CS375: Concurrent Programming

◇ Who am I? ◇

James Heather

Email: `jheather@dcs.rhbnc.ac.uk`

My office is McCrea 112, but I am in only on Thursdays. If you want to talk to me, you should email me to arrange to see me on a Thursday, or collar me after a lecture.

I'm just about to submit my PhD thesis on security protocol analysis, with CSP—the essence of this course—at the heart of the research.

◇ What's it all about? ◇

Concurrent systems — made up of independent but communicating components — are all around us. Familiar examples include:

- ◇ The network of bank cash machines
- ◇ The internet
- ◇ The network of "Switch" machines
- ◇ The components of a PC
- ◇ The telephone system

Understanding, designing and building concurrent systems is a major challenge for computer science. The problems involved are in a different league from the problems of sequential programming, and a systematic approach is essential.

This course aims to equip you with some of the theory, tools and techniques needed to understand and analyse concurrent systems, and to enable you to take a systematic approach to designing your own.

◇ What will we do? ◇

Despite the title, this course does not involve large amounts of programming in the usual sense — it's not like CS220, for example. We'll be concentrating on techniques for modelling, analysing and understanding concurrent systems. We will begin by looking at some concurrent programs, experimenting with them, and understanding why they do or do not work correctly. Then we will learn CSP, which is a theoretical notation or language for modelling concurrent systems. CSP is supported by various software tools which enable systems to be analysed and debugged, and we will use two in particular — ProBE and FDR — to assist in learning CSP and also to perform analyses of the systems which we consider.

When we look at concrete programs, they will be written in Pascal-FC, a language which is convenient for teaching as it supports a range of concurrent programming features and styles. Pascal-FC runs on the departmental server `tartan`, and is also available for PC systems. Previous knowledge of Pascal is not necessary, as it is easy to acquire a reading knowledge of Pascal-FC, and the Pascal-FC programming which we do will be limited to relatively small modifications of existing programs.

◇ Aims ◇

- ◇ To give an appreciation of the range of applications of concurrent and distributed systems, and the benefits of concurrency.
- ◇ To demonstrate that concurrent systems are complex and that systematic techniques are needed for analyzing and reasoning about them.
- ◇ To introduce CSP as a theory of concurrent systems and as a framework for specification and analysis.
- ◇ To use the CSP tools ProBE and FDR to gain practical experience of modelling and analysing concurrent systems in CSP.

◇ Programme ◇

There are three timetabled hours of teaching per week.

Day	Time	Place	Type
Thursday	11:00–12:00	?	Lecture
Thursday	12:00–13:00	?	Lab session
Thursday	14:00–15:00	?	Lecture

I will often ask questions during lectures, so stay awake! Also, you are encouraged to ask questions as we go along. If there's something you don't understand, it's almost certain that several other students also don't understand it, so you will be doing everyone a service by asking.

Copies of the lecture slides for each lecture will be distributed at the beginning of the lecture.

◇ Assessment ◇

- ◇ 80% by exam, 20% by coursework (4 courseworks, 5% each)
- ◇ I'll tell you the dates for setting and handing in of coursework next week.
- ◇ If you want comments on your coursework, hand in two copies.

◇ Books and Software ◇

The following books are recommended:

- ◇ *Concurrent and Real Time Systems: The CSP Approach*, S. Schneider, Wiley, ISBN 0-471-62373-3
- ◇ *Communicating Sequential Processes*, C. A. R. Hoare, Prentice-Hall, ISBN 0-13-153289-8
- ◇ *Theory and Practice of Concurrency*, A. W. Roscoe, Prentice-Hall, ISBN 0-13-674409-5

All of them should be in the campus bookshop. You need at most one, and in fact you should be able to get by with just the course notes.

The book on CSP by Hinchey and Jarvis is **not** recommended, as it contains many serious technical errors.

Pascal-FC, ProBE and FDR are installed on the departmental server `tartan`. There is also an implementation of Pascal-FC for PCs, which I will put in the ftp area. Several example programs and some documentation are included with the distribution.

◇ The Ornamental Gardens ◇

We will use the following scenario, taken from the book "Concurrent Programming" by Burns & Davies, to illustrate some problems in concurrent programming.

An ornamental garden is open to the public, and entry is controlled by two turnstiles. We are required to produce a computer system which will count the number of visitors entering the garden.

We will attempt to produce a solution to this problem, with the following features.

- ◇ The program will consist of two independent parts, or *processes*. Each process will handle one of the turnstiles. The two processes will run *concurrently*, i.e. simultaneously or *in parallel*.
- ◇ A global variable, which both turnstiles can access, will record the number of visitors.

We will suppose that 20 visitors enter through each turnstile, and just think about the overall counting.

The program is written in Pascal-FC.


```
program gardens1;

var
  count: integer;

process turnstile1;
var
  loop: integer;
begin
  for loop := 1 to 20 do
    count := count + 1
end;  (* turnstile1 *)

process turnstile2;
var
  loop: integer;
begin
  for loop := 1 to 20 do
    count := count + 1
end;  (* turnstile2 *)

begin
  count := 0;
  cobegin
    turnstile1;
    turnstile2
  coend;
  writeln('Total admitted: ',count)
end.
```

◇ Processes in Pascal-FC ◇

The unit of execution in the Pascal-FC system is the *program*. Within one program, it is possible to activate several processes in parallel. The Pascal-FC run-time system deals with communication between these processes. Here is an outline of a typical Pascal-FC program:

```
program OUTLINE;

process P;
begin
  ...
end;

process Q;
begin
  ...
end;

begin
  cobegin
    P;
    Q
  coend
end.
```


Points to note:

- ◇ `cobegin P; Q coend` causes the processes `P` and `Q` to be executed in parallel.
- ◇ The order of the processes within `cobegin ... coend` is unimportant.
- ◇ `cobegin ... coend` cannot be nested. The parallel processes must be declared using `process` — they cannot be arbitrary statements.
- ◇ There can be a sequential series of statements before or after `cobegin ... coend`. There can be several `cobegin ... coend` blocks in sequence (or indeed none at all).

We are not going to study the Pascal-FC language comprehensively; instead, we will plunge in and look at examples, which can then be copied and modified. Language features will be discussed as necessary.

A few points about Pascal syntax, for now:

- ◇ Slightly more verbose than C++. For example, `begin ... end` around blocks instead of {...}.
- ◇ Variables are declared as in:

```
var
   i : integer;
   x : array [1..10] of real;
```

◇ In Pascal-FC, identifiers are case-insensitive.

◇ There is a distinction between functions, which return results, and procedures, which do not. For example:

```
procedure MyProc(x:integer);
begin
  ...
end;


function MyFun(x:integer):boolean;
begin
  ...
  return(true);
end;
```

◇ The syntax for conditionals, `for` loops, `while` loops, `repeat` loops etc. is different from C++, and more verbose.

◇ Examples ◇

Pascal-FC:

```
function Factorial(n:integer):integer;
var
  i,p:integer;
begin
  p := 1;
  for i := 1 to n do
    p := p * i;
  return(p);
end;
```

C++:

```
int Factorial(int n)
{
  int p = 1;

  for (int i = 1; i <= n; i++)
    p = p * i;

  return(p);
}
```

Pascal-FC:

```
procedure PrintFactorial(n:integer);
var
  p:integer;
begin
  p := 1;
  while n > 0 do
    begin
      p := p * n;
      n := n - 1
    end;
  writeln(p);
end;
```

C++:

```
void PrintFactorial(int n)
{
  int p = 1;

  while (n > 0)
  {
    p = p * n;
    n = n - 1;
  }

  cout << p << "\n";
}
```


Pascal-FC:

```
procedure PrintFactorial(n:integer);
var
  p:integer;
begin
  p := 1;
  repeat
    p := p * n;
    n := n - 1
  until n = 0;
  writeln(p);
end;
```

C++:

```
void PrintFactorial(int n)
{
  int p = 1;

  do
  {
    p = p * n;
    n = n - 1;
  }
  while (n > 0);

  cout << p << "\n";
}
```

Pascal-FC source files are normally called `something.pfc` and they are compiled and executed by

```
pfc program.pfc
```

There is no separate compilation phase — Pascal-FC programs are compiled into an intermediate form and then interpreted.

The `pfc` command is `/usr/local/pasfc/bin/pfc` so you will probably want to add

```
export PATH=$PATH:/usr/local/pasfc/bin
```

to your `.bashrc` file.

There is some online documentation in the directory

```
/CS/ftp/pub/CS375/PascalFCdocumentation
```

The language reference manual is

```
lrm.ps
```

and the user guide for the system is

```
sun_ug.ps
```

To view either of these documents, type (e.g.)

```
ghostview lrm.ps
```

from within the correct directory.

The book "Concurrent Programming" by G. Burns and A. Davies (published by Addison Wesley) describes Pascal-FC and covers a range of topics in concurrent programming, including some CSP. Most of the programs discussed in Burns & Davies can be found in

```
/usr/local/pasfc/ex
```

but you will need to copy them into your own directory before running them.

There is a very simple Pascal-FC program in

```
/CS/ftp/pub/CS375/test.pfc
```

which you can use to check that you are able to use the Pascal-FC system. The first practical class takes you through the process of logging in to `tartan` and running the test program.

◇ Heavyweight and Lightweight Processes ◇

The Unix operating system allows a number of processes to be executed concurrently; more precisely, it gives the appearance of concurrency by switching between processes very frequently. There is quite a large overhead associated with switching between Unix processes: in particular, each process has its own virtual address space, so when execution switches between processes any memory being used by the old process has to be paged out and replaced by the memory image which the new process left when it stopped executing previously. For this reason, Unix processes are described as *heavyweight*.

A Pascal-FC program is executed by a single Unix process, regardless of how many processes are declared and executed within the Pascal-FC program. The Pascal-FC system is responsible for switching between the processes in a `cobegin ... coend` block. Pascal-FC processes are described as *lightweight*, because there is far less overhead involved in controlling them. Lightweight processes are also known as *threads*: the idea is that there may be many "threads of control" within a single Pascal-FC execution.

Threads are also available in Java.