THE GLOBAL EDUCATION NETWORK

DBLN CPSI 4492

This course is being offered at Griffith College, CAPA's academic partner in Dublin. The Irish academic system differs from the US, particularly with grading. Griffith College professors expect students to undertake a good deal of independent study to achieve a high mark in their classes. For additional information about this class, please contact the Boston Program Advising Team at 1-800-793-0334.

Concurrent Programming

Continuous Assessment: 60%
Exam: 40%

Intended Module Learning Outcomes

On successful completion of this module learners will be able to:

1. analyse the advantages of concurrent programming
2. explain the necessity for synchronisation when sharing resources
3. solve problems that require synchronisation
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. describe the problems associated with resource allocation
8. describe the difference between the shared memory model for threads and the distributed memory model for processes
9. describe the client server architecture

10. write concurrent programs and client/server applications in a high-level language such as Java, C#, C or Ada

Module Objectives

This is the final programming module on the B.Sc. honours degree and it builds on the work completed in all three years: Programming, Object-Oriented Programming, Data Structures and Algorithms and Programming Paradigms. It gives a comprehensive analysis of the Concurrent Paradigm that is now central to the world of programming. Concurrency is central because the underlying hardware is now multi-core and to exploit this power it is necessary to have an in-depth understanding of concurrent concepts. Concurrent programming is not new – it has been there from the very beginning, particularly, in the field of operating systems. However, it is new to the world of every day professional programmers and it is essential that our graduates understand both its strengths and its limitations. It is a very difficult field of programming because processes are non-deterministic and to write concurrent systems that are correct we must understand how to manage limited shared resources. Therefore, a key objective, once again, is the acquisition, on behalf of the learner, of good software engineering skills and the application of these skills to the design and implementation of concurrent software components. Learners gain an understanding of the need for, and advantages of, concurrent and parallel systems; a mastery of 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; an ability to write concurrent and parallel programs to solve real world problems.

Module Curriculum

Fundamentals

- Motivation for concurrency;
- simple examples; advantages;
- disadvantages

Idea of a process

- 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;
- facilities for concurrency provided by programming languages and operating systems;

- C# , Windows, Linux and Unix; Java virtual machine; writing threads in Java

Resource sharing

- Mutual exclusion;
- semaphores;
- fairness;
- deadlock;
- starvation;
- monitors;
- protected objects;
- condition variables;
- various kinds of shareable resources,
- degrees of sharing
- deadlock prevention.
- Classic problems: readers/writers, producer/consumer, bounded buffer.
- General problems requiring concurrent solutions, e.g. lift control, train control, etc.

Allocating resources and scheduling

- Strategies for allocating resources; fairness;
- resource allocation algorithms.
- Necessity for scheduling algorithms and thread priority.

Communicating processes (processes without shared memory)

- Distributed memory model; Pipes; channels; message passing; remote procedure call.
- Distributed Topologies
- Server design and the implementation of client server architecture over sockets.
- Deploying services on a client server architecture.
- Supporting distributed processing using server model.

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.