1

Ejercicios de Programación de Sistemas Distribuidos

Generación de números primos

La generación de números primos es un proceso que requiere una capacidad considerable de cómputo. Para aliviar esta tarea se propone realizar un programa, escrito C y utilizando MPI, que explote las diferentes CPUs de un sistema distribuido.

El programa pedido deberá distinguir entre dos tipos de procesos: proceso *master* y procesos *worker*. En cada ejecución existirá un único proceso *master* y, al menos, dos procesos *worker*.

El proceso *master* debe contener un vector de números enteros donde se **almacenarán** los números primos. Estos números serán **generados, únicamente,** por los procesos *worker*. De esta forma, el proceso *master* se encargará de sincronizar las peticiones a los distintos procesos *worker* y de recibir los números generados. En cada ejecución del programa se deberán generar `VECTOR_LENGTH` números primos. Así, una vez el proceso *master* haya recibido `VECTOR_LENGTH` números primos, se finalizará la ejecución. Además, para repartir el trabajo a cada proceso *worker*, se obtendrá a través de la línea de comandos el tamaño de grano. Este parámetro indica, por cada petición del proceso *master*, la cantidad de números a generar por cada proceso *worker*. La ejecución por la línea de comandos se realiza de la siguiente forma:

```
mpiexec -hostfile machines -np numProc generatePrimes tamGrano
```

donde *machines* es el fichero con las direcciones IP de los ordenadores en los cuales se ejecutará el programa, *numProc* es el número de procesos que intervienen en la ejecución del programa, *generatePrimes* es el fichero ejecutable del programa pedido y *tamGrano* es un número entero que indica la cantidad máxima de números primos a generar por cada proceso *worker*, para enviarlos al proceso *master*.

Es importante destacar que en cada proceso *worker*, la diferencia entre los números generados y enviados al proceso *master* puede diferir, como máximo, en *tamGrano* números.

Consideraciones a tener en cuenta:

- El sistema distribuido donde se ejecutará el programa es heterogéneo. Es decir, los recursos de cada ordenador pueden ser diferentes.
- El tamaño de grano debe tener un valor entre 10 y 100.
- Suponed que, para todos los casos, `VECTOR_LENGTH > (numProc * tamGrano)`
- Es posible que no todos los procesos *worker* generen la misma cantidad de números primos, ya que esto depende del tamaño de grano y de la velocidad de CPU donde se ejecute cada proceso.
- La función *generatePrimeNumber* devuelve un número primo. Esta función se supone implementada, por lo cual no es necesario implementarla en este ejercicio.

La siguiente porción de código muestra parte del programa descrito. Se pide implementar las funciones *executeMaster* y *executeWorker* para que el programa cumpla los requisitos descritos en el enunciado.

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <fcntl.h>

/** Cantidad de números primos a generar */
#define VECTOR_LENGTH 100000

/** Proceso master */
#define MASTER 0

/** Flag que indica el fin de ejecución */
#define END_OF_PROCESSING -1


// Prototipos de funciones
int generatePrimeNumber ();
void executeMaster (int grainSize, int numProc);
void executeWorker (int grainSize);


int main(int argc, char *argv[]){

   int numProc, rank;
   int grain;

        // Init MPI
        MPI_Init (&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &numProc);


        // Comprobar argumentos de entrada
        if (argc != 2){
            printf ("Número incorrecto de argumentos\n");
            MPI_Abort (MPI_COMM_WORLD, 0);
        }

        // Comprobar número de procesos
        if (numProc < 3){
            printf ("Número incorrecto de procesos\n");
            MPI_Abort (MPI_COMM_WORLD, 0);
        }

        // Obtener tamaño de grano
        grain = atoi(argv[1]);

        if ((grain > 100) || (grain < 10)){
            printf ("Tamaño incorrecto de grano\n");
            MPI_Abort (MPI_COMM_WORLD, 0);
        }

        // Proceso master ;
        if (rank == MASTER){

            executeMaster (grain, numProc);
        }

        // Procesos worker
        else{
            executeWorker (grain);
        }
```