Operating Systems: Project #1

Introduction

This assignment will focus in the following tasks:

- Creation and management of processes;
- Inter-process communication using pipes;
- Asynchronous signals;
- I/O multiplexing by using `select()`.

Processes

A process is an instance of a program that is executing. In this assignment, the student will learn how to create processes and how to understand the UNIX process model.

Signals

A signal is a software notification of an event sent by the operating system to a process. Signals can be used by the operating system to report exceptions to running processes, such as the usage of an invalid memory reference or the occurrence of an error while accessing a file. Signals may also be used between processes to notify some particular conditions. To guarantee the portability of an application, signals should be referred by their name. When you use the signal number it might not be the same on different operating systems. Every signal has a custom default treatment preset by the operating system. As an example, the signal `SIGTERM` will force the termination of a process. By using a signal handler, a process can override the default action associated with a signal, which is normally the termination of the process. There is only one signal (`SIGKILL`) that cannot be caught by a signal handler.

Pipes

Pipes are the simplest mechanism offered by the operating system for inter-process communication. A pipe is a communication buffer between two processes: it has two descriptors, one for writing another for reading. Write and read operations are done in a FIFO order (first-in-first-out).

There are two kinds of pipes: unnamed pipes and named pipes (also known as FIFOs).

- **Unnamed pipes** only allow communication between hierarchically related processes (parent and child processes);
- **Named pipes** allow the communication between any process. A special file is created in the file-system through which processes can write or read data. Named pipes are persistent over time.

I/O Multiplexing

When a process needs to handle multiple I/O events it should make use of the `select()` routine. This system call allows a process to block reading in a set of file-descriptors: when one of those descriptors has something to read, the process is unblocked and conducts the read operation.

Objectives

At the end of this Project students should be able to:

- Create and manage processes in Unix;
- Cope with signal management;
- Master the communication between processes using pipes and named-pipes;
- Know how to use `select()` to implement I/O multiplexing.

Support Material

- K. A. Robbins, S. Robbins, "*Unix Systems Programming: Communication, Concurrency, and Threads*", Prentice Hall:
  - Chapter 2 - *Programs, Processes and Threads*
  - Chapter 3 - *Processes in UNIX*
  - Chapter 4 – *"The select Function"* (section 4.4)
  - Chapter 6 – *"Unix Special Files"*(section 6.1 to 6.4)
  - Chapter 8 – *"Signals"*

- "*Programming in C and Unix*", available at

  http://gd.tuwien.ac.at/languages/c/programming-dmarshall/:

  - *Process Control: <stdlib.h>,<unistd.h>*
  - *Interprocess Communication (IPC): Pipes*
  - *IPC:Interrupts and Signals:*<signal.h>

- Online Unix manual
  - Manual page of signal().
  - Manual page of pipe().
  - Manual page of select().

- Course materials:
  - Notes on Unix basic commands
  - Notes on reading and writing to files
  - Notes on Makefiles

Project Assignment

Overview

The objective of this assignment is to provide an approximation of the value of π by using a Monte Carlo method.

The Monte Carlo methods were initially proposed by John von Neumann, Stanislaw Ulam and Nicholas Metropolis in the 1940s, while the authors worked at the Los Alamos National Laboratory in America's nuclear weapons project (Manhattan Project).

Monte Carlo methods are often used in the simulation of physical and mathematical systems, or whenever it is difficult to reach an exact result by using a deterministic algorithm. The core concept behind Monte Carlo methods is the idea that an approximate result for a problem can be reached by repeatedly random sampling values inside a region, performing a computation over those inputs and aggregating the results.

Computers are the best tool for calculating Monte Carlo approximations. Furthermore, these algorithms can be easily parallelized, thus allowing them to explore the full potential of modern multi-core architectures.

Your task for this assignment is to write a concurrent program for calculating an approximation of the value of π using a Monte Carlo Method that is able to outperform, in execution time, the sequential version that we will be provided.

Figure 1 gives an idea of how value of π can be calculated by using a Monte Carlo method:

1. First, consider that you have a square and you have inscribed a circle inside the square;
2. Then generate a finite number of random points inside the square and count how many are inside the circle and how many are outside;
3. The ratio of the two counts is an estimate of the ratio of the two areas (given by π/4);
4. Last, to get the value of π multiply the previous value by 4.

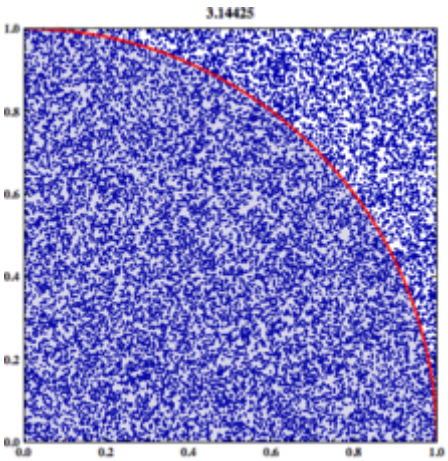

Figure 1 - Monte Carlo method applied to approximating the value of π (source: Wikipedia.org)

To make things more exciting, we propose a **challenge**: your program should be able to outperform the sequential version when executing in a computer with a multi-core processor. In the following section we explain how the value of the observed speedup can be calculated.

Speedup

Speedup is a metric defined by the following formula:

$$S_P = \frac{T_1}{T_P}$$

Where S_P is the speedup obtained when executing the program in a system with P cores or CPUs, T_1 is the execution time of the sequential version of the program, a T_P is the execution time of the concurrent program in a machine with P cores or CPUs.

Quality attributes

Your application must make use of:

(a) multiple processes;

(b) signals to communicate events between processes;

(c) pipes to communicate data between processes;

(d) `select()` routine to do read/write multiplexing in multiple pipes.

To fulfil the objectives of this assignment you should be as creative as you want, provided you have included this list of OS features in your solution.

Notes

- Do not start coding right away. Take enough time to think about the problem and to structure your design;
- Implement the necessary code for **error detection and correction**;
- **Avoid busy-waiting** in your code!!
- Assure a **clean shutdown** of your system. Release all the used resources and **stop all processes** before exiting the Master process;
- Always release the OS resources that are not being used by your application;
- **Plagiarism or any other kind of fraud will not be tolerated**. Attempts of fraud will result in a ZERO and consequent failure in the OS course.

Submission instructions

1. The names and student ids of the elements of the group must be placed at the top of the **source code file(s)**. Include also the time spent in this assignment (sum of the time spent by the two elements of the group).
2. Any external libraries must be provided with the source code file, including the usage instructions. Use a **ZIP** file to archive multiple files.
3. You must provide a short **REPORT** explaining the design and implementation of your solution and justifying the design options that you have made.
4. The project should be submitted at http://moodle.dei.uc.pt.
5. The enrolment key for moodle is **so_2011**
6. The submission deadline is **October 18, 2011 (23h55)**
7. Finally, go to the Google spreadsheet that was created for this assignment and insert the value of the **speedup** that you did obtain. Do not forget to specify the CPU that was used. **Link:**
   https://docs.google.com/spreadsheet/ccc?key=0AszqyxPh8hlzdG4xQlJ6bzlqWENTRUJoZEVvX1JUbnc&hl=en_US