Aims:
The aim for this module is to have our advanced undergraduates develop a “whole-stack” understanding of the complex interactions among modern computer system hardware (specifically, a modern, multi-core CPU and its memory hierarchy), a modern operating system, and the applications that run atop them. These interactions dictate application correctness and performance. These fundamental principles will be taught in the concrete framework of the x86 CPU architecture and the Linux/UNIX operating system. Students should not only understand these fundamental principles, but be able to put them into practice in real C programs.

Learning outcomes:
On successful completion of the module, a student will be able to:

1. understand the C and Linux/UNIX programming environment, from the hardware (memory hierarchy, memory model between cores) to low-level operating system functionality (file and network I/O, process management, virtual memory system, program linking and loading);
2. understand the interactions between hardware and the C and Linux/UNIX programming environment that affect application correctness and performance;
3. design correct applications that perform well in the C and Linux/UNIX programming environment.

Content:
This module teaches students topics in computer systems design and systems programming, with an emphasis on the C programming language, Intel CPU architecture, and UNIX/Linux operating system architecture. Topics covered will include:

- Machine-level representation of code and data (assembly language; how C constructs are compiled
into x86 assembly language; machine-level representations of data types and machine and C arithmetic; stack frame structure in C on the x86; array layout in memory; pointer arithmetic);

- Use of the gdb debugger on C code in the Linux/UNIX environment;

- Undefined behaviour in the C language specification and real C programs, and its consequences;

- The memory hierarchy: registers, L1 caches, L2 caches, L3 caches, DRAM, SSD, disk; cache architectures; performance implications of caches on program performance;

- Linking: static linking, relocatable object code, symbol resolution, loading executables, dynamic linking and shared libraries, position-independent code, interposition;

- Processes, signals, context switches, exceptions, system call error handling, job control, process creation and management;

- Virtual memory: address translation, memory mapping, dynamic memory allocation, memory-related bugs in C;

- System I/O: filesystem I/O, system call interface and semantics;

- Concurrent systems programming: threads in C on Linux/UNIX, x86 memory consistency model, races, deadlocks.

**Requisites:**
In order to be eligible to select this module, a student must be registered on a programme for which it is a formally-approved option or elective choice AND must already have basic proficiency in C programming (including dynamic memory allocation), as well as basic knowledge of computer arithmetic (e.g., two’s complement integer representation) and prior exposure to assembly language for some CPU. UCL CS undergraduates will normally have met these prerequisites by having passed the relevant compulsory modules in prior years.