Operating Systems: Project #2

Introduction

This assignment will focus in the following topics:

- Shared Memory, as a mean for sharing data between processes;
- Memory Mapped Files, as a way to manipulate files on disk and also share data between collaborative processes;
- POSIX Semaphores, for controlling accesses to shared resources;

Objectives

At the end of this project, students should be able to:

- Create and use a segment of shared memory to share data between processes;
- Map a file into a buffer in memory and use it to share data between processes;
- Use the library `<semaphore.h>` to create and use semaphores to synchronize the access to shared resources between concurrent processes.

Support Material

- K. A. Robbins, S. Robbins, "Unix Systems Programming: Communication, Concurrency, and Threads", Prentice Hall:
  - *Chapter 14 – "Critical Sections and Semaphores"*
  - *Chapter 15 – "POSIX IPC" (secções 15.1 a 15.3)*
- W. Richard Stevens, Stephen A. Rago, "Advanced Programming in the UNIX® Environment: Second Edition", Addison Wesley, 2005
  - 14.9 – "Memory-Mapped I/O"
- "Programming in C and Unix", available at

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

  - IPC:Semaphores
    - POSIX Semaphores: `<semaphore.h>`
  - IPC:Shared Memory
  - IPC:Shared Memory - Mapped memory
- Online Unix manual
  - Manual pages of `sem_post`, `sem_wait`, `sem_open`, `sem_init`.
  - Manual pages of `fork`, `exec`.
  - Manual pages of `shmget`, `shmctl`, `shmat`.
  - Manual pages of `mmap`, `munmap`
- Course materials:
  - Notes on Unix basic commands;Notes on reading and writing to files

Project Description

The goal of this project is to **build a parallel program for calculating the heat (temperature) variation on a surface along the time**. For doing so, your program must use the ***Heat Equation***. The heat equation describes the temperature change over time, given initial temperature distribution and boundary conditions. Your program **must use a set of collaborative processes** to calculate the heat equation results for all points on a surface in chronological time steps.

Input

The input for your program is a `.csv` file, which we will be provided, containing the initial heat distribution for all the points of a metallic square surface. The file can be interpreted as a 2-dimensional array representing the temperature at different points on the square. Figure 1 illustrates the initial distribution of heat along the target surface using colours. Considering that the boundary temperature is held at zero, the initial temperature is zero on the boundaries and very high in the middle.

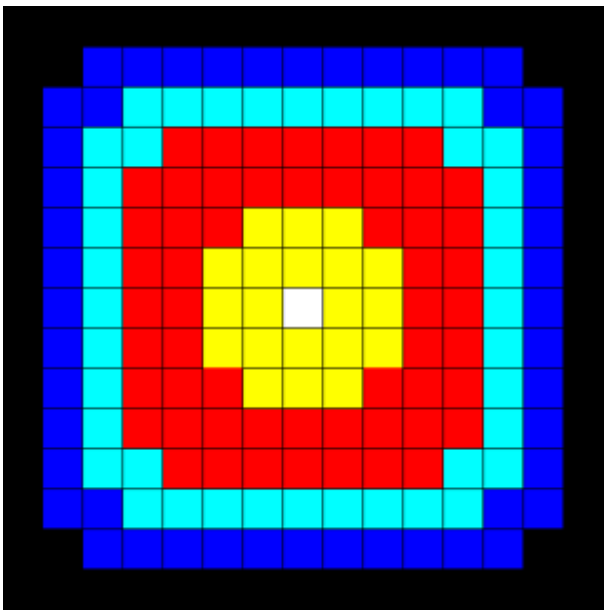

Figure 1 - Temperature variation in a square metallic surface

(©Blaise Barney, Lawrence Livermore National Laboratory)

Output

Your program must produce 4 `.csv` files as output using the same structure of the input file:

1. `firstQuarter.csv` will contain the values for the heat distribution when ¼ of the time steps are elapsed;
2. `halftime.csv` will contain the values for the heat distribution when½ of the time steps are elapsed;
3. `thirdQuarter.csv` will contain the values for the heat distribution after ¾ of the time steps;
4. `final.csv` will contain the values for the heat distribution after all steps have been performed.

Heat Equation sequential algorithm

A fundamental rule for applying the Heat Equation is that the calculation of the temperature of an element is dependent upon its neighbor elements values. This concept is described by the following formula and Figure 2:

$$U_{x,y} = U_{x,y} + C_x\left(U_{x+1,y} + U_{x-1,y} - 2U_{x,y}\right) + C_y\left(U_{x,y+1} + U_{x,y-1} - 2U_{x,y}\right)$$

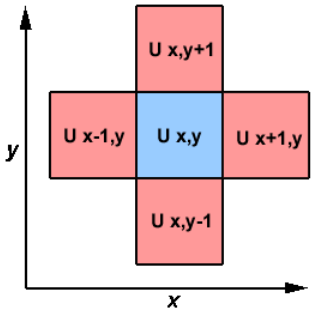

Figure 2 - Position of elements used for each calculation (©Blaise Barney, Lawrence Livermore National Laboratory)

Consider that this formula must be applied to each point on the surface on each chronological step. A serial program could contain the following code:

```
cx = cy = 0.1;
n_steps = 0;
while (n_steps < TOTAL_STEPS) {
  for (int y = 1; y < SQUARE_DIMENSION - 1; y++)
    for (int x = 1; x <  SQUARE_DIMENSION - 1; x++)
    new_matrix [x][y] = matrix [x][y] +
              cx * (matrix [x+1][y] + matrix [x-1][y] - 2 * matrix [x][y]) +
              cy * (matrix [x][y+1] + matrix [x][y-1] - 2 * matrix [x][y]);
  ++nsteps;
  tmp = matrix; matrix = new_matrix; new_matrix = tmp;
}
```

Noteworthy information

i. The surface under analysis is **square**;
ii. Boundary temperature of the square surface is **zero** (0ºC);
iii. The points with the **highest** temperatures can located at the **centre** of the surface, but there can other spots on the surface with equally high temperatures;
iv. $C_x = C_y = 0.1$
v. Your program must execute at least **100 time steps**, but this value should grow if the total execution time of the program is very short;
vi. You should keep track of the execution time of your program for calculating the speedup;
vii. The provided input file **cannot be modified**.

Speedup

In this project, as we have done on the previous one, the speedup obtained by parallelizing the solution is very important, thus you should:

1. **start by writing a sequential** version of the program based on the above code;
2. use the command `time` to **get the total execution** time of this initial version;
3. **build a parallel** version, get its execution time;
4. calculate the observed **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) Shared Memory;

(c) Memory Mapped Files;

(d) POSIX semaphores.

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

- **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.
- 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;
- Use a `Makefile` for simplifying the compilation process;
- Be sure to have some debug information that can help us to see what is happening.

Example:

```
#define DEBUG  //remove this line to remove debug messages
(...)
#ifdef DEBUG
  printf("Creating shared memory\n");
#endif
```

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 submission deadline is **November 15, 2011 (23:55).**