<!--Sistemas Operativos-->

Proyecto: Monitoreo de Sensores {

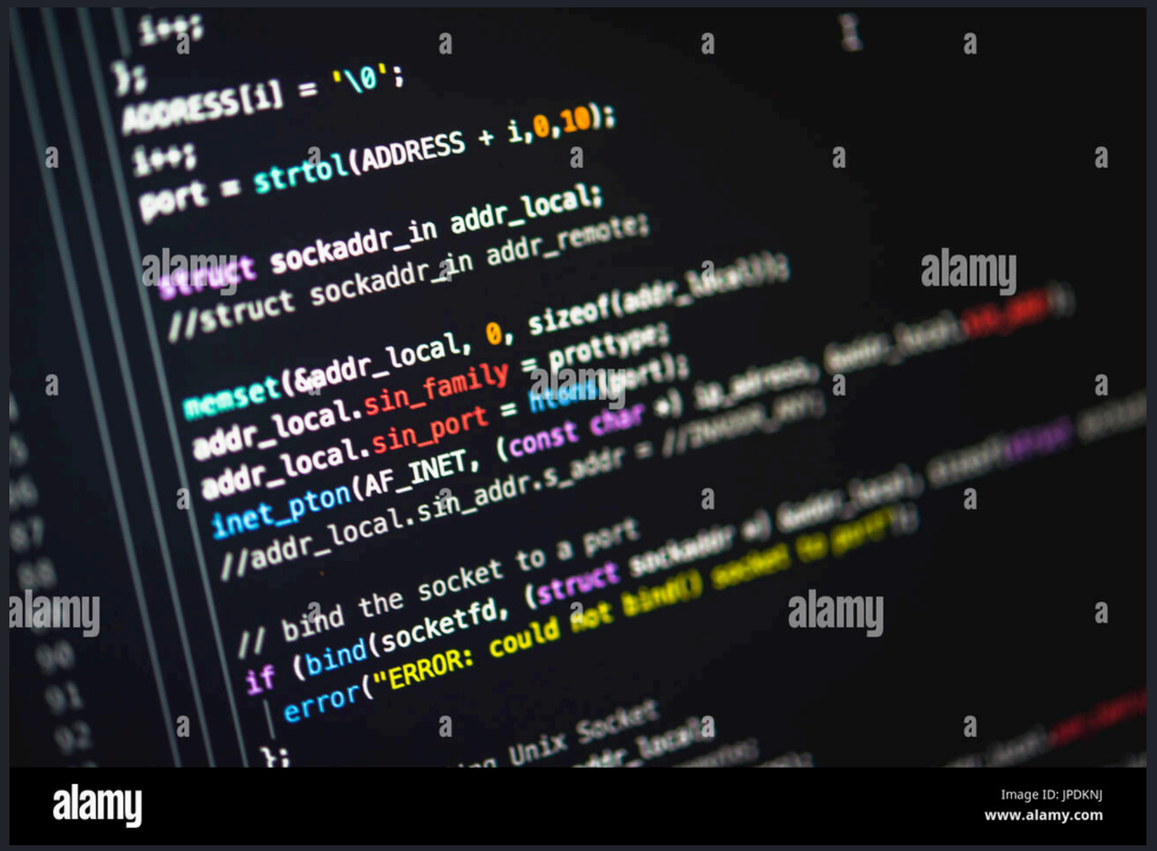
<Por="Juan Paez y Carlos Mejia"/>

}

Introducción {

El proyecto "Monitoreo de Sensores" tiene como objetivo principal desarrollar un sistema de monitoreo en tiempo real para supervisar parámetros ambientales clave, como el pH y la temperatura.

Para lograr esto, se utilizarán pipes nominales para la comunicación entre procesos y semáforos para la sincronización de hilos. Estas tecnologías garantizarán una comunicación segura y una coordinación adecuada entre los diferentes componentes del sistema, lo que permitirá detectar y registrar cambios importantes en los parámetros ambientales de manera eficiente y confiable



}

Funcionalidad {

COMUNICACIÓN ENTRE PROCESOS:

Se emplea un pipe nominal para la comunicación entre el sensor y el monitor, facilitando un intercambio eficiente y sincronizado de datos entre los distintos componentes del sistema.

SINCRONIZACIÓN DE HILOS:

Los hilos del monitor hacen uso de semáforos para asegurar la sincronización en el acceso a los buffers de datos, previniendo condiciones de carrera y garantizando una manipulación segura de las mediciones recibidas del sensor.

PROCESAMIENTO DE DATOS

El monitor procesa las mediciones del sensor, guardándolas en archivos de texto distintos para los datos de pH y temperatura. También vigila los rangos de las mediciones y emite alertas si detecta valores fuera de los límites establecidos.

ESTRUCTURAS DE DATOS COMPARTIDAS

Se crean buffers de datos para facilitar la comunicación entre los hilos del monitor, permitiendo almacenar temporalmente las mediciones recibidas del sensor y garantizando una transferencia segura de información entre los distintos hilos.

MANEJO DE ARCHIVOS

El proyecto incorpora la lectura de mediciones desde un archivo de texto por parte del sensor, junto con el almacenamiento de las mediciones procesadas en archivos de texto por parte del monitor. Esta metodología facilita una gestión eficiente de los datos generados por los sensores.

}

Descripción del Proyecto {

Los archivos principales implementados en el proyecto tienen roles específicos en el sistema de monitoreo de sensores:

“ sensor.cpp ” :

Se encarga de leer datos de un archivo de texto que contiene las mediciones de los sensores y enviar estas mediciones a través de un pipe nominal al proceso monitor.

“ monitor.cpp ” :

Es responsable de recibir las mediciones de los sensores a través del pipe nominal, procesar y almacenar estas mediciones en archivos de texto separados para los datos de pH y temperatura. Además, supervisa el rango de las mediciones y genera alertas si se detectan valores fuera de los límites establecidos.

“ buffer.cpp “ :

Se utiliza para garantizar una comunicación segura entre los hilos productor y consumidor, evitando posibles problemas de concurrencia.

sensor.cpp {

<!--Pseudocodigo-->

```
Inicio del programa:
    Inicializar variables:
        a = 0  // Contador de mediciones
        opt // Variable para los argumentos de línea de comandos
        tipoSensor = 0  // Tipo de sensor
        intervaloTiempo = 0  // Intervalo de tiempo entre mediciones
        nombreArchivo = nullptr  // Nombre del archivo de datos
        nombrePipe = nullptr  // Nombre del pipe

    Analizar argumentos de línea de comandos:
        Si hay argumentos:
            Si el argumento es -s:
                Asignar el valor a tipoSensor
            Si el argumento es -t:
                Asignar el valor a intervaloTiempo
            Si el argumento es -f:
                Asignar el valor a nombreArchivo
            Si el argumento es -p:
                Asignar el valor a nombrePipe

    Abrir el archivo de datos:
        Si se puede abrir correctamente:
            Leer el archivo línea por línea:
                Incrementar el contador de mediciones
                Escribir la medición en el pipe
                Esperar el intervalo de tiempo especificado

    Cerrar el archivo de datos y el pipe

Fin del programa
```

}

sensor.cpp {

Explicación :

- Se incluyen las bibliotecas estándar necesarias para manejar la entrada y salida, así como para trabajar con archivos de texto y pipes.
- La función main inicializa variables para almacenar los argumentos de línea de comandos y otros parámetros.
- Se analizan los argumentos de línea de comandos utilizando la función getopt() para especificar el tipo de sensor, el intervalo de tiempo, el nombre del archivo y el nombre del pipe.
- Se intenta abrir el archivo de datos y se verifica si se puede abrir correctamente. Si falla, se imprime un mensaje de error.
- Se abre el pipe en modo escritura no bloqueante y se intenta nuevamente si falla la apertura.
- Se lee cada línea del archivo de datos y se escribe en el pipe. Cada medición se muestra en la salida estándar.
- Después de escribir cada medición, el programa espera el intervalo de tiempo especificado antes de la siguiente medición.
- Finalmente, se cierran el archivo de datos y el pipe, y el programa termina con un código de retorno 0.

}

monitor.cpp {

<!--Pseudocodigo y explicacion-->

1. Incluir bibliotecas necesarias: iostream, fstream, pthread.h, unistd.h, semaphore.h, sys/stat.h, fcntl.h, ctime, buffer.h

2. Definir una estructura ArgumentosHilo para los argumentos del hilo, que incluyen los buffers de pH y temperatura, el nombre de la tubería y un semáforo.

3. Definir funciones es_flotante() y es_entero() para verificar si una cadena representa un número flotante o entero, respectivamente.

4. Definir una función obtenerHoraActual() para obtener la hora actual en formato HH:MM:SS.

5. Definir la función h_recolector(void *arg) para el hilo que recolecta datos de los sensores y los maneja entre hilos.

6. Definir la función h_ph(void *arg) para el hilo encargado de manejar los datos del pH.

7. Definir la función h_temperatura(void *arg) para el hilo encargado de manejar los datos de temperatura.

8. En la función main():
    a. Inicializar variables y procesar argumentos de la línea de comandos.
    b. Crear la tubería y abrir la tubería.
    c. Crear buffers y argumentos del hilo.
    d. Inicializar un semáforo.
    e. Crear hilos para el recolector, el manejo del pH y el manejo de la temperatura.
    f. Unir los hilos al proceso principal.
    g. Cerrar la tubería y destruir el semáforo.
    h. Fin del programa.

}

buffer.cpp {

<!--Pseudocodigo-->

```
Clase Buffer
    Atributos:
        - size: entero // Tamaño máximo del buffer
        - mutex: Mutex // Para garantizar la exclusión mutua

        - condProducer: Variable de condición // Para sincronizar a los productores

        - condConsumer: Variable de condición // Para sincronizar a los consumidores

        - dataQueue: Cola de datos // Para almacenar los datos en el buffer

    Constructor:
        Función Buffer(size: entero)
            size <- size
            Inicializar mutex
            Inicializar condProducer
            Inicializar condConsumer
            Crear una nueva cola de datos

    Destructor:
        Función ~Buffer()
            Destruir mutex
            Destruir condProducer
            Destruir condConsumer

    Función agregar(datos: cadena)
        Adquirir mutex
        Mientras el tamaño de la cola de datos sea mayor o igual que el tamaño máximo del buffer
            Esperar en condProducer
        Agregar datos a la cola de datos
        Señalar a los consumidores que hay datos disponibles
        Liberar mutex

    Función remover() -> cadena
        Adquirir mutex
        Mientras la cola de datos esté vacía
            Esperar en condConsumer
        Extraer datos de la cola de datos
        Señalar a los productores que hay espacio disponible en el buffer
        Liberar mutex
        Devolver los datos extraídos
```

}

sensor.cpp {

Explicación :

1. Se incluyen las bibliotecas necesarias para manejar la concurrencia y la sincronización de hilos.

2. Se define la clase Buffer que actúa como un área de almacenamiento temporal para datos transferidos entre productores y consumidores.

3. El constructor inicializa el tamaño del buffer y crea los recursos de sincronización (mutex y variables de condición). El destructor libera estos recursos cuando el buffer ya no es necesario.

El método **add** agrega datos al buffer de manera sincronizada:
- Bloquea el mutex para garantizar la exclusión mutua.
- Espera hasta que haya espacio disponible en el buffer si está lleno.
- Agrega datos a la cola de datos (**dataQueue**).
- Señala a los consumidores que hay datos disponibles.
- Desbloquea el mutex.

El método **remove** elimina datos del buffer de manera sincronizada:
- Bloquea el mutex para garantizar la exclusión mutua.
- Espera hasta que haya datos disponibles en el buffer si está vacío.
- Extrae datos de la cola de datos (**dataQueue**).
- Señala a los productores que hay espacio disponible en el buffer.
- Desbloquea el mutex.
- Devuelve los datos extraídos.

}

Prueba de Funcionamiento{

}

<!--Sistemas Operativos-->

Gracias {

<Por="Juan Paez y Carlos Mejia"/>

}