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, procesa los argumentos 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.

B. Monitor

Prosiguiendo con la clase monitor, esta clase se encargará de recibir las mediciones por parte del proceso sensor, validar la medición, y, de acuerdo con el tipo de medición, asignarla al hilo de esta misma, quien se encargará de guardar los datos en un archivo. Para ello, es importante incluir las librerías pertinentes siendo algunas de estas aptas para el manejo de hilos, errores, tiempo y semáforos.

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

Luego, se definen las siguientes variables y estructuras:

- **MonitorArgumentos:** Contiene las variables que representan los argumentos que debe recibir el proceso monitor.
- **DatosSensor:** Estructura que contiene los datos que recibirá dentro del mensaje por parte del proceso sensor.
- **Definición de *buffers* y contadores:** Se determinan los *buffers* de temperatura y *ph* con sus respectivos contadores que ayudaran a almacenar los datos que sean pasados por el hilo recolector.
- **Definición de semáforos:** Para esto se podrán categorizar en 4 tipos:
  - **Buffer_mutex:** Garantizará la exclusión mutua evitando que se presenten condiciones de carrera en el hilo recolector, es decir, evitar que múltiples procesos sensor accedan a la vez al hilo recolector puesto que es un recurso compartido donde acceden las mediciones de temperatura y de ph.
  - **Temperatura_completa y Ph_completa:** Indicarán que el buffer de la respectiva medición ya está lleno porque se le ha agregado una medición.
  - **Temperatura_vacia y Ph_vacia:** Indicarán que el buffer en la respectiva medición esta libre para recibir una medición.
  - **Recolector_finalizador:** Indicador que determina cuando el hilo recolector ha finalizado su ejecución.

```
// Tamao mximo del buffer
   #define MAX_BUFFER_SIZE 50
/*--- Estrcutura de datos con los argumentos
    para el proceso monitor ---*/
   typedef struct {
   int tam_buffer; // Tamao de los buffer
       donde se colocaran las medidas
   char *arch_temperatura; // Archivo para
       guardar mediciones de temperatura
   char *arch_ph;    // Archivo para guardar
       mediciones de ph
   char *pipe_nominal; // Permite la
       comunicacin entre los procesos
   } MonitorArgumentos;
/*--- Estructura de datos para descomponer el
    mensaje recibido ---*/
   typedef struct {
   int tipo_sensor; // Tipo de sensor: 1 para
       temperatura, 2 para pH
   float medicion; // Medicin del sensor
   } DatosSensor;
   /*Declaracion de variables haciendo uso de
       la palabra clave extern
   para decirle al compilador de que la
       definicon se encuentra en otro
   lugar. (monitor.c)*/
   extern sem_t buffer_mutex;
   extern sem_t temperatura_completa;
   extern sem_t ph_completa;
   extern sem_t recolector_finalizado;
   extern int cont_temperatura;
   extern int cont_ph;
   extern char
       temperatura_buffer[MAX_BUFFER_SIZE];
   extern char ph_buffer[MAX_BUFFER_SIZE];
```

Continuando, tenemos la función *abrirArchivo*, la cual se utilizará para abrir los archivos de salida de las mediciones de temperatura y *ph*.

```
/*--- 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; // Retorno del puntero del
```

```
        archivo abierto
    }
```

Ahora, se presenta la función *H_recoelctor*, la cual es la encargada de leer el mensaje enviado a través del pipe y asignar la medición a su respectivo *buffer*. Para ello, la función inicialmente crea el pipe y luego lo abre, seguidamente, lee el mensaje y lo descompone haciendo uso de la estructura *DatosSensor*; una vez obtenga el tipo de sensor y la medición del mensaje que recibió, realizará la asignación a los *buffers*.

Para lo anterior, si el tipo de sensor que recibió de la medición es "1", será interpretado como una medición de temperatura, por lo que primeramente, validará que la medición recibida esté dentro del rango determinado para este sensor, si es el caso, pondrá en ejecución el semáforo de buffer_mutex, guardará la medición dentro de temperatura_buffer y se le indicará al semáforo de *temperatura_completa* que ya ha sido guardada la medición y puede recibir otra medición en espera; en caso de que la medición este fuera del rango demarcado para el sensor este se descartará. Este mismo paso a paso lo ejecutará *H_recolector* en caso de recibir una medición de tipo "2", es decir de tipo *ph*, haciendo uso del *buffer*, contador y semáforos de este mismo. Posteriormente, se presentan las funciones de los hilos *ph* y *temperatura* (*H_ph y H_temperatura*), las cuales se encargan de obtener las mediciones de sus respectivos *buffers* y escribirlos en un archivo de salida, ambas funciones siguen el mismo patrón por lo que, se profundizará en una explicación general de su comportamiento.

Inicialmente, se hace un llamado a la función *abrirArchivo*, el cual permite abrir el archivo donde se escribirán las mediciones, luego mediante los semáforos *ph_completa* o *temperatura_completa* se validará que hayan datos disponibles en cada uno de los *buffers* para escribir; se obtiene el valor de la medición y, se crea la variable *horaActual* la cual obtendrá la hora de guardado, estos dos componentes son los que se escribirán dentro del archivo correspondiente. Finalmente se utilizan los semáforos respectivos para indicar que se puede escribir otra medición y se cierra el archivo de escrita una vez todas las mediciones que contienen los *buffers* hayan sido escritas.

```
/*--- Funcin del hilo H-ph ---*/
void *H_ph(char *arch_ph) {
  FILE *archivoPhFinal =
     abrirArchivo(arch_ph, "a"); // Abrir
        archivo modo escritura
  while (1) {
    sem_wait(&ph_completa);
    sem_wait(&buffer_mutex);
    // Obtener la ltima medicion de ph del
       buffer
    float valorPh = ph_buffer[cont_ph - 1];
    printf("\nValor ph:%.2f", valorPh);
    // Obtener el tiempo actual de guardado en
       el archivo
    time_t horaActual;
    time(&horaActual);
    // Escribir en el archivo la medicin y la
       hora de guardado
    fprintf(archivoPhFinal, "%.2f (hora de
       guardado: %s)\n",
                valorPh,
                    asctime(localtime(&horaActual)));
    fflush(archivoPhFinal); // Forzar la
       escritura de los datos
    sem_post(&buffer_mutex); // Permitir el
       acceso a otro proceso
  }
  fclose(archivoPhFinal); // Cerrar el archivo
     de texto
  return NULL;
}
/*--- Funcin del hilo H-temperatura ---*/
void *H_temperatura(char *arch_temperatura) {
  FILE *archivoTempFinal =
     abrirArchivo(arch_temperatura, "a"); //
        Abrir archivo modo escritura
  while (1) {
    sem_wait(&temperatura_completa);
    sem_wait(&buffer_mutex);
    // Obtener la ltima medicion de
       temperatura del buffer
    float valorTemp =
       temperatura_buffer[cont_temperatura -
       1];
    printf("\nValor temperatura:%.2f",
       valorTemp);
    // Obtener el tiempo actual de guardado en
       el archivo
    time_t horaActual;
    time(&horaActual);
    // Escribir en el archivo con la hora
       actual
    fprintf(archivoTempFinal, "%.2f (hora de
       guardado: %s)\n",
          valorTemp,
              asctime(localtime(&horaActual)));
    fflush(archivoTempFinal); // Forzar la
       escritura de los datos
    sem_post(&buffer_mutex); // Permitir el
       acceso a otro proceso
  }
  fclose(archivoTempFinal); // Cerrar el
     archivo de texto
  return NULL;
}
```

Por último, se encuentra la función principal *main*, la cual llama a la función *procesarArgumentos* para validar las correctas entradas al proceso, se hace la asignación de las entradas a variables determinadas para su fácil manejo, se inicializan lo semáforos, se crean los tres hilos (recolector, ph y temperatura) indicándoles la tarea y parámetros necesarios para su ejecución, y, así, para concluir, se determinan las funciones de espera en la ejecución de los hilos y la destrucción de los semáforos.

III. Proceso de Ejecución

A. Compilación de los Archivos Fuente

En primer lugar, se deben compilar los archivos fuentes para que puedan ser ejecutados. El proceso está automatizado por medio de un archivo *Makefile*.

B. Ejecución del Monitor

El proceso monitor se ejecuta hasta que el usuario termine el proceso de manera voluntaria.

C. Ejecución del Sensor

El proceso sensor se ejecuta por cada archivo con información que se quiera analizar, ya sea para analizar la información relacionada con el *ph* o con la *temperatura*.

D. Resultados

A continuación, podrá ver los resultados de analizar la información por defecto brindada por la guía del proyecto para el monitoreo de *ph* y *temperatura*.

```
//Resultados del Anlisis del PH
      6.00 (hora de guardado: Thu May 23
          23:59:59 2024
      )
      6.00 (hora de guardado: Fri May 24
          00:00:09 2024
      )
      7.00 (hora de guardado: Fri May 24
          00:00:39 2024
      )
      7.00 (hora de guardado: Fri May 24
          00:00:49 2024
      )
//Resultados del Anlisis de la Temperatura
      20.00 (hora de guardado: Fri May 24
          00:00:41 2024
      )
      28.00 (hora de guardado: Fri May 24
          00:00:44 2024
      )
      29.00 (hora de guardado: Fri May 24
          00:00:47 2024
      )
      31.00 (hora de guardado: Fri May 24
          00:00:50 2024
      )
```

IV. Conclusiones

El paradigma de programación paralela es muy importante puesto que nos permite entender el desarrollo detrás de aplicaciones que se ejecutan de esta manera. Con este proyecto logramos aplicar todo los conocimientos aprendidos durante el curso y es seguro que serán útiles para nuestro desarrollo como profesionales de las tecnologías de la información.

Para acceder al código fuente del proyecto, presione sobre el siguiente enlace: https://github.com/majo425/ProyectoSO-sensores.