

Module 5: Concurrent and Parallel Programming

Stage	1					
Semester	2					
Module Title	Concurrent and Parallel Programming					
Module Number	5					
Module Status	Mandatory					
Module ECTS Credits	10					
Module NFQ level	9					
Pre-Requisite Module Titles	None					
Co-Requisite Module Titles	Algorithms Design and Analysis					
Capstone Module?	No					
List of Module Teaching Personnel	Mr. Tony Mullins					
Contact Hours				Non-contact Hours		
78				122		
Lecture	Practical	Tutorial	Seminar	Assignment	Placement	Independent Work
30	48			50		72
Allocation of Marks (Within the Module)						
	Continuous Assessment	Project	Practical	Final Examination	Total	
Percentage Contribution	50			50	100	

Intended Module Learning Outcomes

Upon successful completion of this module, learners will be able to:

1. write concurrent and parallel algorithms
2. use a parallel model to map algorithms to an underlying architecture - possible models would be C, C++, Ada tasks, Java threads, OpenMP, Posix threads, Scala actors(agents), F# agents(actors);
3. explain how multi processor, multi core architectures work;
4. explain race conditions and deadlocks;
5. describe the role of semaphores and events in concurrent systems;
6. solve problems requiring both semaphores and events as part of the solution;
7. explain the difference between state based concurrent systems and concurrent systems based on immutable state;
8. write solutions to concurrent & parallel problems using actor based message passing;
9. describe the difference between the shared memory model for threads and the distributed memory model for processes;
10. implement different server architectures (topologies), e.g. client server, peer-to-peer, agent systems, grid architectures;
11. use parallelism to optimise performance of algorithms

Module Objectives

The future of microprocessor development is based around multi-processor multi-core architectures that will deliver the performance required for future application demands. The difficulty for software developers is how to write programs that harness the power of these new architectures. As a result the fundamental aim of this module is to teach learners how to write software for these machines. The general module aims are to provide learners with an understanding of the need for, and advantages of, concurrent and parallel systems; to master a new programming paradigm that is different from that of the single threaded one; a description of how processes and threads are managed in multi-processor, multi core machines; an understanding and mastery of the many classical problems arising with concurrent and parallel tasks; an awareness of the need for such issues as fairness, process synchronisation, deadlock avoidance, etc.; the ability to write concurrent and parallel programs to solve real world problems; an understanding of multi-core architectures and their significance for the implementation of parallel systems; a mastery of notations to express solutions to parallel problems.

Module Curriculum

Topic	Description
Fundamentals	Multi core, multi processor systems; Hardware Architectures; Motivation for concurrency and parallelism; simple examples; advantages; disadvantages Process versus threads; priority of processes; process creation and destruction (Fork in Unix, Task in Ada, thread in java); processes sharing memory; dynamic process creation; distributed memory; facilities for concurrency and parallelism provided by programming languages and operating systems; C, C++, Ada tasks, Java threads, OpenMP, MPI, Erlang, Posix threads, Scala, F#, Windows, Linux and Unix.
Resource Sharing	Mutual exclusion; semaphores; fairness; deadlock; starvation; monitors; protected objects; condition variables; various kinds of shareable resources, e.g. memory, files, printers, etc; degrees of sharing, e.g. grab whole file or grab a single record; deadlock prevention. Classic problems: readers/writers, producer/consumer, bounded buffer. General problems requiring concurrent and parallel solutions e.g. lift control, telephone exchange, etc. Strategies for allocating resources; fairness; resource allocation algorithms. Necessity for scheduling algorithms, thread priority, resource allocation problems.
Parallelism	Task parallelism, data parallelism; Parallel algorithms from many areas, including matrix algorithms, graph algorithms, solution to linear equations, searching and sorting, image processing, encryption. Parallel models – mapping parallel algorithms to underlying architecture using e.g. C, C++, Ada tasks, Java threads, OpenMP, Erlang, Posix threads, Scala, F#.
Functional Programming and Concurrency	Message passing systems based on Actors (Scala, F# and Erlang) Avoiding race conditions with the use of immutable state. Communication protocols for actors. Programming with Actors.
Communicating Processes (processes without shared memory)	Distributed memory model; Pipes; channels; message passing; remote procedure call; process identities; multi-casting – broadcast to multiple processes.
Server Architectures	Client server architecture, peer-to-peer architecture, grid computing. Deploying services over these architectures. Developing an Agent based architecture.

Reading Lists and other learning materials

Recommended reading

Concurrent and Parallel Programming.	Mullins	Griffith College Dublin	2012
Concurrent Programming in Java	Doug	Sun	2000
Java Concurrency in Practice	Goetz, et al	Addison-Wesley	2006
The Art of Parallel Programming	Lester	1st World Publishing	2006
The Art of Multi Processor Programming	Herlihy, Shavit	Morgan Kaufmann	2007
Parallel Programming in C with MPI and OpenMP	Quinn	McGraw-Hill	2004

Secondary Reading

Concurrency in Ada	Burns, Wellings	Cambridge University Press	1995
Concurrency: State Models and Java	Magee, Kramer	Wiley	1999
Concurrent Programming	Burns, Davies	Addison-Wesley	1993
Java Thread Programming	Hyde	Sams	1999
Multithreading Applications in Win32	Beveridge, Wiener	Addison-Wesley	1997
Parallel Programming in OpenMP	Chandra, Menon, et al	Morgan Kaufman	2000
Programming Paradigms with Scala	Mullins	Griffith College Dublin	2012
Programming in Scala	Odersky, Spoon, Venners	Artima Press	2010

Module Learning Environment

Accommodation

Lectures are carried out in class rooms / lecture halls in the College. Lab tutorials are carried out in computer labs throughout the Campus. All have the language software required to deliver the programme.

Library

All learners have access to an extensive range of physical and electronic (remotely accessible) library resources. The library monitors and updates its resources on an on-going basis, in line with the College's Library Acquisition Policy. Lecturers update reading lists for this course on an annual basis as is the norm with all courses run by Griffith College.

Module Teaching and Learning Strategy

The module is delivered through a combination of lectures and practical lab programming sessions. The learners complete a series of worksheets throughout the module that are directly related to the material covered in lectures. The emphasis is on developing sound software engineering skills in practical programming based on theoretical knowledge.

Module Assessment Strategy

The module assessment consists of a series of continuous assignments and a final examination. Each week learners are required to complete a series of programming tasks that relate to the material covered in lectures. The practical lab sessions are used to enforce concepts covered in the lectures and the worksheets are used to ensure that learners are keeping up with the material as it is delivered. Lab sessions are also used to deal with issues emerging from the worksheets. All work submitted by learners is assessed and comments are given to individual learners. All assessment goes to giving a final grade for the work completed by the learner.

Learners are assessed on the following, completed during the course.

Element No.	Weighting	Type	Description	Learning Outcomes Assessed
1	10%	Assignment	Programming problems that involve use of threads and parallel processing	1,2,11
2	10%	Assignment	Programming problems that involve race conditions and that require the use of condition variables to solve	1,2,4
3	10%	Assignment	Programming problems that may be solved using semaphors	1,2,4,6
4	15%	Assignment	Distributed processing and client server architecture	1,2,4,6,,9,10
5	5%	Assignment	Concurrent systems based on immutable state and solution to problems using actor based model of concurrency	1,2,7,8
6	50%	Examination	Questions on the exam paper are drawn from all aspects of the course and cover all learning outcomes.	1,3,4,5,6,7,8,9,10,11

			The paper consists of 6 questions and learners are required to answer any 5. All questions carry equal marks.	
--	--	--	---	--