Monitoreo de Sensores Mediante Programación Paralela | Proyecto Final de Sistemas Operativos

Maria José
Cárdenas Machaca
Pontificia Universidad Javeriana
Bogotá, Colombia
mariacardenasm@javeriana.edu.co

Diego Alejandro
Albarracin Maldonado
Pontificia Universidad Javeriana
Bogotá, Colombia
di.albarracin@javeriana.edu.co

***Abstract*—Este informe presenta un avance inicial del proyecto de semestre del curso de Sistemas Operativos, el cual implementa temáticas vistas en clase como lo son: hilos, tuberías y semáforos. En esta primera entrega, se hará una contextualización del proyecto, explicación de los componentes que la conformar incluyendo código y, pruebas de ejecución que sustentan el correcto funcionamiento del programa.**

Keywords: Hilos, Pipes, Concurrencia, Paralelismo, .

I. Contextualización

El presente proyecto consiste en diseñar un sistema capaz de monitorear la calidad del agua; para ello, inicialmente, se pretende implementar dos sensores primordiales para esta tarea: sensor de temperatura y sensor de PH, además de poseer un monitor que sea capaz de procesar los datos arrojados por estos sensores y arrojar un mensaje final al consumidor.

Con el fin de hacer óptima la implementación de este programa, se hará la simulación de los datos de entrada de este sistema, construyendo dos archivos *.txt* (*ph.txt y temperatura.txt*) los cuales contendrá un conjunto de datos respectivamente para ser procesados por cada uno de los sensores. Asimismo, es importante mencionar que, dentro de la tarea de procesamiento de datos, cada sensor posee un rango de validación óptima que determina el nivel de calidad de agua.

Parámetro	Valor mínimo	Valor máximo
Temperatura	20°C	31,6°C
PH	6.0	8.0

Fig. 1. Rango de valores mínimos y máximos aceptados por el monitor.

Dentro de la elaboración de este proyecto, se manejarán dos procesos: sensores y monitor, por lo que es necesario el manejo del pipe nominal, el cual permitirá la comunicación entre los sensores y el monitor; a su vez, dentro del proceso monitor se hará manejo de tres hilos:

- **H-recolector:** Recibe las mediciones del pipe nominal.
- **H-ph:** Recoge las medidas colocadas en su buffer y las escribe en el archivo file-ph.
- **H-temperatura:** Recoge las medidas colocadas en su buffer y las escribe en el archivo file-temp.

Finalmente, en esta primera entrega, se incluirán el uso de semáforos, hilos y creación de procesos, además de rectificar el manejo de esto mediante la impresión de mensajes que confirmarán la recepción de datos por parte del proceso sensor hacia el proceso monitor, mensajes de la correcta aplicación de hilos y el buen almacenamiento de los datos en cada uno de los archivos.

II. Implementación

La implementación de este proyecto está dada mediante el lenguaje de C, donde se construirán dos clases: sensor.c y monitor.c, los cuales se explicarán a continuación.

Es importante mencionar que se agregaron dos archivos de texto: *temperatura.txt* y *ph.txt*, los cuales contienen datos de muestra para ser procesados por los sensores y el monitor.

A. Sensor

Es importante mencionar que se agregaron dos archivos de texto: *temperatura.txt* y *ph.txt*, los cuales contienen datos de muestra para ser procesados por los sensores y el monitor.

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
```

Inicialmente, es importante incluir las librerías en esta clase que, a nivel general proporcionan funciones para el manejo de errores, de archivos, de procesos, entre otros. Seguidamente, se definen las siguientes variables y estructuras:

- **BUFFERSIZE:** Determina el tamaño utilizado para la lectura de las mediciones de los sensores.
- **DatosSensor:** Contiene los datos que serán enviados al proceso monitor.

- **SensorArgumentos:** Contiene las variables que corresponden a los argumentos necesarios que debe recibir el proceso sensor.

```
// Tamao del buffer para la lectura del
    archivo
#define BUFFERSIZE 50
/*--- Estructura con los datos a enviar al
    monitor ---*/
typedef struct {
int tipo_sensor;
float medicion;
} DatosSensor;
/*--- Estructura con los argumentos para la
    simulacin del sensor ---*/
typedef struct {
int tipo_sensor; // Tipo de retorno:
    1-Temperatura, 2-PH
int tiempo;     // Tiempo en enviar la medicin
    del sensor al monitor
char *archivo; // Archivo con medidas de
    temperatura o PH
char *pipe_nominal; // Permite la comunicacin
    entre los procesos
} SensorArgumentos;
```

Luego, se implementa la función abrirArchivo, la cual recibe como parámetros el nombre del archivo (archivo) y el modo de apertura (modoApertura), la cual para el caso del proceso sensor, siempre será de lectura ("r"). Ante esto, se realizará tanto el manejo de errores como el resultado de una apertura exitosa, donde, siendo este último el caso, se retornará un puntero hacia el archivo abierto.

```
/*--- Funcin para abrir un archivo de
    texto---*/
FILE *abrirArchivo(char *archivo, char
    *modoApertura) {
  FILE *arch = fopen(archivo, modoApertura);
      // Apertura del archivo
  if (!arch) {                    // Manejo
      de errores en la apertura
    perror("\nError al abrir el archivo\n");
    exit(1);
  } else { // Manejo de apertura xitosa
    printf("\nArchivo abierto\n");
    return arch;
  }
}
```

Continuando, se visualiza la función *abrirPipe*, el cual permitirá la comunicación con el proceso monitor, esta función recibirá como parámetros el nombre del pipe (pipe_nominal) y el modo de apertura *(modoApertura)*, recordando que los pipes pueden ser definidos como:

- ***O_RDONLY*** para lectura.
- ***O_WRONLY*** para escritura, el modo que se implementará para el proceso sensor.
- ***O_RDWR*** para lectura y escritura.

```
/*--- Funcin para abrir un pipe ---*/
int abrirPipe(char *pipe_nominal, int
    modoApertura) {
  int pipe = open(pipe_nominal, modoApertura);
      // Apertura del pipe
  if (pipe < 0) { // Manejo de errores en la
      apertura
    perror("\nError al abrir el pipe\n");
    exit(1);
  } else { // Manejo de apertura xitosa
    printf("\nPipe abierto\n");
    return pipe;
  }
}
```

Así, se hará el manejo de errores y en caso de que la apertura sea exitosa, la función retornará un descriptor del archivo del pipe abierto, el cual es un valor positivo que el programa usa para referirse al archivo durante la ejecución. Adicionalmente, se tiene la función *simularSensor*, la cual recibe por parámetro el tipo de sensor a evaluar *(tipo_sensor)*, el tiempo en el que enviaran los datos entre sí *(tiempo)*, el nombre del archivo con las mediciones a leer *(archivo)* y el nombre del pipe para la comunicación con el proceso monitor *(pipe_nominal)*.

```
/*--- Funcin para simular la lectura de
    mediciones y enviarlas al monitor ---*/
void simularSensor(int tipo_sensor, int
    tiempo, char *archivo, char
    *pipe_nominal) {
  printf("\nIniciando simulacin del sensor\n");

  int pipe = abrirPipe(pipe_nominal,
      O_WRONLY); // Abrir pipe en modo
      escritura
  FILE *archivo_leido = abrirArchivo(archivo,
      "r"); // Abrir archivo modo lectura
  char buffer[BUFFERSIZE];

  // Inicializar la estructura
  DatosSensor datos;
  datos.tipo_sensor = tipo_sensor;

  while (fgets(buffer, BUFFERSIZE,
      archivo_leido)) { // Lectura del archivo

    datos.medicion = atof(buffer);
    printf("\nLectura del archivo: %.2f\n",
        datos.medicion);

    // Enviar la estructura con el tipo de
        medicin y el valor de la medicin
    write(pipe, &datos, sizeof(DatosSensor));
    sleep(tiempo); // Espera un tiempo
  }

  fclose(archivo_leido); // Cierre del archivo
  close(pipe);      // Cierre del pipe
  exit(EXIT_SUCCESS);
  printf("\nSimulacin del sensor
      completada\n");
}
```

Esta función se encargará principalmente de enviar las mediciones leídas al proceso monitor, para ello, llamará a las funciones respectivas para abrir el pipe y el archivo correspondiente, inicializará la estructura de datos a enviar, leerá el

archivo recibido y, escribirá la estructura datos que contiene el tipo de medición *(tipo_sensor)* y el valor de esta misma *(medicion)*; seguidamente, define un tiempo de espera determinado antes de continuar con las siguientes mediciones. Al final de completar la lectura de todas las líneas del archivo, la función cerrará tanto el archivo como el pipe abierto.

Finalmente, se evidencia la función principal *main*, la cual es primordial para el funcionamiento del proceso sensor. Esta función, llama a la función *procesarArgumentos* para extraer los valores, luego valida que se hayan ingresado todos los argumentos requeridos, para luego asignar cada uno de los argumentos a ciertas variables para facilitar el uso a lo largo de la función.

Teniendo en cuenta que, se deben ejecutar diferentes procesos sensor, es necesario crear varios procesos hijos haciendo uso de fok(), donde cada uno tomará la tarea de ejecutar la función simularSensor para el archivo de mediciones asignado. De esta forma, se esperará a que todos los procesos hijo terminen sus tareas para concluir con el programa de sensor.

III. Evaluación de Rendimiento

El algoritmo explicado anteriormente tiene una complejidad de $O(n^3)$. Sin embargo, su eficiencia se puede mejorar realizando programación de hilos de ejecución. Los hilos de ejecución nos permiten explotar las ventajas del paralelismo debido a que se puede dividir el algoritmo en tantas partes como se quiera y ejecutar las operaciones matemáticas de manera simultanea, en lugar de una a una.

El proceso descrito previamente disminuye de manera abismal el tiempo necesario para la conclusión del algoritmo ya que la programación de hilos de ejecución admite un uso mucho más eficiente de los recursos de computación en donde se esté ejecutando el algoritmo.

A. ¿Por qué mejora el tiempo de ejecución del algoritmo?

La programación de hilos le permite al programador interactuar con mayor cercanía con los recursos de computación con los que cuenta la máquina en donde se está ejecutando determinado programa. Al momento de crear un hilo en un programa, el planificador de procesos del sistema operativo se encarga de asignar el hilo a los diferentes núcleos del procesador que se encuentren disponibles y, de esta manera, se logra maximizar la capacidad de ejecutar operaciones de forma paralela y, en general, se maximiza el rendimiento de los recursos de computación disponibles.

B. Programación de Hilos de Ejecución para la Multiplicación de Matrices

ción C, se hace uso de las funciones dentro la librería `pthread.h`, en este caso, se utilizan las funciones `pthread_create` y `pthread_create`. La primera se usa para la creación de hilos de ejecución y la asignación de un proceso o parte de un proceso a un hilo. Por otro

A continuación se presenta como se programó y se asigno el proceso de la multiplicación de matrices a diferentes hilos de ejecución.

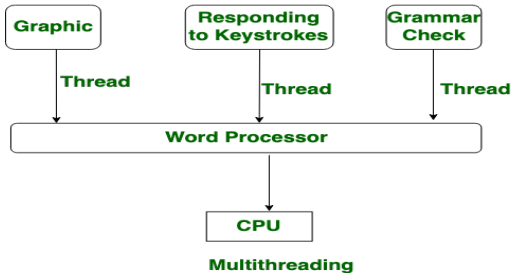

Fig. 2. Diagrama simple que representa la asignación de un proceso a hilos de ejecución.

```
/*Creacion de tantos hilos de
   ejecucion como se hayan recibido
   como argumento*/
pthread_t hilos[H];
/*Ciclo para asignar un proceso a
   tantos hilos como se hayan
   recibido por argumento*/
for(int h=0; h<H; h++){
   /*Instancia de un puntero de
      tipo datosMM.
   Se asigna su tamano en memoria
      por medio del metodo malloc*/
   struct datosMM* valoresMM =
      (struct datosMM *)
      malloc(sizeof(struct
      datosMM));
   valoresMM->N = N;
   valoresMM->H = H;
   valoresMM->mA = mA;
   valoresMM->mB = mB;
   valoresMM->mC = mC;
   /*Asignando como id el valor de
      h*/
   valoresMM->idHilo = h;
   /*Funcion par la creacion de
      hilos de ejecucion.*/
   pthread_create(&hilos[h], NULL,
      multiplicacionMatriz,
      valoresMM);
}
/*Puesta en espera para terminacion
    de cada uno de los hilos*/
for (int h = 0; h < H; h++)
{
   pthread_join(hilos[h], NULL);
}
```

C. División del Proceso de Multiplicación de Matrices

Dada la naturaleza del algoritmo de multiplicación de matrices, se ha realizado la división del proceso de multiplicación en dos. Al primer hilo se le asignan las operaciones asociadas a la multiplicación de la primera mitad de la matriz y, al segundo hilo se le asigna la multiplicación de la segunda mitad de la matriz.

A pesar de que se puede ajustar para diferentes parámetros, en este caso, esta implementación debe cumplir las siguientes pre-condiciones para su correcto funcionamiento:

- El tamaño de la matriz debe ser par, es decir *2n*.

- La cantidad de hilos debe ser igual a 2.

D. Código Fuente

Para observar el código fuente con la implementación del algoritmo mejorado haciendo uso de las ventajas de la programación de hilos indague el siguiente repositorio al cual puede acceder haciendo clic en el siguiente enlace: https://github.com/diego4lbarracin/Matrix_Multiplication

IV. Paradigmas de Programación

V. Metodología de Experimentación

VI. Evaluación Exhaustiva

VII. Conclusiones

VIII. Referencias

[1] https://www.geeksforgeeks.org/difference-between-multi-tasking-and-multi-threading/