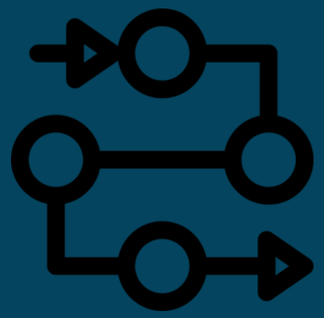
SISTEMAS OPERATIVOS

PROYECTO

Monitoreo de Sensores

Juan Felipe Galvis, Juan Diego Reyes, Andrés Pérez



PROCESO SENSOR

```c
int main(int argc, char *argv[]) {
    int tipo_sensor = 0;        // Variable para almacenar el tipo de sensor
    int tiempo = 0;             // Variable para almacenar el tiempo de espera entre lecturas
    char *archivo = NULL;       // Puntero para el nombre del archivo de datos del sensor
    char *pipe_nominal = NULL; // Puntero para el nombre del pipe

    int opt;                    // Variable para el retorno de getopt
```

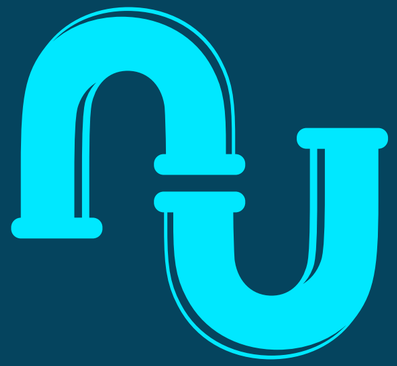
Banderas para los parámetros de entrada

```c
while ((opt = getopt(argc, argv, "s:t:f:p:")) != -1) {
    switch (opt) {
        case 's':
            tipo_sensor = atoi(optarg); // Convierte el argumento de tipo de sensor a entero
            break;
        case 't':
            tiempo = atoi(optarg);      // Convierte el argumento de tiempo a entero
            break;
        case 'f':
            archivo = optarg;           // Asigna el argumento de archivo a la variable correspondiente
            break;
        case 'p':
            pipe_nominal = optarg;      // Asigna el argumento de pipe a la variable correspondiente
            break;
        default:
            print_usage();              // Imprime el uso correcto si hay una opción inválida
            exit(EXIT_FAILURE);
    }
}
```

Verificaciones y apertura

```c
// Verifica que todos los argumentos requeridos han sido proporcionados
if (tipo_sensor == 0 || tiempo == 0 || archivo == NULL || pipe_nominal == NULL) {
    print_usage(); // Imprime el uso correcto si falta algún argumento
    exit(EXIT_FAILURE);
}

// Abre el pipe en modo escritura
int pipe_fd = open(pipe_nominal, O_WRONLY);
if (pipe_fd == -1) {
    perror("open pipe"); // Imprime un mensaje de error si falla la apertura del pipe
    exit(EXIT_FAILURE);
}

// Abre el archivo de datos del sensor en modo lectura
FILE *file = fopen(archivo, "r");
if (file == NULL) {
    perror("fopen"); // Imprime un mensaje de error si falla la apertura del archivo
    close(pipe_fd);
    exit(EXIT_FAILURE);
}
```

Escritura en Buffer



```c
char line[256]; // Buffer para almacenar las líneas leídas del archivo
// Bucle para leer las líneas del archivo
while (fgets(line, sizeof(line), file)) {
    float value = atof(line); // Convierte la línea leída a un valor flotante
    // Escribe el tipo de sensor y el valor en el pipe
    write(pipe_fd, &tipo_sensor, sizeof(int));
    write(pipe_fd, &value, sizeof(float));
    sleep(tiempo); // Espera el tiempo especificado antes de la próxima lectura
}
```

PROCESO MONITOR

```c
#define DEFAULT_BUFFER_SIZE 10 // Tamaño por defecto del buffer

// Estructura que define el buffer circular
typedef struct {
    float *buffer; // Buffer circular para almacenar valores
    int size; // Tamaño del buffer
    int in; // Índice de entrada
    int out; // Índice de salida
    sem_t full; // Semáforo para contar elementos llenos
    sem_t empty; // Semáforo para contar espacios vacíos
    pthread_mutex_t mutex; // Mutex para sincronizar el acceso al buffer
} Buffer;

Buffer ph_buffer; // Buffer para valores de PH
Buffer temp_buffer; // Buffer para valores de temperatura
int pipe_fd; // Descriptor de archivo para el pipe
```

Función inicializador de buffer y semáforos

```c
// Función para inicializar un buffer
void init_buffer(Buffer *buffer, int size) {
    buffer->size = size; // Inicializa el tamaño del buffer
    buffer->buffer = (float *)malloc(sizeof(float) * size); // Reserva memoria para el buffer
    buffer->in = 0; // Inicializa el índice de entrada
    buffer->out = 0; // Inicializa el índice de salida
    sem_init(&buffer->full, 0, 0); // Inicializa el semáforo de elementos llenos a 0
    sem_init(&buffer->empty, 0, size); // Inicializa el semáforo de espacios vacíos al tamaño del buffer
    pthread_mutex_init(&buffer->mutex, NULL); // Inicializa el mutex
}
```

Liberar memoria y destrucción

```c
// Función para liberar la memoria del buffer y destruir los semáforos y el mutex
void free_buffer(Buffer *buffer) {
    free(buffer->buffer); // Libera la memoria del buffer
    sem_destroy(&buffer->full); // Destruye el semáforo de elementos llenos
    sem_destroy(&buffer->empty); // Destruye el semáforo de espacios vacíos
    pthread_mutex_destroy(&buffer->mutex); // Destruye el mutex
}
```

Escritura mediciones

```c
// Función para escribir una medición en un archivo
void write_measurement(FILE *file, float value) {
    time_t now = time(NULL); // Obtiene el tiempo actual
    struct tm *gmt_time = gmtime(&now); // Convierte el tiempo a GMT
    gmt_time->tm_hour -= 5; // Ajusta la hora a GMT-5
    mktime(gmt_time); // Normaliza la estructura de tiempo

    char time_str[100];
    strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", gmt_time); // Formatea la hora
    fprintf(file, "%s: %.2f\n", time_str, value); // Escribe la medición en el archivo
    fflush(file); // Asegura de que los datos se escriban en el archivo inmediatamente
}
```

Función recolectar

```c
void *recolector(void *arg) {
    while (monitor_running) {
        int tipo_sensor;
        float value;
        ssize_t num_bytes = read(pipe_fd, &tipo_sensor, sizeof(int)); // Lee el tipo de sensor del pipe
        if (num_bytes <= 0) {
            // No hay más datos o se cerró el pipe
            usleep(10000); // Dormir por un breve momento antes de intentar leer de nuevo
            continue;
        }
        read(pipe_fd, &value, sizeof(float)); // Lee el valor del sensor del pipe

        if (value < 0) {
            printf("Valor erróneo recibido: %.2f\n", value); // Imprime un mensaje si el valor es erróneo
            continue;
        }

        Buffer *buffer = (tipo_sensor == 1) ? &temp_buffer : &ph_buffer; // Selecciona el buffer correspondiente
```

```c
        sem_wait(&buffer->empty); // Espera un espacio vacío en el buffer
        pthread_mutex_lock(&buffer->mutex); // Bloquea el mutex
        buffer->buffer[buffer->in] = value; // Escribe el valor en el buffer
        buffer->in = (buffer->in + 1) % buffer->size; // Actualiza el índice de entrada
        pthread_mutex_unlock(&buffer->mutex); // Desbloquea el mutex
        sem_post(&buffer->full); // Incrementa el semáforo de elementos llenos
    }

    printf("Recolector ha terminado de recibir datos.\n");
    return NULL;
}
```

Datos de PH y Temperatura

```c
void *h_ph(void *arg) {
    bool print_title = false;
    char *file_name = (char *)arg;
    FILE *file = fopen(file_name, "a"); // Abrir archivo en modo append
    if (file == NULL) {
        perror("fopen");
        return NULL;
    }
    if(!print_title){
        fprintf(file, "\tResultados de sensores de Ph\n\n");
        print_title = true;
    }
    while (monitor_running || sem_trywait(&ph_buffer.full) == 0) {
        if (!monitor_running && sem_trywait(&ph_buffer.full) != 0) {
            break; // Salir del ciclo si el monitor ha terminado y el buffer está vacío
        }
        sem_wait(&ph_buffer.full); // Espera un elemento lleno en el buffer
        pthread_mutex_lock(&ph_buffer.mutex); // Bloquea el mutex
        float value = ph_buffer.buffer[ph_buffer.out]; // Lee el valor del buffer
        ph_buffer.out = (ph_buffer.out + 1) % ph_buffer.size; // Actualiza el índice de salida
        pthread_mutex_unlock(&ph_buffer.mutex); // Desbloquea el mutex
        sem_post(&ph_buffer.empty); // Incrementa el semáforo de espacios vacíos

        if (value < 6 || value > 8) {
            printf("Alerta: PH fuera de rango: %.2f\n", value); // Imprime un mensaje si el PH está fuera de rango
        }
        write_measurement(file, value); // Escribe la medición en el archivo
    }
    fclose(file); // Cierra el archivo
    return NULL;
}
```

Establecer hora para archivo
Inicializador de buffer
Creación del pipe

```
// Establecer la zona horaria a la de Colombia
setenv("TZ", "America/Bogota", 1);
tzset();

init_buffer(&ph_buffer, tam_buffer); // Inicializa el buffer de PH con el tamaño especificado
init_buffer(&temp_buffer, tam_buffer); // Inicializa el buffer de temperatura con el tamaño especificado

// Crea el pipe nombrado
if (mkfifo(pipe_nominal, 0666) == -1) {
    perror("mkfifo");
    exit(EXIT_FAILURE);
}
```

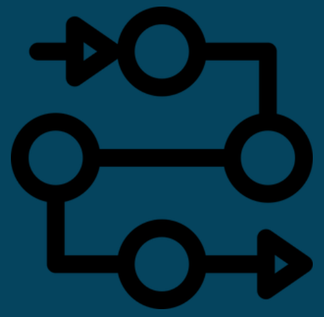
Uso de hilos

```
pthread_t recolector_thread, ph_thread, temp_thread; // Variables para los threads
pthread_create(&recolector_thread, NULL, recolector, pipe_nominal); // Crea el thread para el recolector
pthread_create(&ph_thread, NULL, h_ph, file_ph); // Crea el thread para manejar los datos de PH
pthread_create(&temp_thread, NULL, h_temperatura, file_temp); // Crea el thread para manejar los datos de
peratura

pthread_join(recolector_thread, NULL); // Espera a que termine el thread del recolector
monitor_running = false; // Indica que el monitor ha terminado
pthread_join(ph_thread, NULL); // Espera a que termine el thread de PH
pthread_join(temp_thread, NULL); // Espera a que termine el thread de temperatura
```

Archivo Makefile

Makefile

```
CC=gcc
CFLAGS=-Wall -pthread
TARGETS=sensor monitor

all: $(TARGETS)

sensor: sensor.c
	$(CC) $(CFLAGS) -o sensor sensor.c

monitor: monitor.c
	$(CC) $(CFLAGS) -o monitor monitor.c

clean:
	rm -f $(TARGETS)
	rm -f *.o

```



NUESTRO REPOSITORIOS

- https://github.com/Davidp1905/Sistemas-Operativos/tree/main/Proyecto
- https://github.com/Spdrd/sistemasOperativos/tree/main/Proyecto
- https://github.com/felo312/Sistemas-Operativos/tree/main/Proyecto

MUCHAS GRACIAS