

CSc 133: Networking and Systems Issues

Keith Cheverst
C42
InfoLab21

csc 133

Overview...

- Only 5 lectures, but aim to cover 2 topics:
 - Topic 1: Introduction to OS fundamentals
 - lectures 1 and 2
 - Goal: Provide understanding of OS basics and to build a foundation for further studies in Part II.
 - Topic 2: Introduction to Networking Issues
 - Lectures 3, 4 and 5
 - Goal: Provide understanding of Networking basics and to build a foundation for further studies in Part II.

csc 133

Reading...

- We will use two different books for this course.
 - “*Fundamentals of Operating Systems*”, 5th edition, A. M. Lister and R. D. Eager, Macmillan Computer Science Series, £21.99.
 - “*Computer Networks - A Systems Approach*”, 3rd edition”, Larry L. Peterson and Bruce S. Davie, Morgan Kaufman, £34.99.
- “Operating Systems Design and Implementation”, A. S. Tanenbaum, Prentice-Hall International, ISBN. 0-13-630195-9, about £36.99.
- <http://info.comp.lancs.ac.uk/year1/notes/csc133/>
- www.comp.lancs.ac.uk/computing/staff/kc/keiths_teaching.html

csc 133

Assessment...

- CSC 101 and CSC 103
 - Examination in June
 - Short answer type questions
 - Split into sections on a stream basis (12x, 13x etc.)
 - Exam carries approx 5% of overall 101/103 mark
 - Written Coursework
 - Approx 7% of overall 101/103 mark

csc 133

Topic 1 : Introduction to Operating Systems

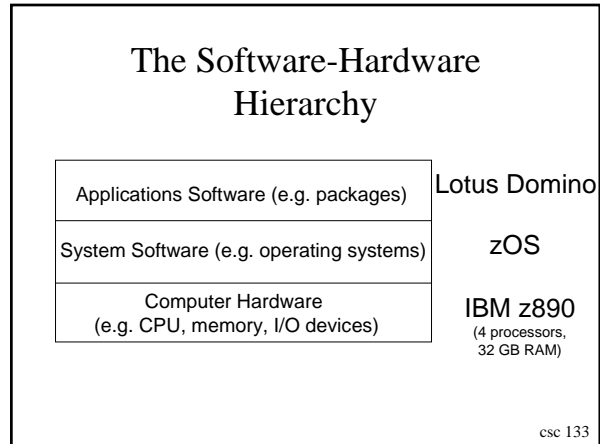
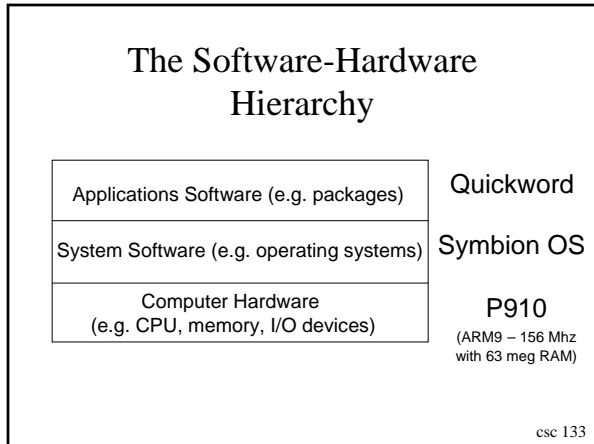
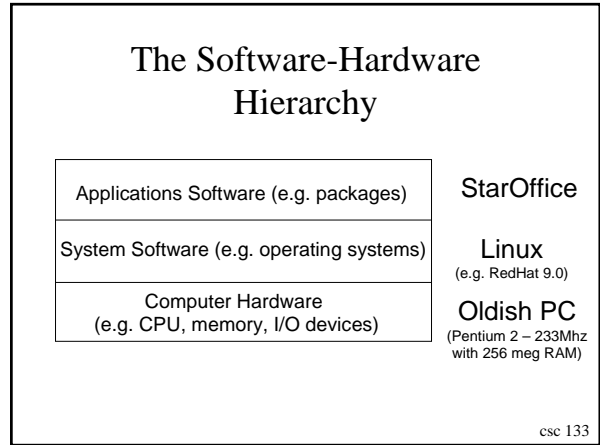
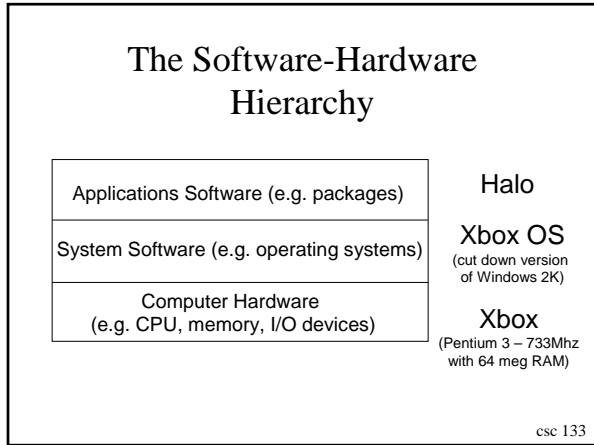
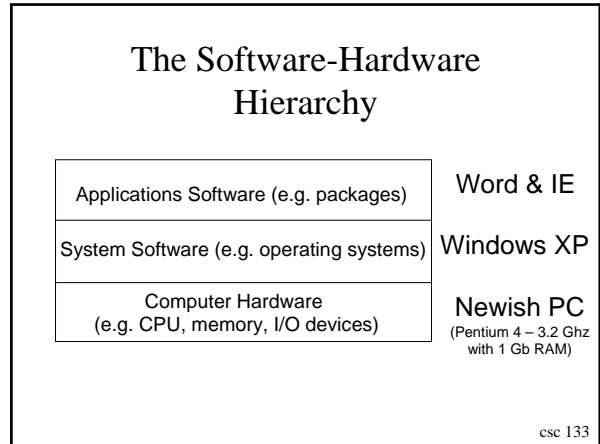
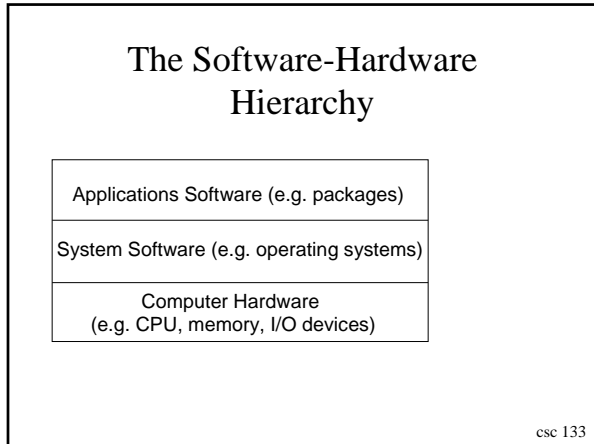
Lectures 1 and 2

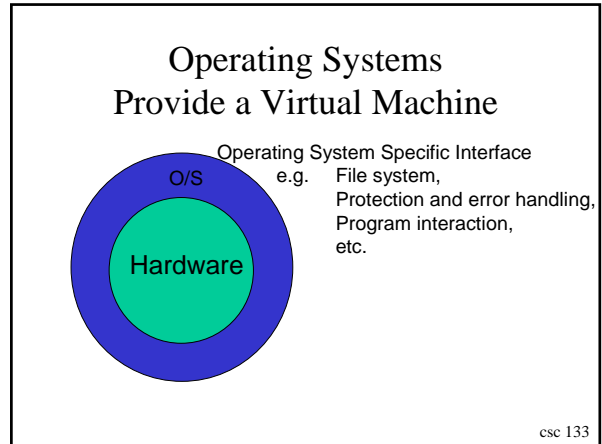
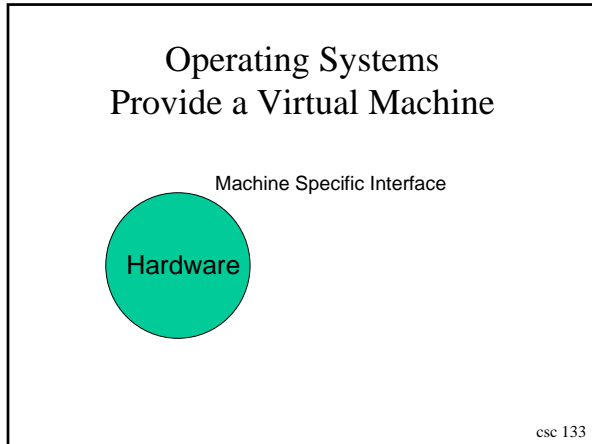
csc 133

Covered in this 1st topic...

- The OS Concept
 - (L&E 1-6)
- Loading and Executing Programs
 - Dispatcher (L&E 41-42)
- The Process Concept and Patterns of process execution
 - (L&E 17-19)
- Scheduling
 - (L&E 124-131)
- Fragmentation
 - (L&E 66-67)

csc 133





Operating Systems Also Manage Resources

- Share resources between multiple programs and multiple users.
 - Orderly and controlled allocation of resources
 - Protection
- Ensure use of resources is optimised.

How things change... Back in 1980 the ZX81 was heralded to have 'multimedia' capabilities because (unlike the model before) it could execute a program AND display a picture at the same time...

csc 133

Loading and executing programs...

- A simple Dispatcher ...
- Basic idea is to load the program from disk and start it executing.
- During its period of execution the program is called a *process*.
- When its finished start executing another one.

Secondary Storage

O/S

High
Low

csc 133

Loading and executing programs...

- A simple Dispatcher ...
- Basic idea is to load the program from disk and start it executing.
- During its period of execution the program is called a *process*.
- **When its finished start executing another one.**

Secondary Storage

O/S

High
Low

csc 133

Approach for Simple Dispatcher

- Sit in a loop waiting for a program to be ready to run.
- Load the specified program from disk into memory.
- When the program has finished go back to the loop and wait for another to be ready.

csc 133

So using this simple dispatcher, what would it look like if we had two programs to execute ???

csc 133

Example Programs

- Two programs to be run.

Program 1...

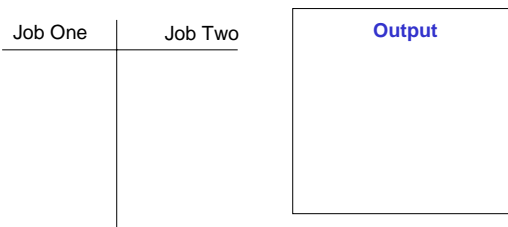
```
Begin
  putl i ne ("1");
  putl i ne ("2");
  putl i ne ("3");
End.
```

Program 2...

```
Begin
  putl i ne ("A");
  putl i ne ("B");
  putl i ne ("C");
End.
```

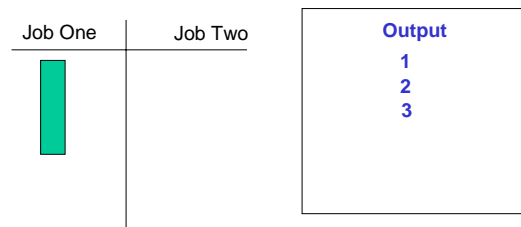
csc 133

Sequential Execution



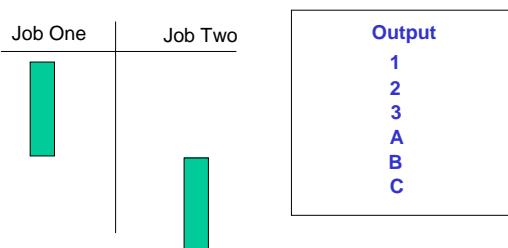
csc 133

Sequential Execution



csc 133

Sequential Execution



csc 133

But what about handling multiple processes at the same time???

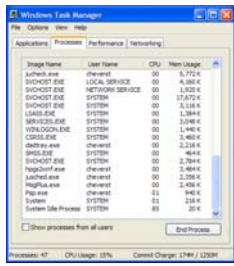
csc 133

Multiple processes...

- Explore windows task manager...

```

C:\Windows\system32\cmd.exe /s /c ps
ps
  PID  PPID  MEMS  VSZ  RSS  TTY  STAT  START  TIME  COMMAND
root   339  0  0  1.1 2240 1224  rtyt  0  07:50  0:00  -bash
root   400  0  0  0.1 1364  488  rtyt  0  07:50  0:00  /usr/sbin/sshd
root   411  0  0  0.1 1364  488  rtyt  0  07:50  0:00  /usr/sbin/sshd
root   412  0  0  0.1 1364  488  rtyt  0  07:50  0:00  /usr/sbin/sshd
root   413  0  0  0.1 1364  488  rtyt  0  07:50  0:00  /usr/sbin/sshd
root   414  0  0  0.1 1364  488  rtyt  0  07:50  0:00  /usr/sbin/sshd
root   445  0  0  1.1 2212 1224  rtyt  0  09:10  0:00  -bash
root   470  0  0  0.1 2020  816  rtyt  0  09:10  0:00  /usr/bin/passwd
root   483  0  0  0.1 2020  816  rtyt  0  09:10  0:00  /usr/bin/passwd
root   488  0  0  0.1 2020  816  rtyt  0  09:10  0:00  /usr/bin/passwd
root   512  0  0  0.1 1364  488  rtyt  0  09:10  0:00  /usr/sbin/sshd
root   513  0  0  0.1 2436  816  rtyt  0  09:10  0:00  ps -eo
  
```



Or the 'ps' command on Unix...

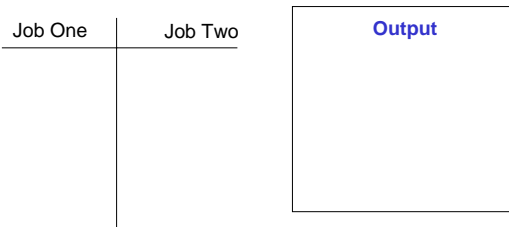
csc 133

But what about handling multiple processes at the same time???

- Concurrent
 - doing things (apparently) at the same time...
- Parallel
 - doing things really at the same time...

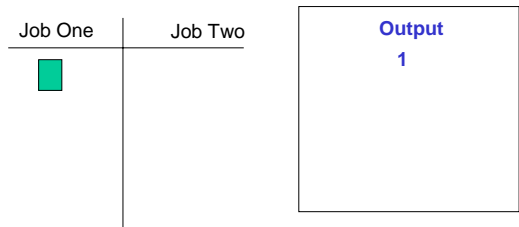
csc 133

Concurrent Execution



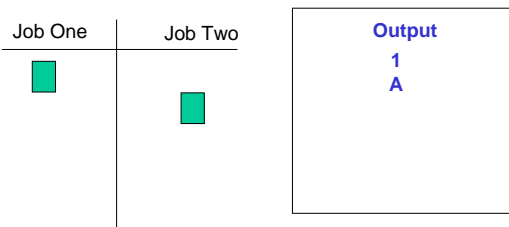
csc 133

Concurrent Execution



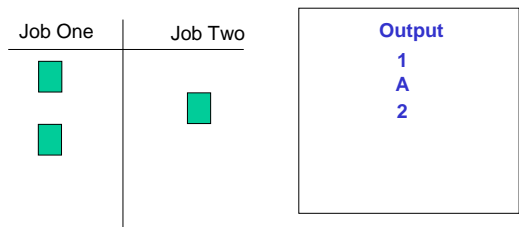
csc 133

Concurrent Execution

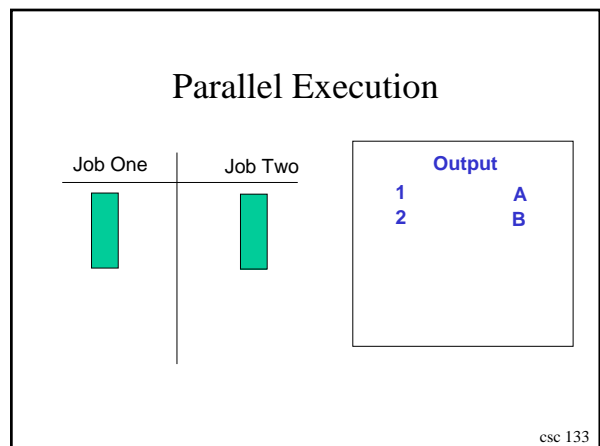
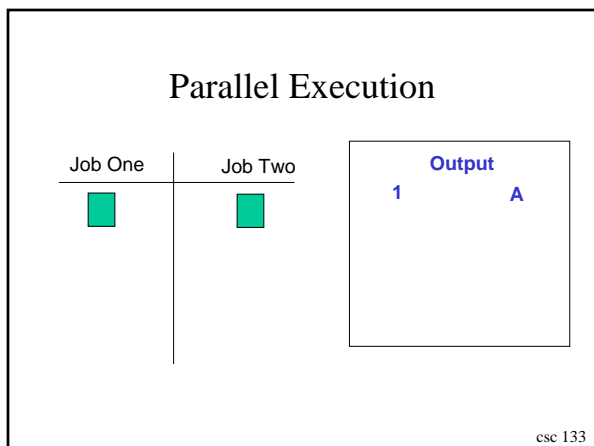
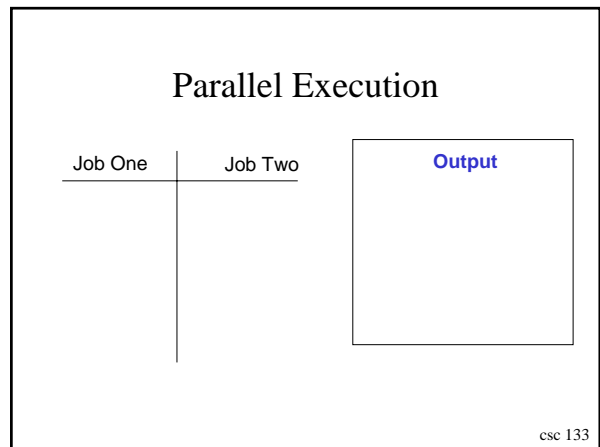
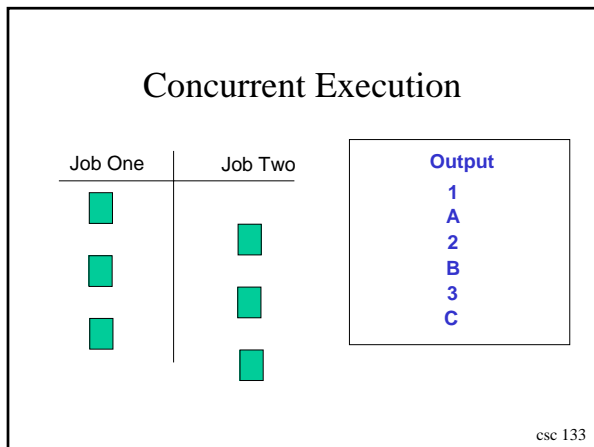
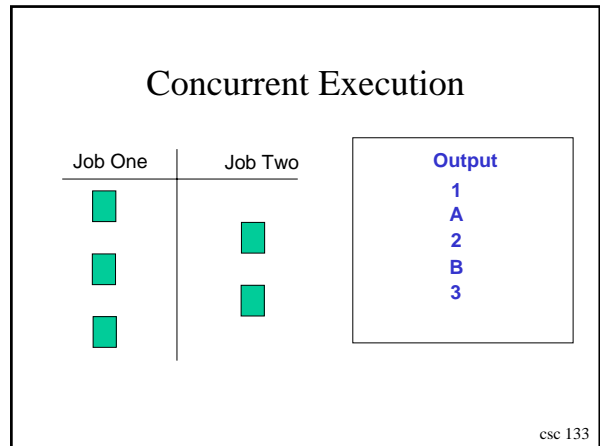
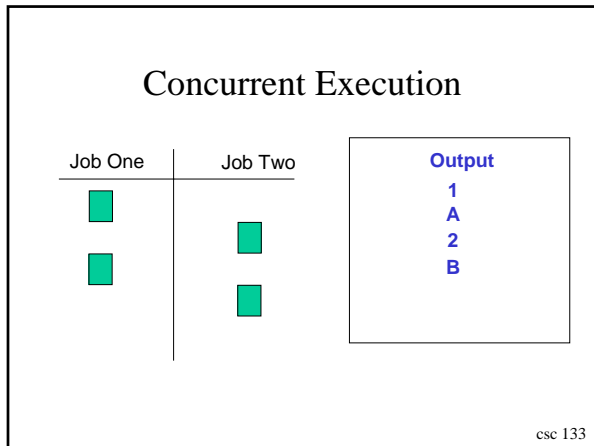


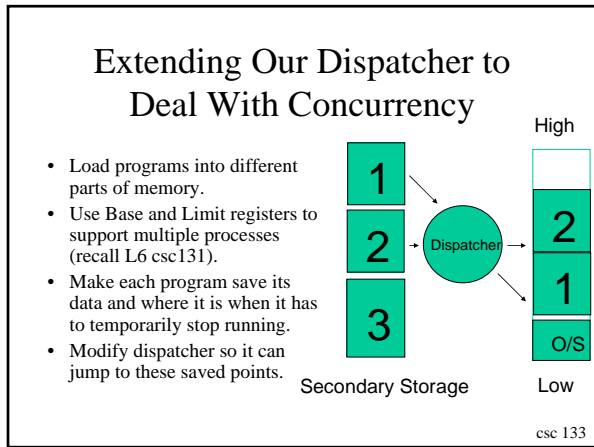
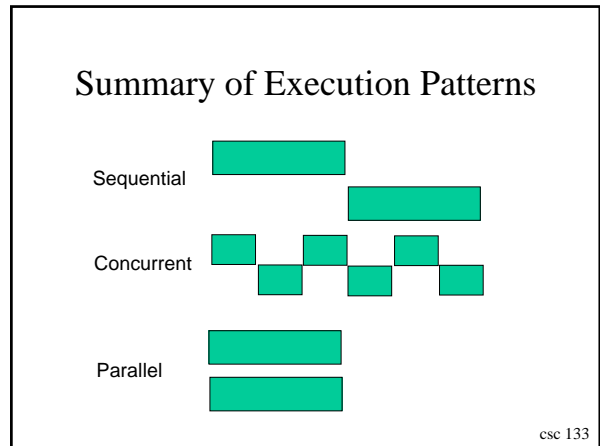
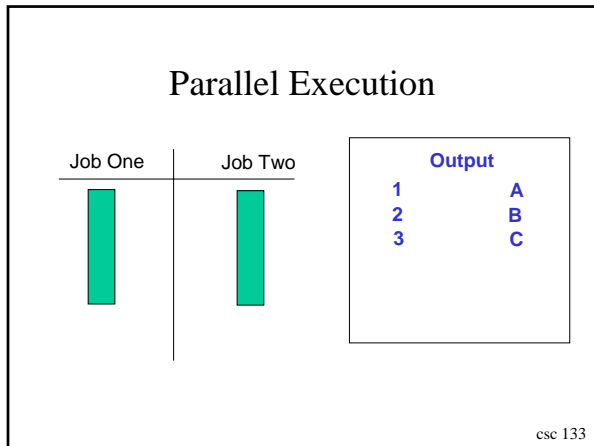
csc 133

Concurrent Execution



csc 133

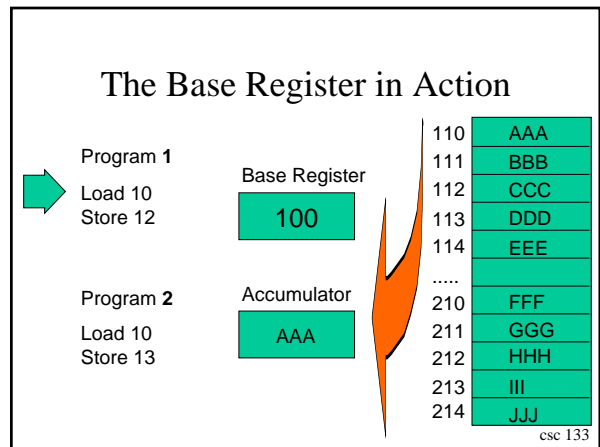
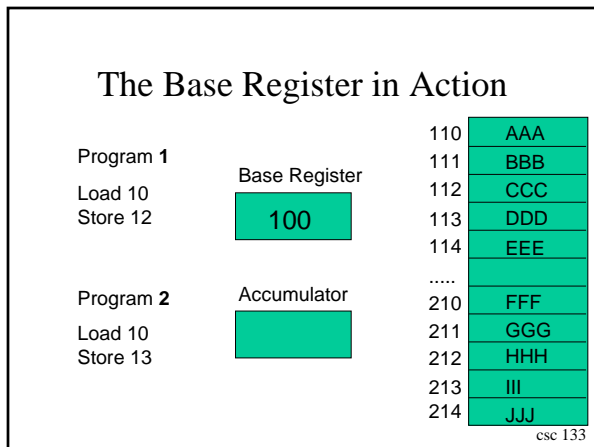


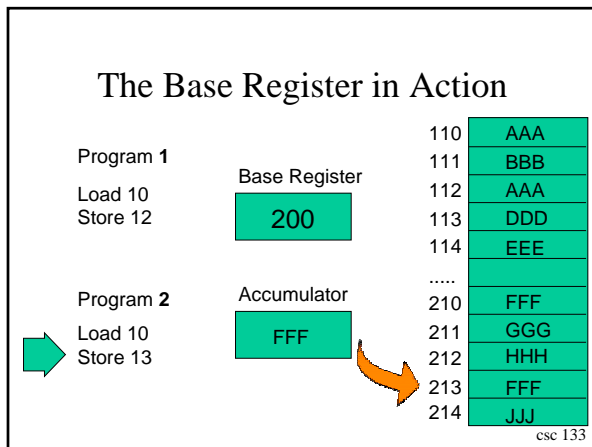
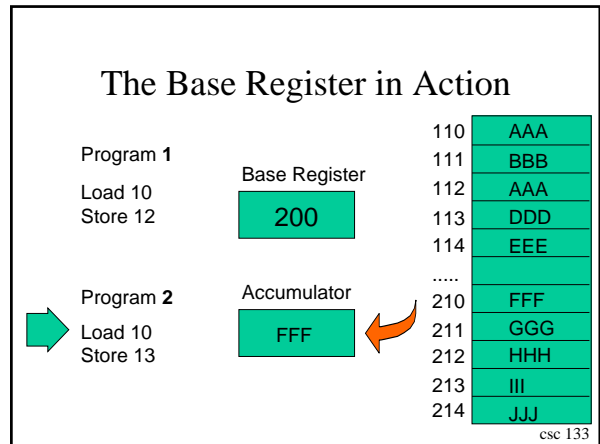
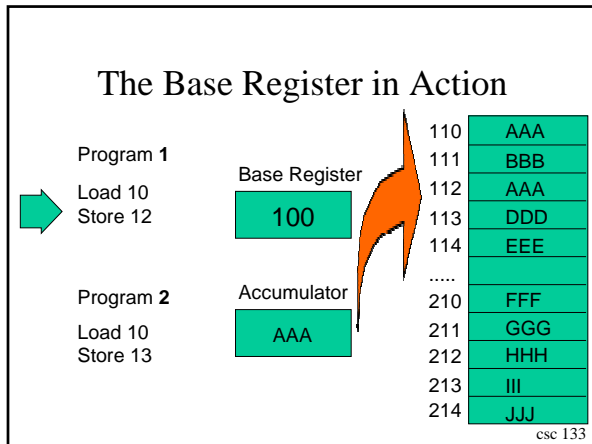


Relocating Code: The Base Register

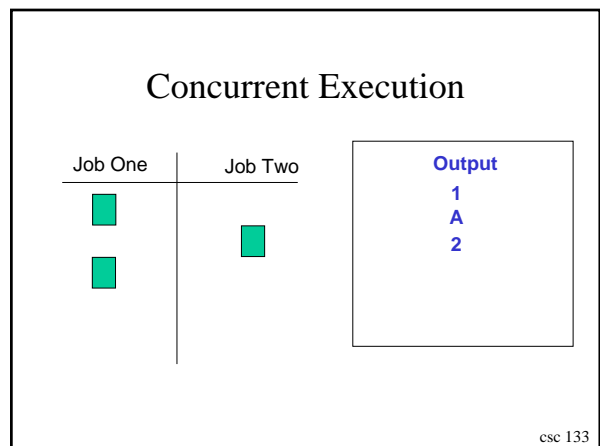
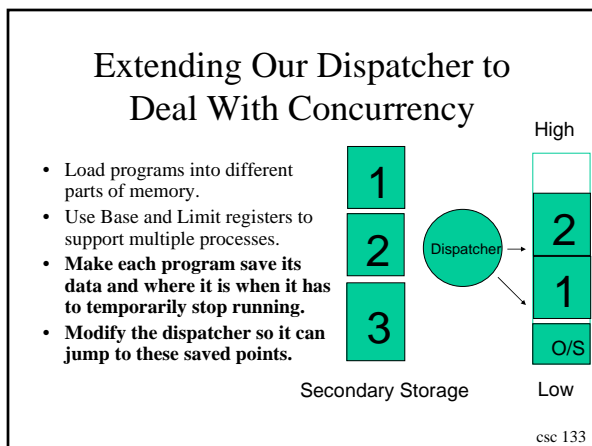
- If we want to run more than one program at a time we need to be able to relocate programs.
- If we can't then we'd have to write every single program such that they all used different addresses.
- Hardware support for relocation can be provided in the form of a base register.
- Value in the base register is added to each operand before address resolution.

csc 133





- ### Protecting Programs From Each Other
- If multiple programs are running we can check that they stay in their own sections of memory using a limit register.
 - Each store access is then checked against the limit register to ensure that it is within the programs allocated store.
- csc 133



Process Context

- Need to save important pieces of information
 - e.g. accumulator and PC
- In practice processes have much more *context* that needs to be saved.
- Term *process context* is used to describe all of the state a process must save to be able to resume execution from the same point.
- N.B. it takes a (relatively) significant period of time to restore the process context and to switch between processes...

csc 133

Which Process to Run

- Question: Given this time overhead, why stop a process and change to another during its execution ?
- Answer: Lots of reasons
- Want to give other processes a chance to run (concurrent execution).
- The process can't continue until another process has finished its task.
- The process is waiting for something to happen (e.g. I/O).
- An exceptional event has occurred (i.e. an interrupt).

csc 133

The Process State Model

- The decision on which process to run therefore has to be based on knowing whether the process can be run.
- During their lifetime processes typically cycle through 3 states
 - Ready
 - Running
 - Blocked

csc 133

The Process State Model .. contd



csc 133

The Process State Model .. contd

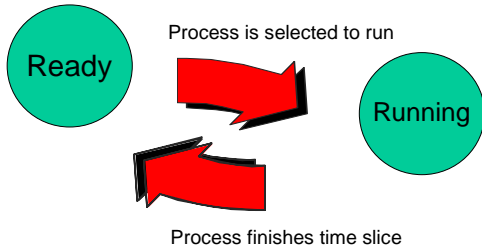


csc 133

Either

csc 133

The Process State Model .. contd

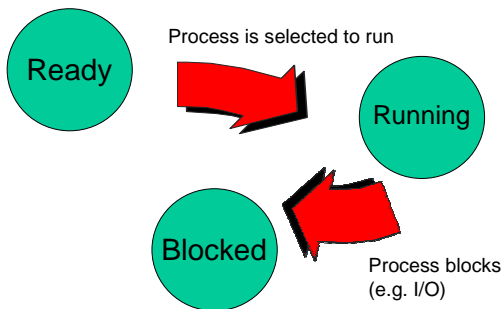


csc 133

OR

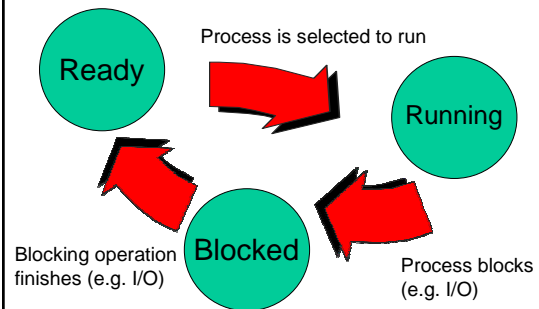
csc 133

The Process State Model .. contd



csc 133

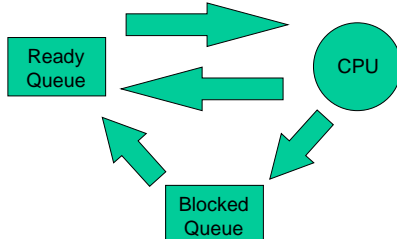
The Process State Model .. contd



csc 133

Resource Issues and Scheduling

- General scheduling model shown below.

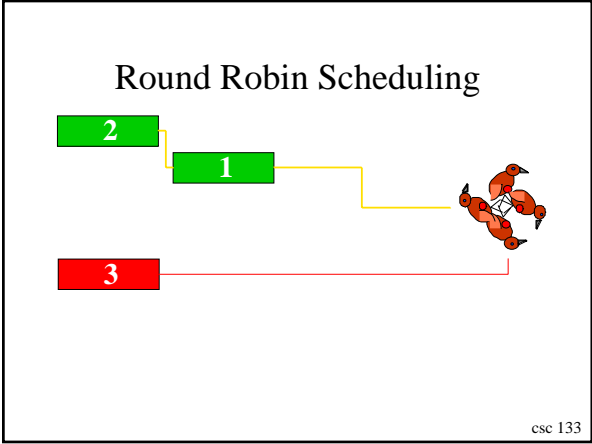
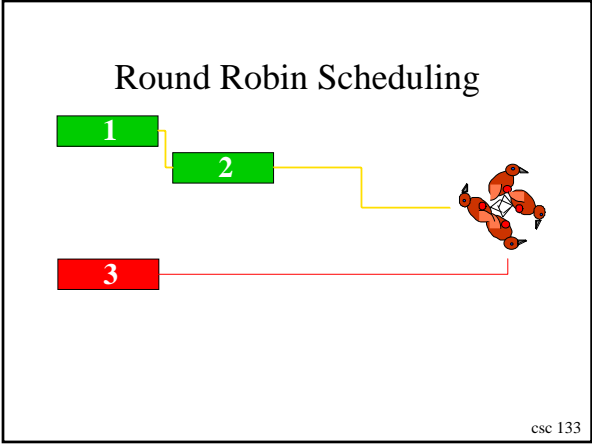
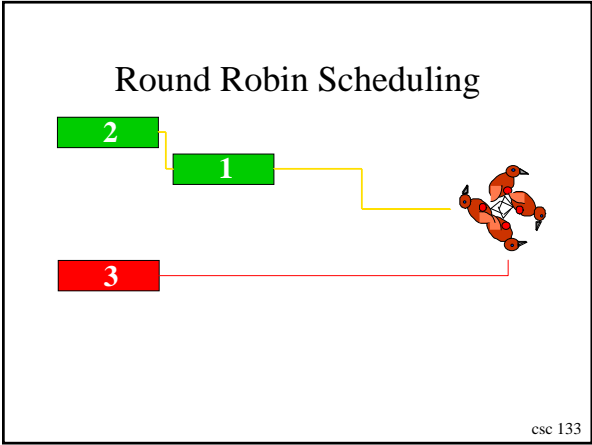
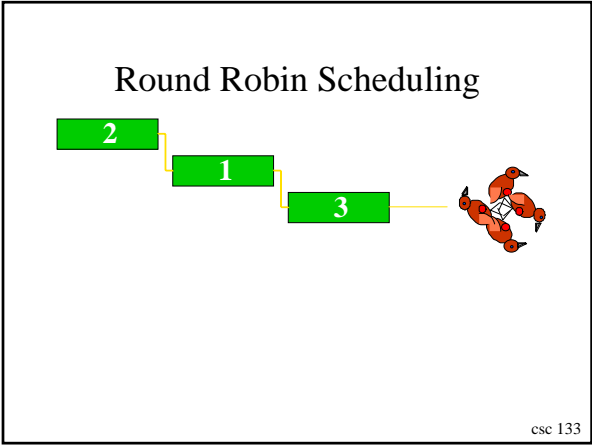
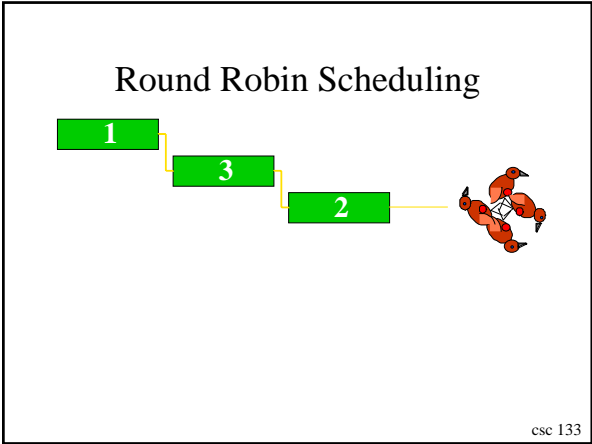
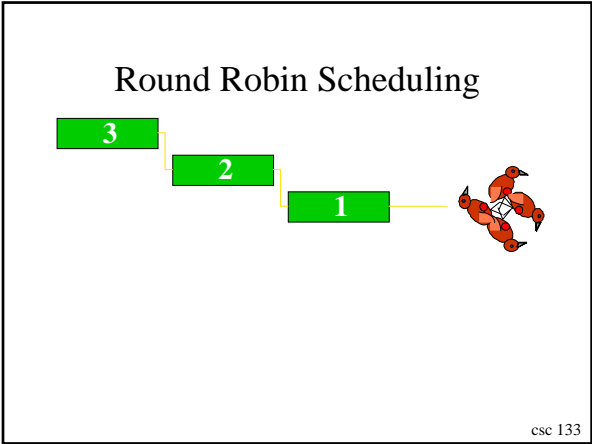


csc 133

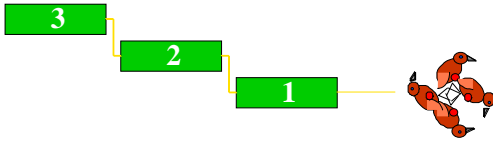
Scheduling and the Dispatcher...

- Higher level decisions taken by a process called a scheduler.
- Scheduler tries to make 'best' use of resources.
- Most systems use some form of round-robin scheduling algorithm.
- Simple scheme which gives each non-blocked process a fixed size time-slice.

csc 133



Round Robin Scheduling



csc 133

Priorities

- Sometimes want different priority processes.
 - e.g. processes associated with interactive applications
- These can be implemented by either:
 - Multiple ready queues.
 - Giving different processes different length time-slices.
 - Giving different processes more time slices.

csc 133

Minimising Transitions to the Blocked State

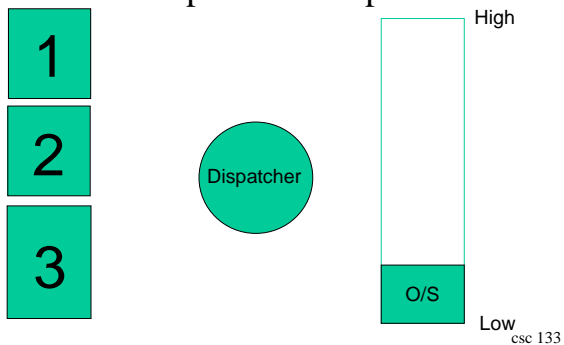
- Important: transitions to blocked state increase overhead because processes not running for their full time-slice.
- Can minimise number of transitions by ensuring resources are available when required.
- This requires additional scheduling strategies, e.g. give resource intensive processes a high priority.
 - Why ?

csc 133

The fragmentation problem...

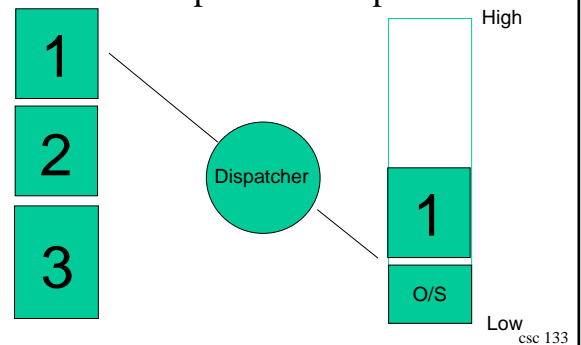
csc 133

The Dispatcher in Operation

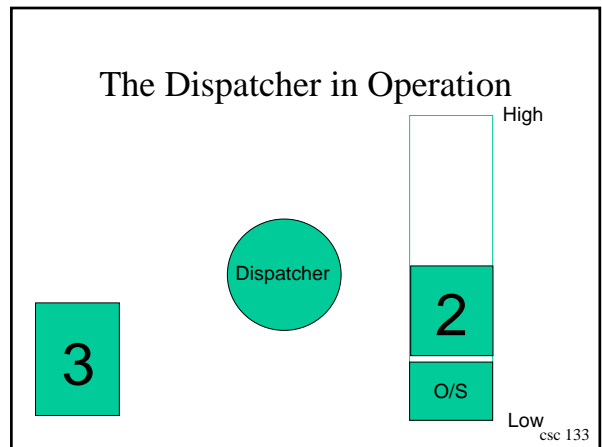
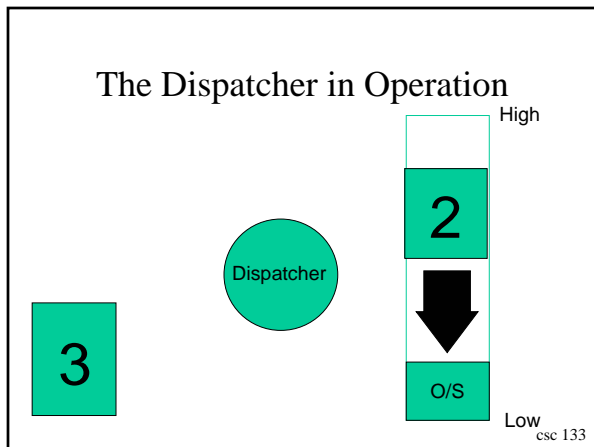
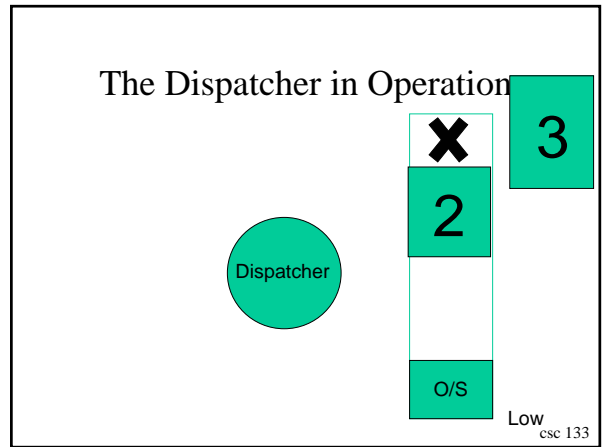
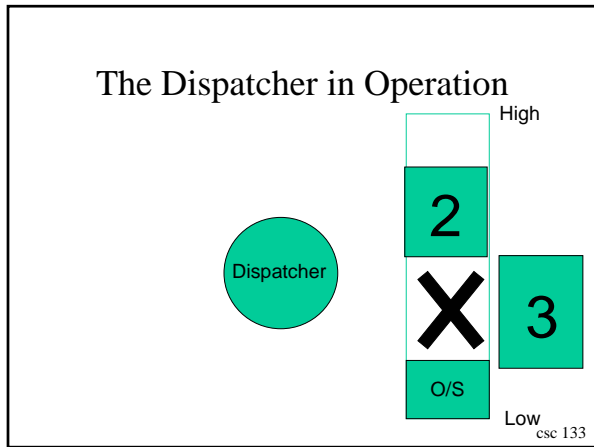
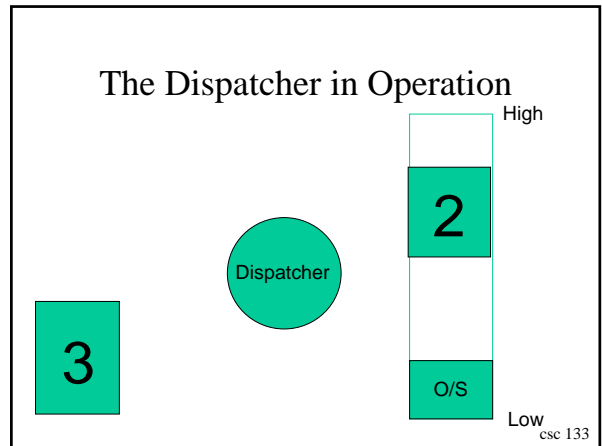
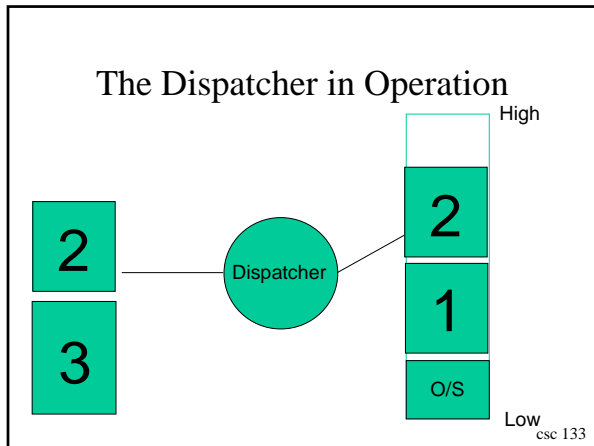


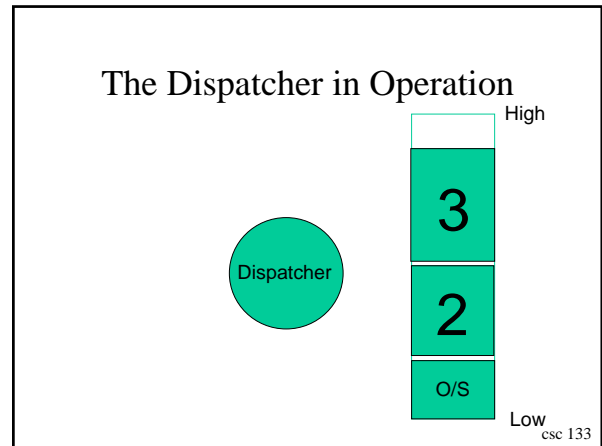
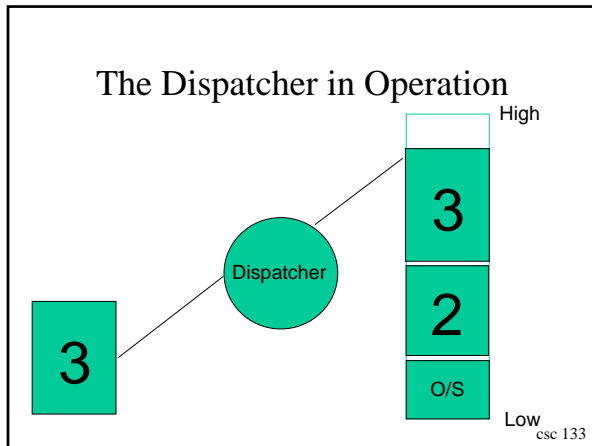
csc 133

The Dispatcher in Operation



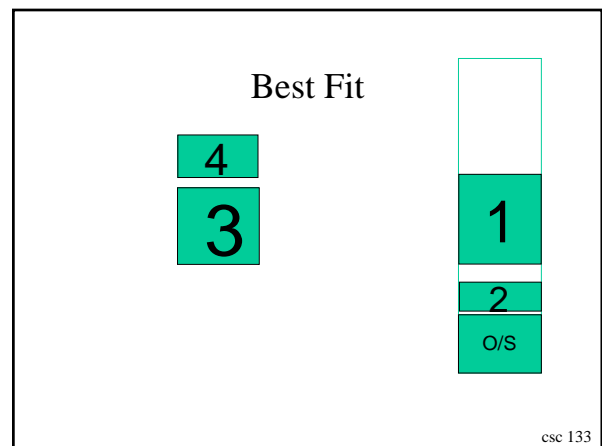
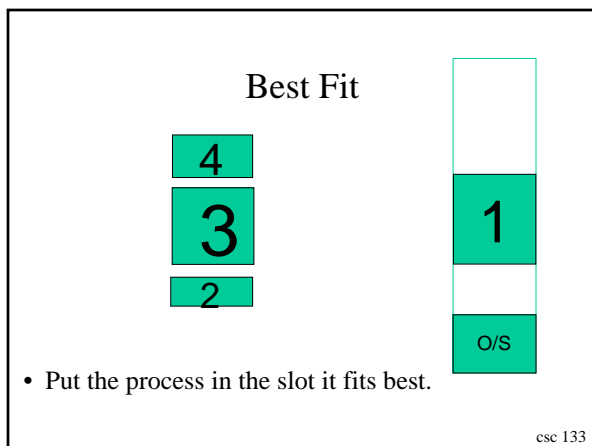
csc 133

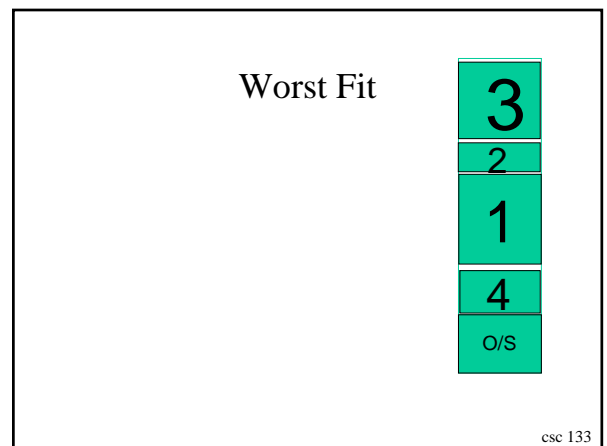
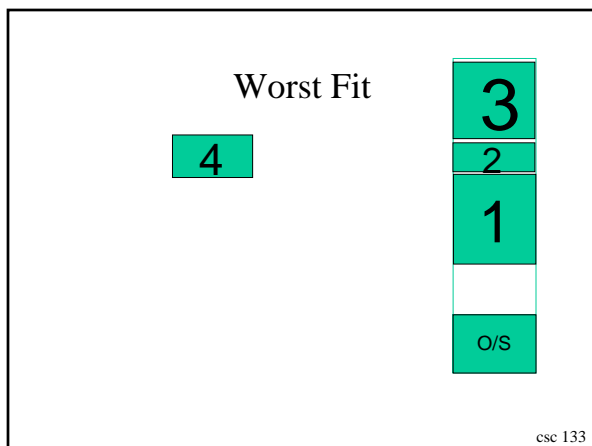
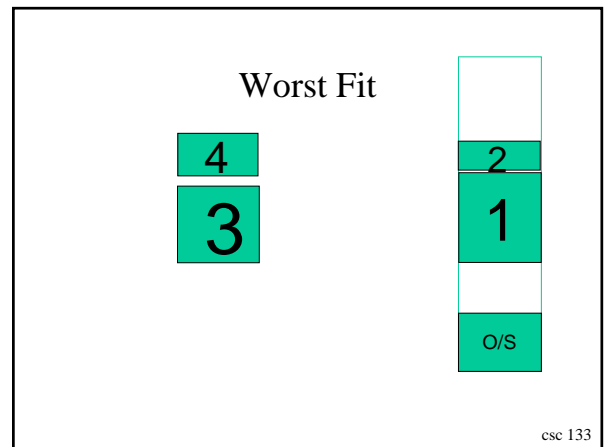
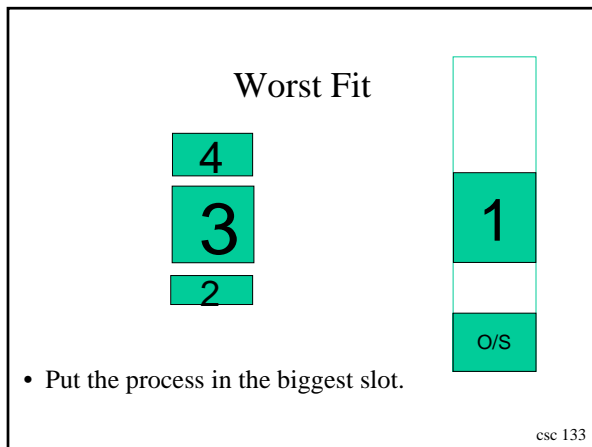
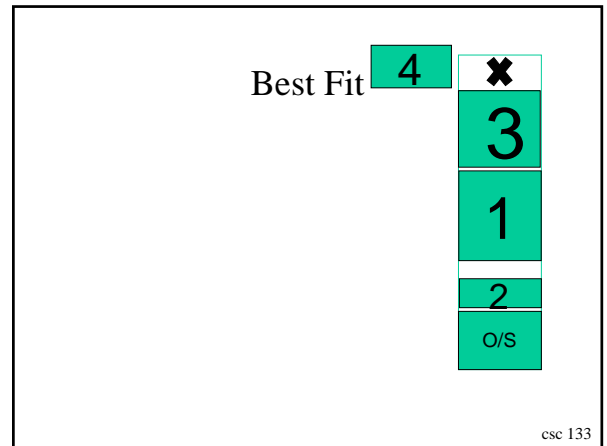
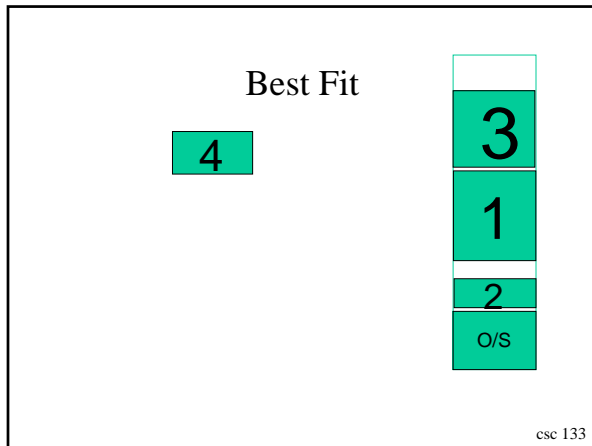




- ### Some Terms
- The reason process 3 couldn't fit was because the memory had become fragmented.
 - The solution is to compact the processes currently running.
 - But ... compaction is very slow (lots of copy instructions).
- csc 133

- ### Reducing the Impact of Fragmentation
- Selecting the optimum place in memory to start a process can help reduce the effect of fragmentation.
 - Three common solutions:-
 - Best Fit
 - Worst Fit
 - First Fit
- csc 133





First Fit

- Save time by not searching through the lists, just put it in the first biggest slot.

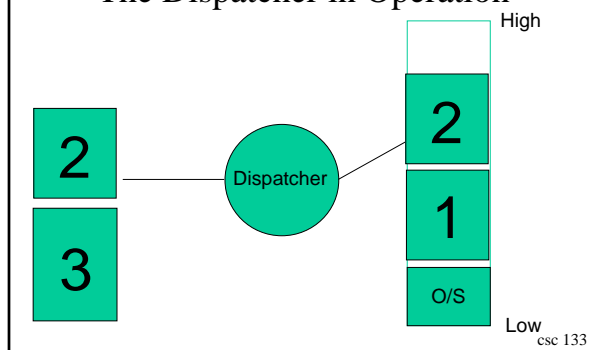
csc 133

Swapping

- Suppose in the original example program one hasn't finished - just been blocked....

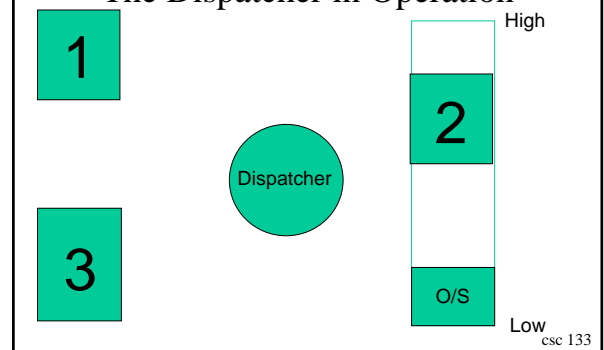
csc 133

The Dispatcher in Operation



csc 133

The Dispatcher in Operation



csc 133

Swapping

- Suppose in the original example program one hasn't finished - just been blocked.
- We can still run process 3 if we can shift process 1 out of memory while it is blocked.
- Best (only) place to put it is on disk - this is called *swapping*.

csc 133

Topic 1: Summary

- Introduced OS as a virtual machine
 - Sharing resources
 - Optimising use of resources
- Explored loading and running of programs by the dispatcher
 - Introduced notion of a process
 - Described different execution patterns: sequential, concurrent and parallel.
- Switching between concurrent processes
 - Process state model
 - Higher level decisions taken by a process called a scheduler
 - Making 'best' use of resources.
 - Most systems use some form of round-robin scheduling algorithm
 - Gives each non-blocked process a fixed size time-slice.
- Approach for supporting concurrent processes
 - Use Base and Limit registers to support multiple processes.
 - Swap blocked processes onto disk to free up memory.
 - Pick appropriate placement strategy to minimise effect of fragmentation.
 - Compact when necessary.

csc 133