

MODULE: CONCURRENT PROGRAMMING (course code BSCH-CPG)

Griffith College Dublin – Computing Science

Fall semester

This module is intended for senior level students who are majoring in this field.

Aims

This module aims to provide you with an understanding of the need for, and advantages of, concurrent systems; to master a new programming paradigm that is different from that of the single threaded one; how multiple competing processes are managed by underlying operating systems such as Unix, Linux, Windows NT and Window2000; a description of how processes and threads are created in Java; a description and understanding of the many classical problems arising with concurrent tasks; an awareness of the need for such issues as fairness, process synchronisation, deadlock avoidance, etc; the ability to write concurrent programs to solve real world problems; an understanding of multi-core architectures and their significance for the implementation of concurrent systems, an introduction to server design using the client server model.

Learning Outcomes

Upon successful completion of this module, you should 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

Indicative Content

| Topic | Description |
|--------------|--|
| 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, 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 solutions, e.g. lift control, train control, etc. |
| Allocating resources and scheduling | Strategies for allocating resources; fairness; resource allocation algorithms. Necessity for scheduling algorithms, thread priority, writing a round robin scheduler using dynamic allocation of priority values. |
| Communicating processes (processes without shared memory) | Distributed memory model; Pipes; channels; message passing; remote procedure call; process identities; multi-casting – broadcast to multiple processes; problems of deadlock; synchronisation. |
| Applications of concurrency | Server design and the implementation of client server architecture over sockets. Deploying services on a client server architecture. |

Assessment Methods

To assess this module we will use two different modes: workbooks and a formal closed book examination. There will be two workbooks each worth 20%. Each workbook will consist of a series of problems related to the material covered in the lectures and tutorials. Students will be required to solve a range of problems and implement their solutions in a high level language (Java, C#, C, Ada). The examination paper will consist of six questions and will have a weighting of 60%. Each question will carry a mark of 20. Students will be expected to complete 5 of these questions.