Accredited

AS and A LEVEL

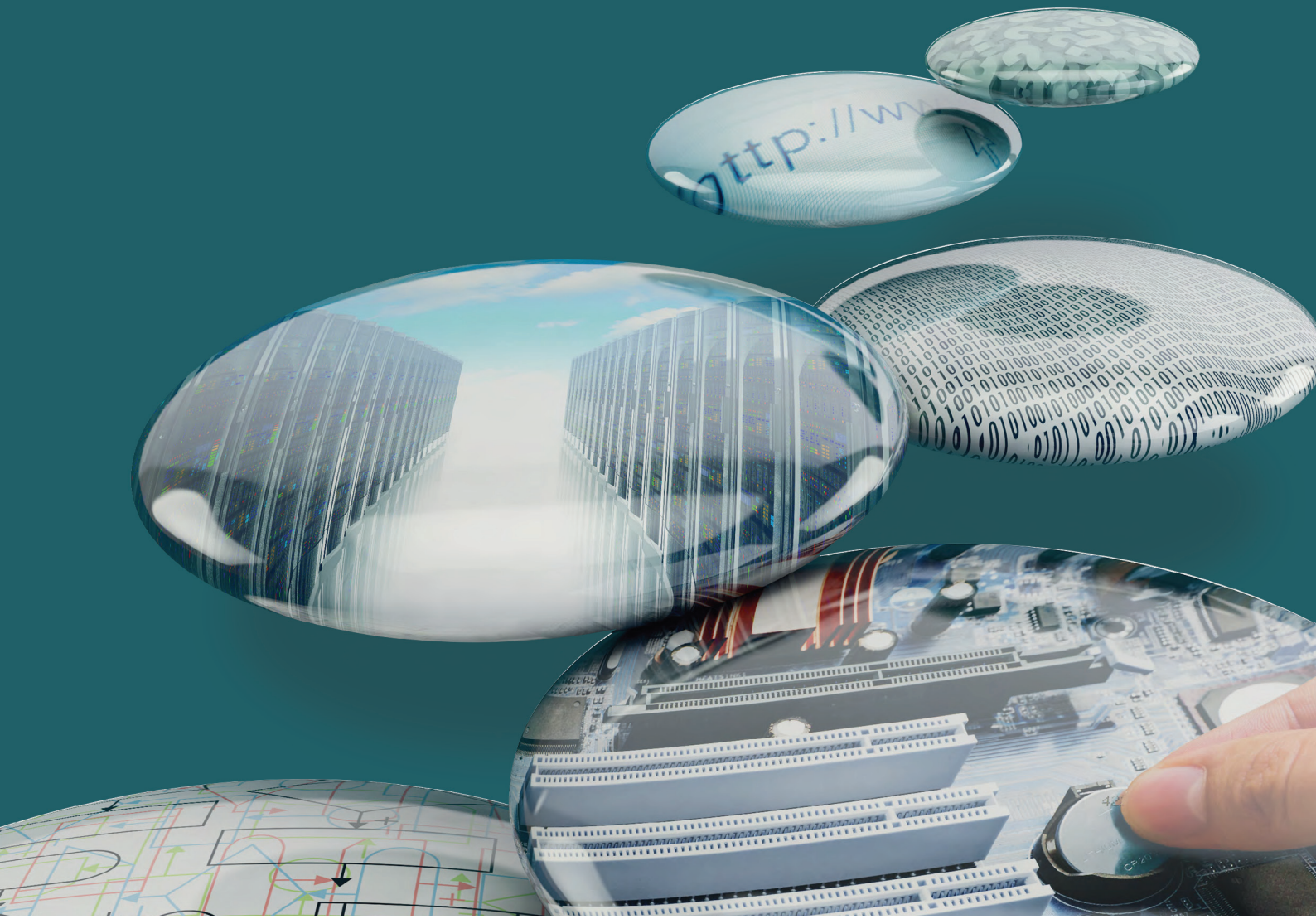
Topic Exploration Pack

H046/H446

COMPUTER SCIENCE

Theme: Thinking concurrently

September 2015



CAMBRIDGE UNIVERSITY PRESS
www.cambridge.org

OCR
Oxford Cambridge and RSA

We will inform centres about any changes to the specification. We will also publish changes on our website. The latest version of our specification will always be the one on our website (www.ocr.org.uk) and this may differ from printed versions.

Oxford Cambridge and RSA Examinations is a Company Limited by Guarantee.
Registered in England. Registered company number 3484466.

Registered office: 1 Hills Road
Cambridge
CB1 2EU

OCR is an exempt charity.

Contents

This Topic Exploration Pack should accompany the OCR resource ‘Thinking Concurrently’ learner activities, which you can download from the OCR website.

This activity offers an opportunity for English skills development.

Thinking Concurrently

For the topic ‘ Thinking concurrently’ students will need to be familiar with the following points:

a) Determine the parts of a problem that can be tackled at the same time.
b) Outline the benefits and tradeoffs that might result from concurrent processing in a particular situation.

Concurrency is known as when several things are happening simultaneously. For instance in the example of a Gantt chart (see ‘Thinking ahead’) some of the processes that happen will inevitably be happening at the same time. Building a house, planning a party, division of labour for a production line.

A description of a simple task that might be adapted to be concurrent is:

- Take a dirty plate
- Wash the plate
- Dry the plate
- Put the plate on a shelf
- Continue until there are no dirty plates

If another person was assigned to help you with the task, labour could be divided between you. You may decide to do things in a different order (perhaps putting a few plates into a pile before putting them on the shelf), and in this case you have access to the same resources as each other, but essentially if someone washes up and someone puts things away, then the process should be completed faster.

Now, for a more extreme example of this, imagine if your primary objective is to clean the dishes, but you don’t have a tea towel. Your only option is to air-dry the plates on a drying rack. This is obviously going to take quite a while, but you have no choice but to wait around (remember, this is your only objective for now!) until the plates are dry. Wouldn’t it be great if you could do some other stuff while waiting for them to dry?

This is the whole point of working concurrently, so that whilst you are waiting for that process to finish you can check your emails on your phone, go to the toilet, make dinner etc.

In computer science we usually start off using single-threaded programming, i.e. you can pretty much put your finger onto the code and follow it through the sequence of instructions, jumps to different function calls etc.

Using multiple threads is like having a few fingers on your code where they are all executing separately. Each finger still moves the same way, but they all move separately.

Computers with multicore CPUs can run multiple instructions simultaneously, but in Python there is a Global Interpreter Lock (GIL) that limits a python program to one core only. The python interpreter will run for a little while and then pause to switch to another thread, but it does this so fast that they seem to be running simultaneously.

Activity 1

Out line some of the problems that might occur if you assigned 3, 4 or 5 people to washing up the dishes. Would you expect the time it takes to go down more?

Threads are best used on programs that have to wait for IO (Input or output) routines, for instance, constantly polling a server to check for emails in the background, reading/writing to and from files or taking care of hardware routines like flashing lights. If we did any of these things in a single-threaded fashion we could not expect our program to be responsive to input, and in the case of GUI programming would cause the buttons etc to be unresponsive until the current process had finished what it is doing.

Prior knowledge

Students will need knowledge of dictionaries and functions to complete the first activity. The activity could be adapted for other programming languages.

Common student misconceptions

Working concurrently in programming is actually quite a hard thing to achieve, especially where threading is involved.

However, the concept of wanting to do things at the same time should make sense to students.

More on this can be found at: http://inventwithpython.com/blog/2013/04/22/multithreaded-python-tutorial-with-threadworms/

Activity 2 – Threading in python

This practical activity shows students some of the good and bad sides of threading in a simple way.

The worksheet provided gets students to produce a very simple program that creates 10 threads that all finish after 5 seconds, showing that they work concurrently, the students then extend this to set the delay to 1, 2, 3 seconds etc. (This is given as 01_threads_simple.py)

Students are then asked to create a menu system to allow them to input values for the message and for the delay. This is where the students should get stuck, since the interpreter is 'locked up' waiting for user input, so it can't output the values. (This is given as 02_threads_menu.py)

A third implementation uses the EasyGUI library, which 'frees up' the interpreter. (This is given as 03_threads_easygui.py)

This activity should take roughly 2 hours to complete.

Activity 3 – Routing and deadlock

This activity from CS Unplugged illustrates the sort of problems that might occur when resources (for example roads or networks) are being shared, and can sometimes be in a state of 'deadlock'.

5 or more students are in each group and each person should be given two objects that are unique to them e.g. two oranges with the letter A on, then the next person has B on etc.This could be more easily achieved with pieces of paper.

A discussion is presented at the end of the activity which details the use of locks in case two processes want a certain piece of data at the same time.

A video is given which very quickly explains the activity

The activity should take roughly 30 minutes.

http://csunplugged.org/routing-and-deadlock

Activity 4 – Dining philosophers

The dining philosophers problem is one of the most popular examples available when it comes to explaining how concurrent processes might be involved in deadlock situations.

The worksheet provided is a bit different from just talking about the problem in hand as the students must first act the problem out and leads them to consider what they could do to prevent deadlock occurring.

If any students are interested there is an implementation of the dining philosophers over at:

http://rosettacode.org/wiki/Dining_philosophers#Python

(NB: The implementation is in Python 2; although all you need to do to make it work with Python 3 is to put brackets around the print functions.)

Activity 5 – Playing card tasking

This next exercise requires a standard set of playing cards.

In groups of four take an equal number of cards at random (deal them). Between you discuss and devise at least two different ways of completing the process of sorting the cards into the sequence Spades (Ace, 2, 3 - King), then Hearts, Diamonds and Clubs.

Hints

Work through the process together initially. Have one person do it and time them. Calculate an average time per sorting operation (by dividing the total time by 52).
Now think about how to allocate tasks - for example one person could deal a suit to each of the four students and the students could then sort them before combining them at the end.

Ponder this...

Is the resulting time a quarter of the original? Why or why not?
Are there more efficient ways to divide up the task?

Can you apply any part of Amdahl's Law (http://en.wikipedia.org/wiki/Amdahl's_law) to this process?

The first activity can be achieved with Python 3 installed, although the very last bit of the activity uses EasyGUI (http://sourceforge.net/projects/easygui/files/0.97/). The students only need to have the 'easygui.py' python file in the same folder as what they are working on to make it work.

For Activity 2 you could have the different objects available (doesn't really matter what as long as there are two of each) to play the routing and deadlock game.

Activity 4 requires the use of decks of cards, so you might want to have them handy.

We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

If you do not currently offer this OCR qualification but would like to do so, please complete the Expression of Interest Form which can be found here: www.ocr.org.uk/expression-of-interest

Version 1

OCR customer contact centre
General qualifications
Telephone 01223 553998
Facsimile 01223 552627
Email general.qualifications@ocr.org.uk