COMP2017 & COMP9017: Systems Programming

School of Computer Science, University of Sydney

Copyright Warning

Week 1: Introduction to the unit

We will cover: UNIX & C basics

You should read: Section 1.1, 1.2 and optional 1.3 - 1.6 of Bryant & O'Hallaron

Textbook
Computer Systems: A programmer's perspective. Third Edition, Randal E. Bryant and David R. O'Hallaron. 2016. ISBN 9781292101767

Lecture 1: Systems Programming

Course overview and general guidelines

COMP2017 & COMP9017

This unit is *Systems Programming* a.k.a. the C course

- Administrivia
- Course Outline
    - Introduction to memory model with C
    - Memory management and aggregate data types in C
    - Parallelism and concurrency and programming in C

Lectures

Coordinator/Lecturer is Dr. John Stavrakakis

There are 12 lectures

Time: Friday 12 - 2pm Lecture Theatre: Eastern Avenue Auditorium

Lecture audio and left screen to presenter are always recorded

Labs

Core experience gained here

Attendance expected

There are 12 tutorials, each is 2 hours

Time: Monday or Tuesday. Refer to your timetable.

Labs (cont.)

Who wouldn't attend their labs?

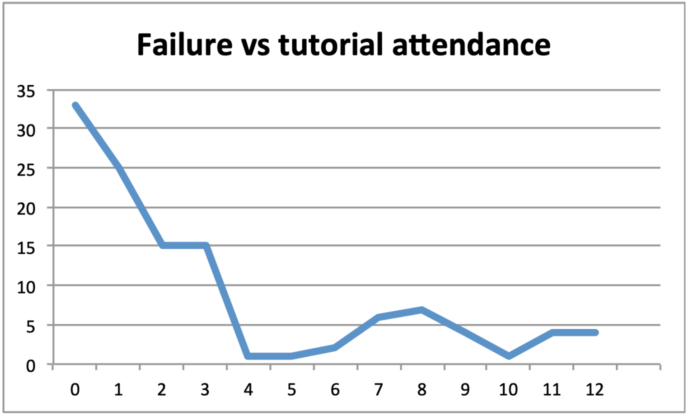


Figure 1: 2017 S1. 481 students

About The Coordinator/Lecturer

The unit coordinator and lecturer is
Dr. John Stavrakakis

PhD in Computer Science
Specialises in 3D computer graphics

Overall course administration and design

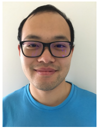
Coordinator/Lecturer for over
`c=680;(rand()%2*(c+920))+c` students over
many courses this semester.

Please be considerate of his time.

About the Teaching Assistants (TA)

Tyson Thomas

tinkerer, sketcher and trekkie



William Wang

med, science, cooking

The teaching assistants help with the preparation and delivery of the course contents. Seminars, labs, tutorials, quizzes, assignments, challenges, computer examinations. They also conduct several other duties.

About our teaching team

We have a fantastic team of tutors. Each are talented in their own regard.

- Tyson Thomas
- William Wang
- Anuj Dhavalikar
- Gregory McLellan
- Dennis Chen
- Byung Hoon Cho
- Andrew Xu
- Xiaowen Hu
- Alistair de Vroet
- Leon Chen
- Hanin Zenah
- Ziyun (Erik) Chi

If you encounter any problems with our tutors, please contact the TA or the coordinator directly.

Online resources: Canvas

This semester Canvas will be used for:

- Accessing your progressive grade
- Web links to other important places

canvas.sydney.edu.au

Lectures are recorded

Available in canvas.sydney.edu.au

Audio is recorded and the screen. The screen images are only those shown on the display that is to the presenters left hand side.

Don’t just listen or simply attend, take good notes. The better your notes the easier it is to revise.

Some, but not all, solutions to tutorial exercises will be provided.

Lecture slides are not full in detail

Online resources: Ed

We keep most of the course materials on Ed

Download lecture slides

Download tutorial exercises

Access assessments

Submit online assessments here!

Ed

Online resources: Ed (cont.)

You will visit Ed quite often.

Announcements, information, updates and discussions

With the discussion forum, keep in mind:

When you post a question choose the most appropriate category.

Anyone posting code solutions to assessments will be banned, and may face further disciplinary action.

Any person doing anything inappropriate will face disciplinary action

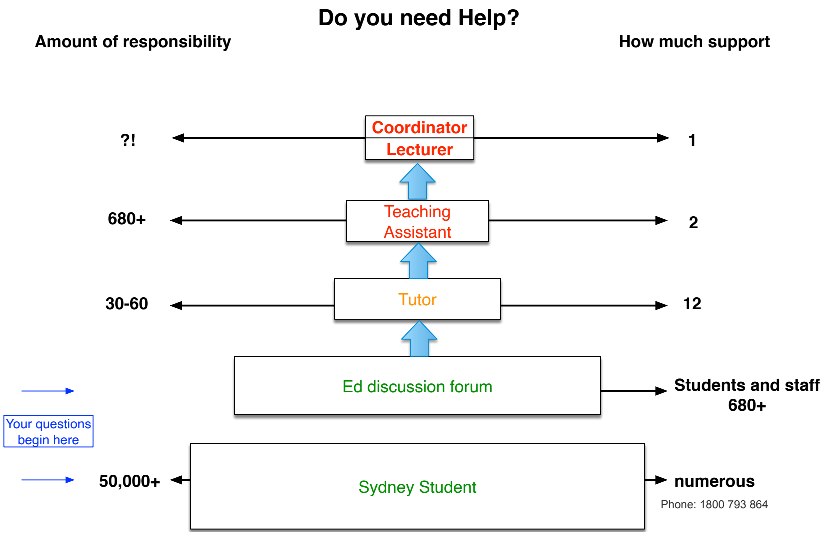
Ed is an excellent platform for seeking understanding of general concepts and clarification of problems.

It is **not** for exchanging code. It is **not** for sharing ideas about assessments.

Where to get help?

1. Student admin → `https://sydneystudent.sydney.edu.au` or contact the Student Centre. Please check these first. E.g. timetable, passwords, payments, enrolments etc.
2. Ed discussion forum → Ed
3. Your tutor in your designated laboratory
4. Contact a TA for administration → Ed (private thread)
5. The teaching assistants email:
   Tyson Thomas, William Wang,
6. Consultation with Dr. John Stavrakakis:
   14:00 - 15:00 Friday, Zoom Link

The hierarchy of help



Learning outcomes

Systems programming will take more time to get right than applications programming

These are outcomes of a pass grade in this course:

- Learn how to program with C
- Learn about memory models of programs
- Learn about parallel and concurrent programming
- Learn how to access and manipulate memory in C

And the ultimate goal is...To write good software

Course outline

Introduction to C and Unix

Programming in C and its memory model

Program development in the Unix environment

Parallel and concurrent programming concepts

Task-parallel programming in C (Pthreads)

Designing and measuring parallel programs

Weeks 1 - 6: C and UNIX

Understanding and being familiar with manipulating data using UNIX programs

Understanding theory and constructs of organising and accessing memory

Understanding the compilation/assembly of programs

Coverage of the core subset of the C language

Weeks 7 - 13: Concurrency

Theory in parallel and concurrent programming

Synchronisation of processes

Implementation using C language & pthreads

On performance: deeper understanding of memory costs/tradeoffs, processing capacities

Expectations

Assumed Knowledge

- Programming knowledge of Java or other similar language
- Data structures is a co-requisite of this course

BEFORE each tutorial

- Read entire tutorial
- Attempt as many exercises as possible
- Bring questions (or ask on ed)

BEFORE each lecture (week 2 - 13)

- Read the related chapter(s) and lecture notes

Expectations

Data structures:

- Stack
- Linked lists
- Queues
- Binary trees
- Hashing
- Graphs

Skills/concepts:

- Unit testing (i.e. junit)
- Algorithm complexity - Big Oh notation
- Recursion
- Sorting algorithms
- Modular solutions (development practice)

Keep it up!

Make sure you keep up with the course

Consult staff EARLY enough for us to help you if you have difficulties or problems!

A reminder that all assessments in this course are **individual effort**.
It is up to you to interpret, understand, solve and implement a solution to a problem. You are responsible for what you will present as your knowledge.

- do not ask your friends.
- do not seek answers, or knowledge about how to begin, or complete an assessment.
- do not copy code from a source without proper citation (see Ed resources on citing examples). Be also aware that citation will impact the grading of an assessment.

Unit of Study information

Course conditions, assessment details and schedule are available on the Unit of Study website:

COMP2017 UoS
COMP9017 UoS

Communication of changes to assessment details will be disseminated via Ed. Students are expected to observe these announcements on Ed and check their unikey email as required for all official university communication.

Assessment components

Assessment	**Due Date**	**Weighting**
P1	11:59 PM, 21 March 2021	10%
P2	11:59 PM, 11 April 2021	15%
P3	11:59 PM, 26 April 2021	15%
P4	11:59 PM, 09 May 2021	15%
P5	11:59 PM, 30 May 2021	15%
Final Exam	Formal Exam Period	30%

Note: all due dates are Sydney local time

Final examination - 30%

The computer examination in this course is similar to an online programming problem. You will be given a problem to solve, but only a fixed time to work on the solution.

The final examination will be 2 hours

It will contribute 30% to your final grade

This is a CLOSED book examination - no material permitted

Platform TBC. Likely Ed git submissions

P assessments 70%

Programming problem

Multipage description

Hundreds of lines of code

1. You write code
2. You write lots of tests
3. You are graded on these and the code itself

Late submissions are penalised as per Unit of Study details.

P assessments 70% (cont.)

Following the submission of the assessment. An oral examination session will be arranged with your tutor.

You will be asked to explain your code to the tutor

Your tutor will ask questions about your understanding of topics in Systems programming based on your explanation.

Absence or Lateness for the the session will result in zero marks.[1]

P assessments 70% (cont.)

All assessments with the session are marked by the tutor. What do you need to consider for these assessments?

- the code must compile
- you make an attempt in the oral examination as described in the assignment description. If it asks to explain a section of code, or an idea, you are prepared and give a decent explanation to that.
- you have handled the error cases presented in the description
- you have tested your own code
- invalid assumptions are void, especially where it trivialises the solution. It cannot be considered. Alway ask.
- if there are issues such as off by one in the code, logic errors. These are not assessed manually.

P assessments 70% (cont.)

- If there is a non-trivial attempt of oral examination, then the assessment will not be marked. You should try, if you don't try, we will know. The point is to work with your tutor to understand the course contents better.
- there will be private test cases that will run on your code
- the code is clean, legible (refer to the coding style guidelines)

[1] exception is Special Consideration

Assessments

All assessments are *individual* work.

The *progressive mark* are all P assessments combined (70%).

You must get ≥40% of the progressive mark and you must get ≥50% of the *paper exam mark* to be *eligible* to pass.

You must also get a combined mark of at least 50% in total, of course[2].

Here are some examples of how this works, in case the above isn't clear:

$\boxed{\text{ProgMark 44\%, Final Exam Mark 50\%, total 48\%: FAIL}}$

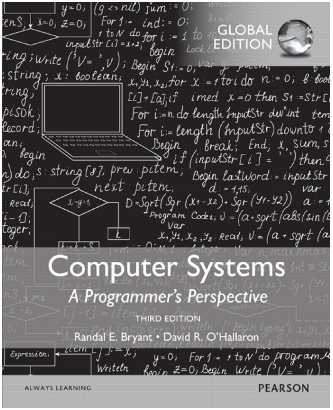
$\boxed{\text{ProgMark 75\%, Final Exam Mark 35\%, total 55\%: FAIL}}$

$\boxed{\text{ProgMark 22\%, Final Exam Mark 80\%, total 51\%: BUT FAIL}}$

[2] If you are having trouble working this out, you are already in trouble.

The course textbook

- This is a good reference textbook
- A super useful catalogue of computer system innards
- It covers many parts of the course except for UNIX
- Course is not based on the book, but the course will make references to selected parts of the book
- The book does cover all topics in this course



Reference books

- The practice of programming, Kernighan & Pike. Addison-Wesley, 1999. ISBN: 020161586X
- Principles of Parallel Programming, Lin & Snyder. Pearson Education, 2008. ISBN: 9780321549426
- Problem Solving and Program Design in C, Jeri R. Hanly, Elliot B. Koffman. Addison Wesley. ISBN: 0321198034
- The Indispensable Guide to C, Paul Davies. Addison Wesley, ISBN: 0-201-62438-9
- The C Programming Language, Second Edition, Brian W. Kernighan and Dennis M. Ritchie. Prentice Hall, Inc., 1988. ISBN: 0-13-110362-8
- Numerical recipes in C, 2nd Edition. William H. Press, Saul A Teukolsky, Saul A. Teukolsky, William T Vetterling, Brian P Flannery. Cambridge University Press 40 W. 20 St. New York, NY United States. ISBN: 978-0-521-43108-8. `http://www.nrbook.com/a/bookcpdf.php`

Software

- Unix based operating system
- C99 supported. GNU C Compiler or clang
- BASH command line programming/scripting, standard UNIX commands
- git for source code management
- pthreads, epoll, semaphore

no fancy IDEs!

Automatic Marking

All assessments that require submission of a computer program can utilise automatic marking.

When tasked to write a program. The program **must compile** against the target architecture specified in the assessment.

A program is correct if it achieves the outcomes as specified in the assessment description. Specifically, the input of the program produces a specific output **all the time**. [3]

Automatic Marking (cont.)

It is generally assumed that students would produce a correct and functional program as an output of any assessment other than where it is explicitly stated.

We use several ways to test your program. Generally, input vs output.

We give your program input data and compare the output with what we expect.

Automatic Marking (cont.)

The *kind* of testing that is done by automatic marking is made known to you. Not everything is known to you.

Pay attention to the description to derive your own test data before you submit.

Anything that is not hand written, tasks, assignments, computer examinations will be marked by computer. There is no hand marking of code that does not compile.

Automatic Marking (cont.)

For example, suppose you have a program called `launch_sequence`, which should output the binary instructions to talk to a device:

```
    > ./launch_sequence begin
    03 02 01 00 6d 61 6b 65 20 72 6f 63 6b 65 74 20 67 6f 20 6e 6f 77 21
    >  ./launch_sequence abort
    10 10 41 62 6f 72 74 2e 10 41 62 6f 72 74 2e 10 10
```

For systems programming. That is how it *must* be.

```
      > ./launch_sequence begin
FAIL 03 02 01 00 6e 61 6b 65 20 72 6f 63 6b 65 74 20 67 6f 20 6e 6f 77 21
FAIL 03 02 01 00 6e 61 6b 65 20 32 72 6f 63 6b 65 74 20 67 6f 20 6e 6f 77
FAIL 03 02 01 00 6e 61 6b 65 20 72 6f 63 6b 64 74 20 67 6f 20 6e 6f 77 21
FAIL 03 02 01 6e 61 6b 65 20 72 6f 63 6b 65 74 20 67 6f 20 6e 6f 77 21
```

Automatic Marking (cont.)

We have many tests for each program you write. To get full marks you must pass each test.

We also keep tests *hidden*: you won't know until after the deadline how you've gone on those.

That means you'll have to really understand your code and think carefully about all kinds of possible inputs to ensure your program will handle them properly.

[3] The exception being cases which randomness are presented as part of the problem, or instability of the algorithm's outcome based on the input.

More details on submission

The program *must be written in C* and must compile and run on the lab computers or other platform that is specified in the assessment (Ed typically, though please read instructions carefully)

The proportion of tests your code passes will be used in the manual marking process for each assessment.

You may be required to *explain* your code to your tutor, or to the Unit Coordinator (i.e. me). If you can't explain what it does, you won't get a mark.[4]

Don't get other people to do your assessments for you. Really.

[4] If you can't understand it, why would you submit it?

Academic dishonesty

☠ ***We run tests on your submissions to see how similar they are to those of other students.***

The software we use is very good at helping us detect different kinds of academic dishonesty, so don't do it.

The software we use is very good at detecting and alerting staff for dishonest practices. Staff are able to confirm such activity through investigation. So please don't do it.

☠ ***We have failed students in the course on the basis of academic dishonesty***

☠ ***Plagiarism, Collusion, Contract cheating — submitting someone else's work as your own — will not be tolerated. You MUST read and understand the University's policy documents on Academic dishonesty and plagiarism. We use both electronic and human means to identify dishonesty. You have been warned.***

What can you cannot expect, or ask help for

All assessments in this course are an individual's effort

You cannot discuss with your friends about the assessment, or share any code. You cannot describe, dictate, explain, draw, or communicate in any way how you solved the assessment or even ideas on how to solve it.

You cannot ask staff about the assessment or expect any help on those activities. The assessments are a measure of your understanding and acquired skill of the course.

You cannot copy code from the Internet unless you make specific acknowledgement of that work.

If it is an assessment and the work you copy is not your own, then you will be referred as a case for academic dishonesty

Get yourself ready for week 2

Reading: Section 1.1, 1.2 and optional 1.3 - 1.6 of Bryant & O'Hallaron Computer Systems: A programmer's perspective. Third Edition, Randal E. Bryant and David R. O'Hallaron. 2016. ISBN 9781292101767

Lab: Available from Friday

Seminar: Although optional, did you see it? available on canvas

Online programming problems: non assessable practice problems in C. available on Ed

Don't wait for it, go for it!