three components that make up a <u>memory hierarchy</u> as used in computer systems:
   ANSWER(S):

Answer(s):

2. Question
   Which of the three is the fastest? Order them in terms of speed [access time].
   ANSWER(S):

Answer(s):

3. Question
   Order them in terms of size.
   ANSWER(S):

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 week's

Short Exam Questions **Q3**

1. Question
   What is the difference between a '*partition*' and a '*program*.'

Answer(s):

2. Question
   What is a '*fixed partition*':

Answer(s):

3. Question
   Differentiate between fragmentation and compaction:
   a) Fragmentation: xxx. B) Compaction: xxx.

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?
  - 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 stating point for this module [unit].
  - Please feel free to add to the glossary's throughout the unit…

Glossary

Using the on-line resources and any other resources compile a glossary of the terms below [MM: memory management] ; the glossary is full of potential exam questions of the form "define X" or "briefly explain X":

- Safety (in context of MM)→
- Protection (in context of MM)→
- Absolute addresses (in context of MM)→
- Multiprogramming (in context of MM)→
- Fixed partitions (in context of MM)→
- Relocate (in context of MM)→
- Concurrent programs (in context of MM)→

Glossary

- Base register (in context of MM)→
- Limit register (in context of MM)→
- MMU (in context of MM)→
- Physical address (in context of MM)→
- Virtual address (in context of MM)→
- Address translation (in context of MM)→
- Fragmentation (in context of MM)→
- Compact (in context of MM)→

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 multi programming, memory management, virtual memory, paging, , covers MMU, segmentation – in chapter 9 the operating system in: Chalk BS, Carter AT, Hind RW (2004) Computer Organisation and Architecture: An introduction 2nd Edition, Palgrave, ISBN 1-4039-0164-3 .
  - Section on memory protection, uniprogramming, multiprogramming, memory management, swapping, partitioning, paging, virtual memory, segmentation – in chapter 6 the operating system support in: Computer Organization and Architecture, Fifth Edition by William Stallings.
  - Section on virtual memory, paging, covers MMU, segmentation – in chapter 6 the operating system machine level in: Structured Computer Organization, 5/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, ISBN-10: 0131485210, ISBN-13: 9780131485211, Publisher: Prentice.

- **Web resources:**
  - Extensive (and may be best): Technical Resource and Course Web Site for Computer Organization and Architecture, Sixth Edition by William Stallings; available [on-line] @ http://williamstallings.com/COA6e.html
  - Software and supplementary material for Andrew S. Tanenbaum's Books available [on-line] @ http://www.cs.vu.nl/~ast/books/book_software.html
    - Operating Systems: Design and Implementation, 2nd ed. available [on-line] Zip file containing figures in PostScript PDF .eps .jpg @ http://www.cs.vu.nl/~ast/books/book_software.html
    - Modern Operating Systems, 2nd ed. available [on-line] Zip file containing figures in PostScript PDF .eps .jpg ; PowerPoint sheets for a course using this book (48 MB) ; Simulators @ http://www.cs.vu.nl/~ast/books/book_software.html
  - 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

Questions

Introduction to Questions:

The set of questions are based on lecture 11.

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

- **Questions (To Be Answered based on Lecture 11)**

1. **Question**

   What steps are undertaken by an OS when it loads a uniprogram into memory?

Answer:

Questions

1. Answer to Question 1:

- OS undertake the following steps

Answer(s):

1.
2.
3.
4.
5.

Questions

2. Question

What is the difference between:

a) absolute addresses;

b) physical Address; &

c) virtual address?

ANSWER(s):

2. Answer to Question 2:

- The difference are [in detail]:

Answer(s):

3. **Question**
   What are base and limit registers? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

4. **Question**
   In the case of a simple computer system with real memory. Explain why a multiprogrammed operating system would need to be able to relocate code? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

5. **Question**
   Explain how a "loader program" can produce reloadable code while it is loading a program binary into main memory? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

6. **Question**
   With the aid of a diagram, describe the structure of a base and limit system and explain how it achieves relocation? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

7. **Question**
   What other benefits does a base plus limit hardware provide? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

8. Question
What is ‘swapping’ in the context of operating system memory management and why is it useful? [In the context of relocation of programs in memory.]

Answer

Answer(s):

Questions

9. Question
Explain briefly what is meant by the term multiprogramming?

Answer

Answer(s):

Questions

10. Question

With the aid of a diagram, describe the structure of a simple base and limit system and explain how it achieves relocation.

NOTE: answer must contain two components: a fully labelled diagram and a concise explicit description of a base and limits system and how it achieves relocation.

Answer(s):

Revision Exercises

- Scan read Lecture 11’s Questions.
  - Answer Lecture 11’s Questions
    - Particularly those questions you had difficulties with when you first tried them.

Background Reading

- Remember if you do not have MOS or MOS 2nd edition the key issues, highlighted in **GREEN**, are the concepts to look for in any book:
  - Reading: MOS 4.1 & 4.2;
  - MOS (Ed 2) section Relocation and protection: chapter 4; section 4.1 basic memory management; subsection 4.1.5 relocation and protection;
  - MOS (Ed 2) section Memory; subsection relating to figure 1-9; use of 'base-limit [register] pair;
  - MOS (Ed 2) Chapter 1 introduction; section 1.4 computer hardware review subsection 1.4.2 memory – virtual address – memory management unit;
  - MOS (Ed 2) Chapter 4 memory management; section 4.3 virtual memory; subsection4.3.1 paging – memory management [unit];
  - MOS (Ed 2) Chapter 4 memory management; section 4.6 design issues for paging systems; subsection 4.6.3 page size – fragmentation.
  - MOS (Ed 2) Chapter 4 memory management; section 4.2 swapping – [memory or internal] compaction.
  - Good introduction, re. *Lecture* 06 *Memory* Management - Presentation Transcript **...** Need hardware support for address maps (e.g., ***base and limit*** **registers**), available [on-line] @ http://www.slideshare.net/RajNOX/lecture-06-memory-management, (Last date accessed 5-07-2010).
  - Lecture 9: Memory management, available [on-line] @ http://www.docstoc.com/docs/21097488/Operating-Systems-Lecture-9-Memory-Management-Demand-paging (Last date accessed 5-07-2010).

COMP25111 Lab Exercise 2: Scheduling

Duration: 2 sessions

Aims

To give you some insights into process scheduling and into semaphores, and some practice in the use of threads in Java.

Learning outcomes

On successful completion of this exercise, a student will:
- Have modified an illustrative non-preemptive scheduler
- Have implemented semaphores to allow threads to rendezvous at a barrier

Summary

Using Java, and in particular using Java threads to simulate processes,

1. implement different scheduling algorithms by writing `RoundRobinScheduler.java` and `PriorityScheduler.java` (starting from `RandomScheduler.java`) for use by the `TestScheduler.java` program, and
2. complete the implementation of semaphores in `Semaphore.java` for use by the `TestBarrier.java` program.

There are very brief descriptions of scheduling and semaphores below. For more information, consult the course textbook(s).

The unextended deadline is the end of your (second) scheduled lab.
If you attend the lab you will, if you need it, automatically get an extension to midnight the same day - you must use submit to prove you finished in time.
You must get your work marked in this lab or in your next scheduled lab.
You can also get an extension for good reason e.g. medical problems.

Description

For this lab exercise you should do all your work in your `COMP25111/ex2` directory.

Copy the starting (Java) files into your `COMP25111/ex2` directory from
`/opt/info/courses/COMP25111/ex2`
These files are (for part 1): `TestScheduler.java`, `Scheduler.java`, `RandomScheduler.java`, `SimProc.java`
(for part 2): `TestBarrier.java`, `Semaphore.java`, `SemScheduler.java`, `SemProc.java`

Introduction

The scheduler used in an operating system is not an easy thing on which to experiment in a modern networked PC. Instead this exercise will modify a Java class which acts like a scheduler to "processes" which are simulated by Java threads.

Unlike real processes, these threads do not get timesliced - but instead relinquish the processor at points of their own choosing by calling a method on the scheduler. Thus we are doing non-preemptive[1] scheduling.

The threads are also responsible for introducing themselves to the scheduler when they start, and removing themselves when they finish. In a real system, the Operating System ensures that these things are done without the process calling the scheduler - but this artificiality does not really change how the scheduler itself operates.

Part 1: Alternative scheduling strategies

You are provided with an example scheduler in `RandomScheduler.java` (a sub-class of `Scheduler.java`) which chooses randomly which "process" to run next from those that are available.

The program `TestScheduler.java` currently uses `RandomScheduler` to run a number of simulated processes (described below). Start by running it as it stands, to get some understanding of the set-up.

Read the code you are given for hints to write two different schedulers (variants of `RandomScheduler`):

`RoundRobinScheduler.java`

which uses round-robin[2] scheduling instead of random.

Note: `RoundRobinScheduler` needs to get a different process from the list each time around the loop. If you simply increment an index, you might skip a process when the previous process finishes. (Why? If you don't know, try it and see what happens.) Don't just check the length of the list. (Why? You might get away with it in this simple simulation, but what happens in a more general situation?) Instead, you need to find a well-engineered way to do whatever is necessary each time a process enters or leaves the list.

`PriorityScheduler.java`

which does scheduling among processes using "priorities", where priority is a property of the process which can change as it runs (see `getCount` in the description of the Simulated Processes below). If there are several processes with the same, highest priority, it doesn't matter which is run.

[1] **Non-preemptive scheduling**: the process keeps the CPU until the process terminates or it switches to 'blocked' state (simple to implement).
Preemptive scheduling: the process can run (continuously) for a maximum of some fixed time. If it is still running at the end of this time, it is interrupted and the scheduler will pick another process to run (needs timer). Reference: COMP20051, An Introduction to Process and Thread Scheduling: Lecture 5, previous lecture notes for COMP25111.

[2] In the general case, if there were many processes in the ready state, the CPU can be given to each process, in turn, in a round-robin fashion. This is the basic principle of **round-robin scheduling**! Reference: COMP20051, An Introduction to Process and Thread Scheduling: Lecture 5, previous lecture notes for COMP25111.

Modify `TestScheduler.java` to run with your new schedulers to demonstrate their working.

Details of the Simulated Processes

The simulated processes are instances of `SimProc.java`, and should not be changed. Each process has a name, and computes a number of values of the `collatz` function.

(*It doesn't matter what the `collatz` function computes - but you can see from the code that it is quite simple. The interesting fact about it is that the loop terminates for all known arguments if large enough intermediate values can be handled on the way - and there is no known proof of this termination in general. This is known in the literature as the "3x+1" problem, but also as the Collatz problem, the Syracuse problem, Kakutuni's problem, Hasse's algorithm, and Ulam's problem!*)

The process continues to call the `collatz` function, relinquishing the processor after each call, until it has computed 4 non-negative results for random inputs. Thus the number of time-slices required by each process is not easily determined.

When using the priority scheduler on these processes, the priority should be the result of calling the `getCount()` method. This returns an integer showing how many non-negative results the process still needs to compute.

Part 2: Semaphores and Barriers

You are given a program which runs several "processes" and uses semaphores[3] to provide "barrier[4] synchronisation" - i.e. all the processes should wait for the last to arrive each time around a loop before proceeding. However, in the code you are given, the method bodies for the `P()`[5] and `V()` semaphore operations are empty, so each process, once started, will run to completion before the others run.

To get some understanding of the set-up, start by running `TestBarrier.java` (which uses "processes" provided in `SemProc.java` and a scheduler in `SemScheduler.java`) without modifying the semaphore code in `Semaphore.java`. Note you also need the `Scheduler.java`, and the `SemProc.java` classes.

Barriers

A barrier is a mechanism whereby all the threads that reach it stop and wait for the last to arrive. They then all proceed and the barrier is re-initialised to hold up the next batch of threads to reach it (typically the same threads, each executing a loop containing the barrier).

[3] **Semaphores**: Ensure that when one process/thread is executing in its critical section no other process is allowed to execute in its critical section. If several requests occur at once, one process only should proceed; other processes must wait outside critical section. Reference: COMP20051, Process/Thread Synchronisation, Lecture 7, previous lecture notes for COMP25111.
[4] When a **barrier** is reached, all threads are forced to wait for the last thread to arrive. Reference: C. Ball & M. Bull, Barrier Synchronisation in Java, available on-line @ http://www.ukhec.ac.uk/publications/reports/synch_java.pdf , (Last date accessed 6/10/2011).
[5] Semaphores are equipped with two operations, historically denoted as **V** (also known as signal() [from Dutch 'proberen' ("test")]) and **P** (or wait() [from Dutch 'verhogen' ("increment"); "up" in MOS]). Operation **V** increments the semaphore S, and operation **P** decrements it. Reference: Semantics and Implementation, in Semaphore (programming), available on-line @ http://en.wikipedia.org/wiki/Semaphore_%28programming%29 , (Last date accessed 6/10/2011).

This way, one thread does not get too far ahead of any other - regardless of the underlying scheduling mechanism.

You can see the barrier code, which uses 2 semaphores, in the method `reachBarrier()`. **You do not need to understand how the barrier is implemented from semaphores to do this exercise.**

Semaphores

Although Java uses a rather different mechanism for synchronization between threads, it is still possible to implement semaphores in this context. Below is some pseudocode for the operations on a general semaphore called "sem", which has a `value` and a queue of threads which are waiting. The trick it uses is that negative values of the semaphore are used to count how many threads are queuing - this simplifies the logic and is quite standard.

```
  P():
      decrement sem.value ;
      if (sem.value >= 0) return ;
      // otherwise, this thread needs to stop running and join the queue,
      so move current thread from the scheduler's ready queue to sem.queue
      ;

  V():
      increment sem.value ;
      if (sem.value > 0) return ;
      // otherwise, there is a thread to awaken from the queue, so
      move the front thread from sem.queue to the scheduler's ready queue.
```

Modify `Semaphore.java` to implement the methods `P()` and `V()` as above. **You must not modify any other code in this class, or any code in the other associated classes.** In translating this into Java, be careful to be thread-safe, i.e. to ensure that threads cannot accidently interfere with the operation of P() and V() by other threads. Re-run `TestBarrier` to see the results.

Assessment

You must use `labprint` and `submit` as normal. They will look for:
`RoundRobinScheduler.java`, `PriorityScheduler.java` and `Semaphore.java`

The marks are awarded as follows:
Part 1:
4 - Correct Round Robin scheduler
4 - Correct Priority scheduler
Part 2:
8 - Correct operation of semaphores
Both parts:
4 - Sensible style of code
Total 20

Remember to get your solution marked as soon as possible after it has been submitted.

References:

[1] R. B. Muhammad, Operating Sytems, re. Round Robin Scheduling, Priority Scheduling, Semaphores, available on line @ http://www.personal.kent.edu/~rmuhamma/OpSystems/os.html, last accessed 6/10/2011.

ex2 Hint 1

The first stage of the ex2 is to get the basic [sub-set of] classes compiled and then execute and produce an output.

Prior to this it is a good idea to visualise [abstract] the classes that will be working together. One of the ways to do this is to visualise them in an UML class diagram; you will be learning about these later on in your degree – so don't worry – but if you want to find out more about UML class diagrams undertake some research; figure 1: [basic] UML class diagram of [sub-set of] classes.

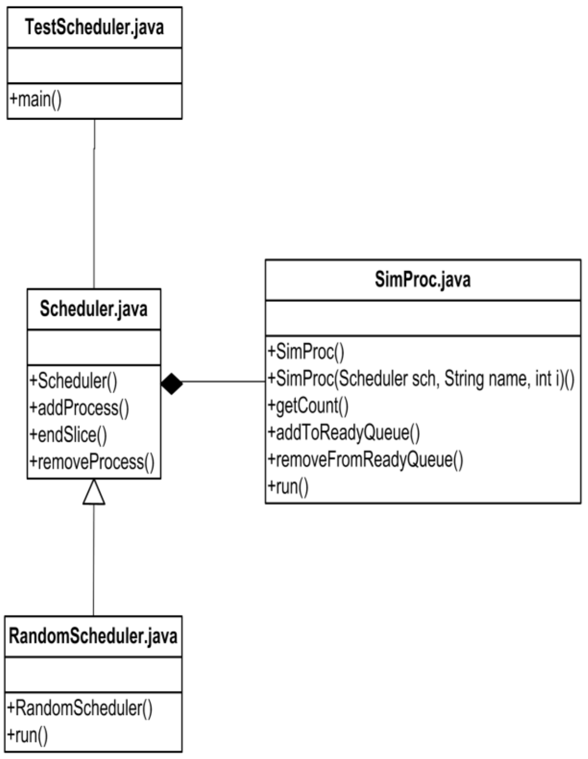


Fig. 1. UML class diagram of [sub-set of] classes.

It is also an idea to visualise the sequence of instructions; or the method call sequence; sometimes viewed as "the interactions between objects in the sequential order that those interactions occur [1];" this can be done utilising the UML sequence diagram. In the case of the sub-set of classes: `TestScheduler.java`, `Scheduler.java`, `RandomScheduler.java`, `SimProc.java` the basic outline of the sequence diagram is depicted in Figure 2; **note** this sequence diagram does NOT stick to the normal UML convention for labelling message arrows – it labels the arrows with the object method call – as specified in the java code.

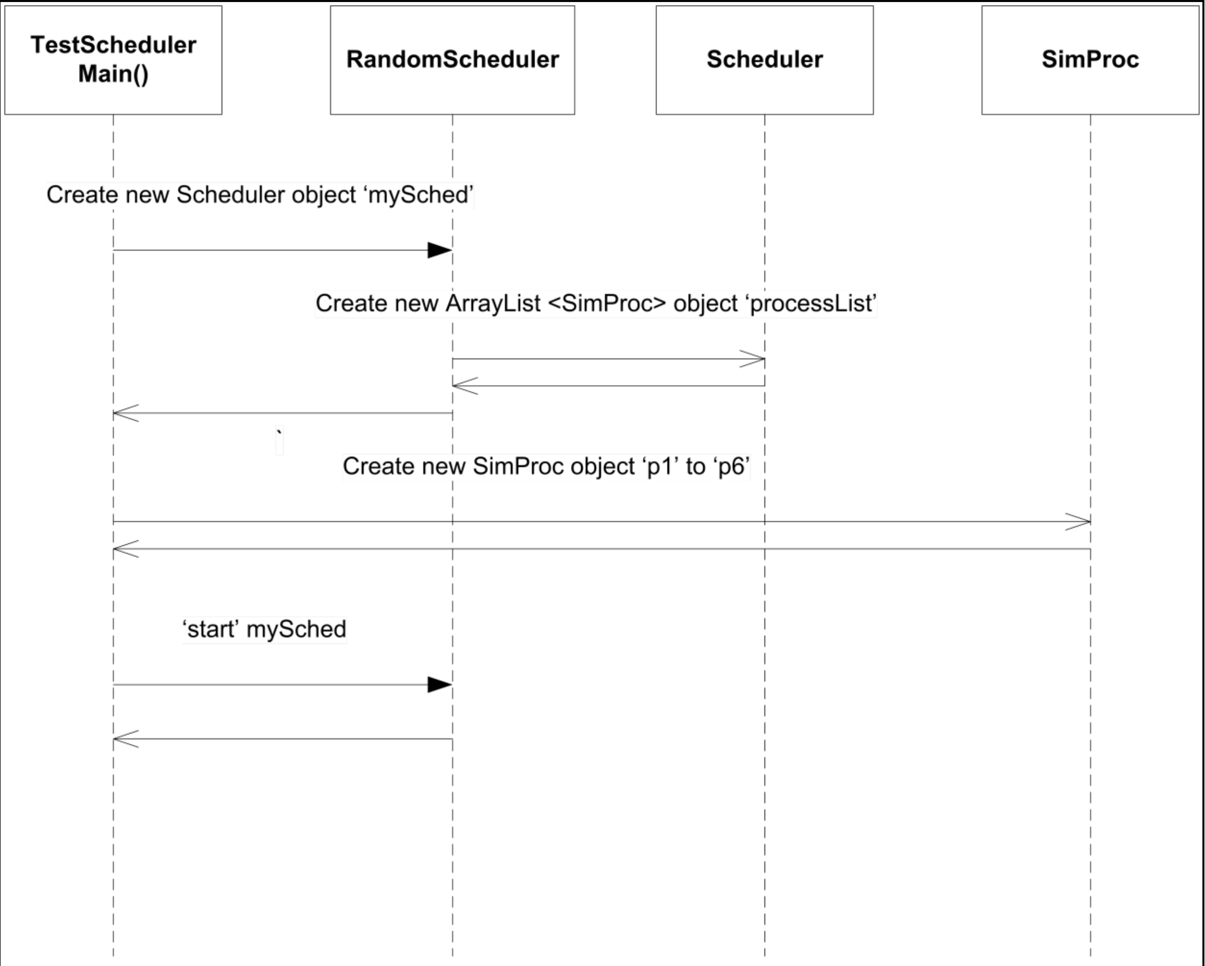

TestScheduler Main()
RandomScheduler
Scheduler
SimProc
Create new Scheduler object 'mySched'
Create new ArrayList <SimProc> object 'processList'
Create new SimProc object 'p1' to 'p6'
'start' mySched

Fig. 2. Basic UML sequence diagram of [sub-set of] classes.

The next step is the compilation and execution of: `TestScheduler.java`, `Scheduler.java`, `RandomScheduler.java`, `SimProc.java`. One you have compiled the four files run them and the resultant output you should acquire is depicted in figure 3 [Remember as it is "Random" order the output threads will be in a random order (of the Threads) – hence you will not obtain exactly the same output in figure 3 – but if you inspect the sequence you will indeed see they are in random order].

```
This is Francis - Collatz of 1457712396 is 165
This is Alan - Collatz of 345300012 is 145
This is Francis - Collatz of 2054253694 is -1
This is Brian - Collatz of 1902024452 is 243
This is Edward - Collatz of 1741402412 is 137
This is Brian - Collatz of 2044483757 is 244
This is Francis - Collatz of 424086582 is -1
This is Charles - Collatz of 244589539 is 85
This is Alan - Collatz of 1967502833 is 236
This is Brian - Collatz of 784079238 is 115
This is Francis - Collatz of 115955971 is 239
This is Edward - Collatz of 1800016525 is 220
This is Charles - Collatz of 11682451 is 184
This is Alan - Collatz of 1064384834 is -1
This is Dave - Collatz of 832032679 is -1
This is Francis - Collatz of 1033704605 is -1
This is Edward - Collatz of 1125642362 is 219
This is Brian - Collatz of 902174553 is -1
This is Alan - Collatz of 2222371 is 117
This is Francis - Collatz of 351718320 is 150
This is Edward - Collatz of 1041352256 is 255
This is Charles - Collatz of 293308302 is 204
This is Brian - Collatz of 1230156273 is -1
This is Alan - Collatz of 2052532180 is 176
This is Francis - Collatz of 1458526209 is 118
This is Charles - Collatz of 382063338 is 70
This is Brian - Collatz of 1613859455 is -1
This is Dave - Collatz of 184521508 is 250
This is Brian - Collatz of 2021315824 is 181
This is Dave - Collatz of 1533157800 is 98
This is Dave - Collatz of 2054627414 is -1
This is Dave - Collatz of 1967699861 is 249
This is Dave - Collatz of 1257881609 is -1
This is Dave - Collatz of 2090545418 is -1
This is Dave - Collatz of 1560187619 is 159
```

Figure 3, Output of `TestScheduler.java`, `Scheduler.java`, `RandomScheduler.java`, `SimProc.java files…`

Reading "the Code"

Finally, as will all exercises, we should read through the code. Hence, you should start with `TestScheduler.java` and work out the basic algorithmics it is implementing. Then take a look at `Scheduler.java`, `RandomScheduler.java`, and `SimProc.java`.

One of the questions you could ask yourself is: "Can I break the code up into functional blocks?" Where each block performs (or addresses) separate steps (or issues). If we look at the `RandomScheduler.java` code we should be able to do this.
NOTE: If you are seriously testing your comprehension of the code DO NOT read the next page till you have (yourself) worked out what `RandomScheduler.java` is doing algorithmically.

May be the best advice for sequentially separating concerns in a sequential program is to inspect the code for set of lines that are sequentially operating on the same object [or variable].

First parse of `RandomScheduler.java` separates the `run()` into three separate blocks, reference figure 4.

```
public class RandomScheduler extends Scheduler {

private Random randGen ;

    public RandomScheduler() {
        randGen = new Random() ;
    }

    public synchronized void run() {
        while (true) {
            int noProcs = processList.size() ;
            if (noProcs == 0) System.exit(0) ;
            int next = Math.abs(randGen.nextInt()) % noProcs ;
            SimProc nextRunner = processList.get(next) ;
            synchronized(nextRunner) { nextRunner.notify() ;}
            try { wait() ;}
            catch (Exception e) {
                System.out.println("Unexpected interrupt in run " + e) ;
            }
        }
    }
}
//*****************
public class RandomScheduler extends Scheduler {

private Random randGen ;

    public RandomScheduler() {
        randGen = new Random() ;
    }

    public synchronized void run() {
        while (true) {

        //#BLOCK1:
            int noProcs = processList.size() ;
            if (noProcs == 0) System.exit(0) ;

        //#BLOCK2:
            int next = Math.abs(randGen.nextInt()) % noProcs ;
            SimProc nextRunner = processList.get(next) ;
            synchronized(nextRunner) { nextRunner.notify() ;}

        //#BLOCK3:
            try { wait() ;}
            catch (Exception e) {
                System.out.println("Unexpected interrupt in run " + e) ;
            }
        }
    }
}
```

Figure 4, Segmentation of the `RandomScheduler.java` class into 3 BLOCKs…

#BLOCK1: the first block retrieves the `noProcs` variable, which retrieves the `size` of threads queue (or `processList`). Next, the `if` condition, in the next instruction, checks `if` all the threads (`processes`) have terminated. Note: the basic sequence is that threads run until they have finished undertaking the task they were designated. When they end they are removed from the queue and the `size` of threads queue (or `processList`) is decremented. When all threads have terminated the `size` == 0, hence `noProcs == 0`, then `(noProcs == 0)` so the `run()` is exited; utilising the `System.exit(0)` method.

#BLOCK2: Delineated by the use of variables & objects: `next` & `nextRunner`. The three lines basically:
(1) get a random number `next`;
(2) get the `next` thread from the queue, using `next` as the index to the thread in the queue to actually retrieve a random item from the queue [next]. Given that `next` is a random number;
(3) Finally, `notify` the next thread in the queue (or the `nextRunner`); note a lock is placed on this object, using the `synchronized(nextRunner)` statement. Actually, `notify()` wakes up a single thread which is waiting on the object's lock (object that was locked was `nextRunner`).
The basic sequence of `notify()` [in block 2] and then `wait()` [in block 3]. First the running thread calls the `notify()` method of the `nextRunner` Java object. This "wakes up" `nextRunner` thread of the threads waiting on that object. The `wait()` method will be discussed in BLOCK 3.

#BLOCK3: Puts the current thread (or `process`) into a `wait()` state, to enable other threads to run. Hence, BLOCK 3 calls the `wait()` method of any Java object, which suspends the current thread. The thread [queue] is said to be "waiting on" the given object. This is scoped {} in a try & catch block to deal with an unexpected interrupt.

NOTE [on Thread States] quote:
"Tasks are executed in threads. Threads can be in one of five states: New, Ready, Running, Blocked, or Finished.

When a thread is newly created, it enters the ***New** state*. A ready thread is started by calling its `start()` method, it [then] enters the ***Ready** state*. A ready thread is runnable but may not be running yet. The operating system has to allocate CPU time to it.

When a ready thread begins executing, it enters the *Running state*. A running thread may enter the ***Ready*** state if [the] CPU time [it was allocated] expires [or finishes] or its `yield()` method is called.

A thread can enter the ***Blocked** state* (i.e. become inactive) for several reasons. It may have invoked the `join()`, `sleep()`, `wait()`, or `lock()` method, or some other thread may have invoked these methods [on the running thread]. It may be waiting for an I/O operation to finish. A blocked thread may be reactivated when the action inactivating it is reversed. For example, if a thread has been put to sleep and the sleep time has expired, the thread is reactivated and enters the ***Ready*** state. Finally, a thread is finished if it completes execution of its `run()` method," [3].

Hence, in `RandomScheduler.java` class when instruction `try { wait() ;}` is executed the thread "enter the ***Blocked** state* (i.e. become inactive)," [3].

IMPORTANT: hence for the other two schedulers there is probably the same number of blocks. It is BLOCK 2 in both `RoundRobinScheduler.java` and `PriorityScheduler.java` that you have to implement.

Best advice is copy `RandomScheduler.java` to `RoundRobinScheduler.java` and `PriorityScheduler.java`, then modify them to perform sequential round robin scheduling and priority based scheduling.

Appendix

Definitions of terminology utilised in ex2.

Pre-emptive Algorithms
"Pre-emptive algorithms are the scheduling algorithms that take a process from the queue and assign the algorithm a set time slot, which it is allowed to run for. If the process does not finish running by then it is put back into the queue at the end and another process is selected to be run. However if during that time it does finish then the process is removed from the queue. The way it determines the algorithm to run next is via the scheduling algorithm that is in place. The animation system animates two types of pre-emptive algorithms, Round Robin Scheduler and Priority Scheduler [2]."

Round Robin
"This scheduling algorithm takes a process from the head of the queue. In other words, it takes the first process that entered the queue. It then will run the process for a specified period of time. This timeslot can be specified by the user when the applet is run. If the process is not finished after its allocated time then the process is put back in the queue at the end and the scheduler takes the next one at the head of the queue and runs it. It continues to do this till the processes have finished running [2]."

Priority Scheduler
"This algorithm has a further property defined with the processes. This property is the priority of the algorithm. Each process can have a priority defined with 1 being the highest. The algorithm will look through the processes in the queue and find the one with the highest priority and run that first. However when the process has been ran for the allocated timeslot, the algorithm deducts the priority by one and inserts it back in the queue. This gives the other algorithms an opportunity to run too [2]."

Reference

[1] UML basics: The sequence diagram, available on line @ http://www.ibm.com/developerworks/rational/library/3101.html, last accessed 6/10/2011.
[2] R. Begum, (May, 2010) Algorithm Animation, Final year Project, The University of Manchester School of Computer Science.
[3] Y. D. Liang, Introduction to Java Programming, ISBN 10: 0132130807 / 0-13-213080-7, ISBN 13: 9780132130806, Publisher: Prentice Hall, Publication Date: 2010, Binding: Softcover, also [some information] is available on-lie @ http://cs.armstrong.edu/liang/intro6e/ , http://cs.armstrong.edu/liang/intro7e/, http://books.google.com/books/about/Introduction_to_JAVA_programming.html?id=fI6dl1flmk8C , and http://www.filestube.com/i/introduction+to+java+programming+liang , last accesses 13-10-2011.