Desarrollo de una Biblioteca de Hilos CEThreads y Algoritmos de Calendarización

Mata C. Carlos
Tecnológico de Costa Rica, Estudiante de Ingeniería en Computadores
Grane R. Ignacio
Tecnológico de Costa Rica, Estudiante de Ingeniería en Computadores
Vargas J. Felipe
Tecnológico de Costa Rica, Estudiante de Ingeniería en Computadores
Truque M. Marcelo
Tecnológico de Costa Rica, Estudiante de Ingeniería en Computadores

***Resumen*—Este proyecto tiene como objetivo desarrollar una biblioteca de hilos llamada CEThreads, la cual simula la funcionalidad de Pthreads. Además, implementa varios algoritmos de calendarización, como Round Robin (RR), Shortest Job First (SJF), Prioridad, First-Come, First-Served (FCFS) y Tiempo Real, para la simulación de un canal que permite el paso de barcos en un sistema concurrencial. Los barcos representan procesos que deben cruzar un canal simulado, donde la biblioteca CEThreads maneja su sincronización y calendarización. La implementación se realizó en C, en un entorno GNU/Linux, y permite la interacción directa del usuario a través de la terminal.**

***Palabras clave*—Hilos, CEThreads, Calendarización, Algoritmos, Sincronización, Pthreads, C, Canal, Sistema Concurrencial**

I. Introducción

La ejecución concurrente de procesos es un concepto fundamental en los sistemas operativos modernos, donde múltiples tareas deben coordinarse y compartirse de forma eficiente los recursos disponibles. Este proyecto, denominado "Scheduling Ships", tiene como objetivo el desarrollo de una biblioteca de hilos llamada CEThreads, que reimplementa funcionalidades básicas de la popular biblioteca Pthreads y permite la simulación de algoritmos de calendarización de procesos. A través de esta biblioteca, se pretende comprender mejor el funcionamiento de la sincronización de hilos y la planificación en sistemas operativos [1].

El modelo simula un canal por el cual barcos (representando hilos o procesos) deben cruzar en función de diferentes políticas de calendarización, tales como Round Robin (RR), First-Come, First-Served (FCFS), Prioridad, Shortest Job First (SJF), y Tiempo Real. Cada algoritmo define una política diferente para el acceso al canal, proporcionando un entorno controlado en el que se pueden estudiar los efectos de diversas estrategias de planificación en un sistema concurrencial.

El desarrollo del proyecto se llevó a cabo en lenguaje C, utilizando un entorno GNU/Linux para aprovechar la capacidad de manejo de recursos de bajo nivel que ofrece este sistema operativo. Adicionalmente, se implementó una interfaz de terminal para que el usuario pueda interactuar con la biblioteca CEThreads, modificando en tiempo real las políticas de planificación de los barcos y observando sus efectos en la simulación.

II. Motivación del Proyecto

La correcta implementación y gestión de hilos es fundamental para cualquier ingeniero en computación, ya que los procesos y su manipulación son esenciales en el manejo de sistemas paralelos. Este proyecto tiene como objetivo profundizar en la comprensión de los conceptos de paralelismo y la eficiencia en la planificación de tareas. La implementación de CEThreads busca ofrecer una alternativa personalizada que permita entender y controlar el funcionamiento de hilos a un nivel más detallado que con Pthreads, lo que permite un mayor control y personalización en el manejo de la concurrencia.

III. Ambiente de Desarrollo

El proyecto fue desarrollado en C sobre GNU/Linux, utilizando CMake para la compilación y CTest para realizar pruebas. Se creó una librería llamada `CEThreads`, que implementa funcionalidades de gestión de hilos, como creación, sincronización y finalización [2]. A través de `CMakeLists.txt`, se definió la creación de esta librería y su enlace con el ejecutable principal:

```
add_library(cethread SHARED cethread.c)
target_link_libraries(${PROJECT_NAME} cethread)
```

Además, se utilizaron pruebas unitarias para validar el correcto funcionamiento de la librería `CEThreads`. Estas pruebas fueron automatizadas con CTest, garantizando que las funciones principales, como la creación de hilos y la gestión de mutexes, se comportaran según lo esperado. Cada prueba se ejecutaba como parte del flujo de compilación, asegurando la estabilidad del sistema.

IV. Atributos

IV-A. Estrategias para el trabajo individual y en equipo de forma equitativa e inclusiva en las etapas del proyecto (planificación, ejecución y evaluación)

Se utilizó la metodología SCRUM para organizar el trabajo, implementando GitHub Projects con issues para cada tarea. Se gestionaron los Pull Requests y se siguió el flujo de trabajo GitFlow, donde cada `branch` correspondía a un issue específico. Para tareas más grandes, se asignaron dos personas, promoviendo la colaboración y distribuyendo equitativamente la carga de trabajo. Durante la planificación, los roles se asignaron según las fortalezas individuales. En la ejecución, se mantuvieron reuniones diarias y comunicación continua a través de WhatsApp y Discord. En la evaluación, se realizaron retroalimentaciones inclusivas utilizando Discord.

IV-B. Planificación del trabajo mediante la identificación de roles, metas y reglas

Los roles fueron asignados al inicio, y cada miembro asumió responsabilidades específicas, como la implementación de la lógica de calendarización o el desarrollo de la interfaz. Las metas se dividieron en entregables parciales con fechas de revisión. Las reglas incluyeron asistencia a reuniones, respeto a los plazos, y una colaboración proactiva.

A continuación se muestra una tabla con los roles asignados en el equipo:

Rol	Responsable(s)
Desarrollo de CEThreads	C. A. Mata, I. Grané
Desarrollo del canal	M. Truque
Interacccion con usuario	M. Truque
Desarrollo de interfaz	C. A. Mata
Lógica de calendarización	L. F. Vargas, I. Grané
Prototipo de hardware	L. F. Vargas, I. Grané

Tabla I
ROLES ASIGNADOS EN EL PROYECTO

IV-C. Acciones que promueven la colaboración entre los miembros del equipo durante el desarrollo del proyecto

Se utilizaron herramientas como GitHub para el control de versiones y plataformas de comunicación en tiempo real como WhatsApp y Discord. Cada Pull Request fue gestionado de manera colaborativa, promoviendo revisiones y discusiones sobre el código.

En la Figura 1 se observa la implementación de GitHub Projects con un Pull Request.

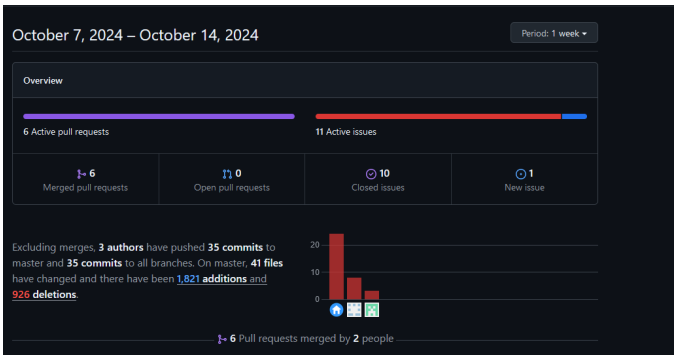


Figura 1. Configuración de GitHub Project

IV-D. Cómo se ejecutan las estrategias planificadas para el logro de los objetivos.

Las estrategias planificadas se ejecutaron mediante una división clara de tareas, donde cada miembro cumplía con su parte mientras recibía retroalimentación constante del equipo. La Figura 2 muestra este proceso mediante Github Project.

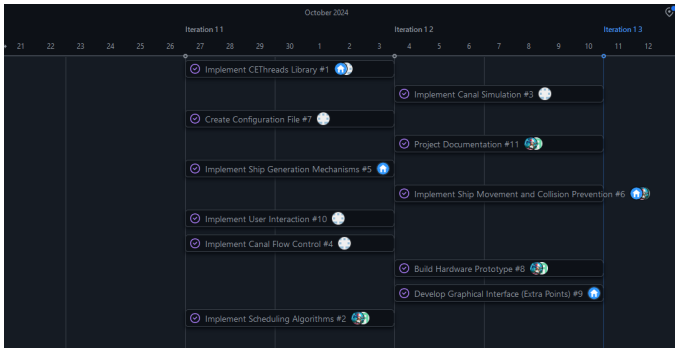


Figura 2. Iteraciones

IV-E. Evaluación para el desempeño

El desempeño se evaluó mediante revisiones periódicas de código y avance del proyecto en reuniones, comparando los resultados obtenidos con las metas definidas. El historial de los *commits* (que serán evaluados como **aportes** en este contexto), así como su consistencia, se puede apreciar en la Figura 3

Figura 3. Aportes de cada miembro

IV-F. Evaluación para las estrategias utilizadas de equidad

Las estrategias de equidad e inclusión se evaluaron al asegurar que todos los miembros tuvieran acceso igualitario a recursos y responsabilidades. La Figura 4 viene a desmotrar que todos los integrantes lograron acoplarse a sus roles, de manera que estos pudieran sentirse cómodos pudiendo desarrollar un punto específico del proyecto. En dicha figura, se aprecia la equitativa aportación de los diferentes miembros a la hora de resolver los *issues* referentes al proyecto.

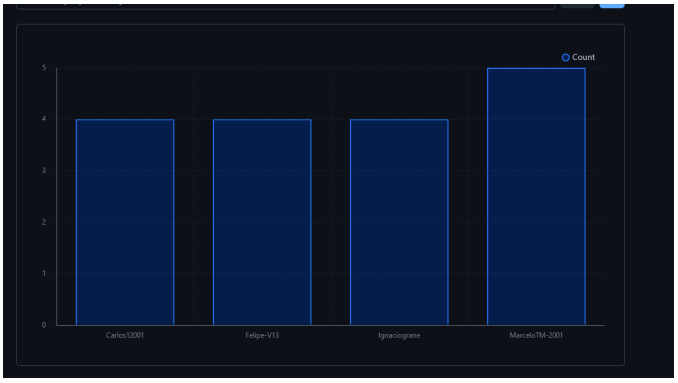


Figura 4. Solución de *issues*

IV-G. Evaluación para las acciones de colaboración

La evaluación de la colaboración se realizó a través de las reuniones de seguimiento, donde se discutieron las interacciones y el trabajo conjunto. Se valoraron aspectos como la disposición para ayudar a otros miembros, la calidad de la comunicación y la integración de las contribuciones individuales en el producto final. Este correcto desempeño se puede ver reflejado en la Figura 5, en donde se observa que la totalidad de los objetivos fueron completados.

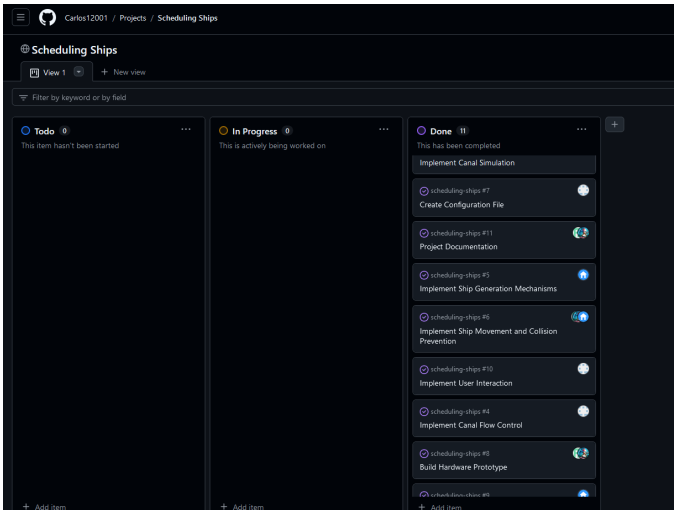


Figura 5. Completitud de tareas y objetivos

V. Diseño

V-A. Diseño del Software

Se detalla un diagrama UML que describe los modulos más importantes del proyecto en la Figura 6.

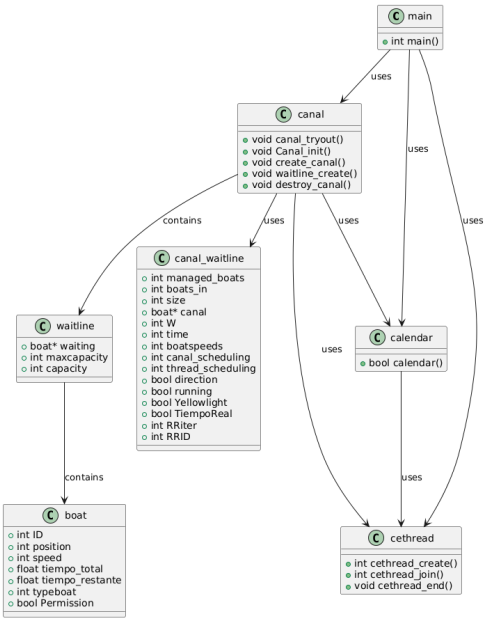


Figura 6. Diagrama de *software* desarrollado

V-B. CEThread

La biblioteca CEThreads fue diseñada para soportar la creación y sincronización de hilos utilizando una estructura simplificada pero eficiente. En lugar de utilizar Pthreads, se implementó una estructura que permite crear hilos utilizando la llamada al sistema `clone`. Esto permitió un manejo más fino de la memoria compartida y el control del ciclo de vida de los hilos, replicando el comportamiento de los hilos ligeros, pero adaptado a las necesidades específicas del proyecto.

La sincronización entre los hilos se gestionó mediante la implementación de un mecanismo de exclusión mutua (mutex). Este fue diseñado para garantizar que solo un hilo a la vez pudiera acceder a recursos compartidos, evitando condiciones de carrera y problemas de inconsistencia. El uso de un mecanismo de espera activa permitió una respuesta más rápida en situaciones donde los recursos se liberan rápidamente, adecuado para nuestro contexto de simulación.

V-C. Algoritmos de Calendarización

La implementación de los algoritmos de calendarización fue fundamental para el control del orden de ejecución de los hilos, que en este caso representan barcos que desean cruzar el canal. Se desarrollaron varios algoritmos estos oson:

- **Round Robin (RR):** Distribuye equitativamente el tiempo de ejecución entre todos los hilos, garantizando un acceso justo al recurso compartido.
- **Shortest Job First (SJF):** Prioriza los hilos con menor tiempo de trabajo pendiente, optimizando el tiempo de respuesta.
- **Prioridad:** Asigna prioridades a los hilos según el tipo de barco, favoreciendo los procesos más importantes, como en simulaciones de emergencias.
- **First-Come, First-Served (FCFS):** Atiende los hilos en el orden de llegada, asegurando trato justo y predecible.

- **Tiempo Real:** Responde rápidamente a emergencias, como barcos de patrulla, respetando plazos críticos de ejecución.

V-D. Manejo del Canal Simulado

El canal fue modelado como un recurso crítico, donde la sincronización de los hilos era esencial para evitar colisiones entre los barcos. La simulación incluyó la gestión de direcciones de tránsito, de modo que en un momento dado solo se permitiera el paso de barcos en una dirección. Además, se desarrollaron mecanismos de manejo de emergencias para situaciones donde barcos de alta prioridad, como patrullas, requerían un acceso prioritario al canal.

La configuración del canal se realizó mediante un archivo de configuración externo, permitiendo ajustar parámetros como la longitud del canal, la velocidad de los barcos y el método de calendarización a utilizar. Esto hizo posible adaptar la simulación a distintos escenarios de prueba y evaluar el comportamiento del sistema bajo diferentes condiciones.

V-E. Interacción con el Usuario

Se desarrolló una CLI para controlar el programa del canal y añadir nuevos barcos. La interfaz se inicia al presionar cualquier tecla seguida de Enter. Los comandos disponibles son los siguientes:

```
Comandos:
    exit: salir del programa
    r: agregar barco a la derecha
    l: agregar barco a la izquierda
    n: barcos normales
    f: barcos pesqueros
    p: barcos patrulla
```

Para correr la GUI del projecto se debe primero correr el projecto schedule-ships y luego se de ir. A ‘bin‘ y correr el programa con Python ‘python3 gui.py‘.

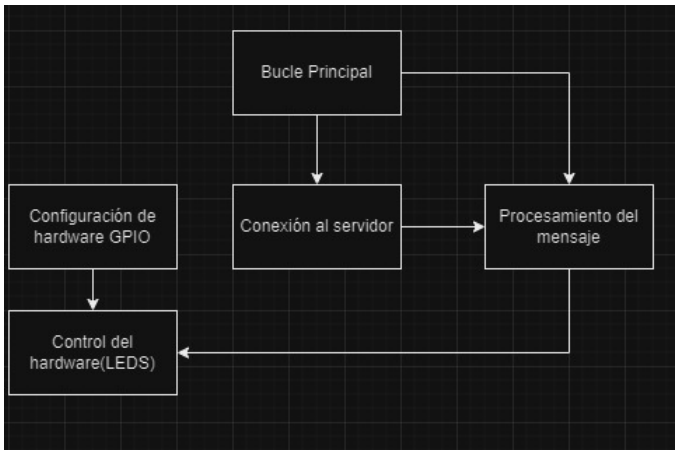
V-F. Configuración del Canal

Para configurar el comportamiento del canal, se utiliza el archivo de configuración ubicado en ‘src/canal/canal.config‘. A continuación se explican los posibles parámetros que pueden ser ajustados en este archivo:

Posibles Claves en el Archivo .config

- **length:** Define el tamaño del canal.
- **c_schedule:** Algoritmo de calendarización del canal.
- **W:** Parámetro de control de equidad o número de barcos que deben pasar en el modo W.
- **time:** Controla el tiempo o duración del semáforo.
- **speed:** Define la velocidad de tres tipos de barcos.
- **left:** Configura la línea de espera del lado izquierdo.
- **right:** Configura la línea de espera del lado derecho.
- **queuelength:** Establece la capacidad máxima de las filas de espera.
- **t_schedule:** Algoritmo de calendarización de los hilos.

V-G. Diseño del Hardware



El diagrama de hardware del código implementado en la Jetson se enfoca en la configuración de los pines GPIO para controlar los LEDs que indican la dirección de los barcos, la luz amarilla, y la posición de los barcos a la izquierda y derecha. Estos pines están configurados mediante la función `GPIO.setup()`, y cada uno tiene una función específica. Por ejemplo, el pin 40 está asociado con la dirección del barco, el pin 13 controla la luz amarilla, y los pines 11, 12, 37, y 38 se utilizan para los barcos a la izquierda y derecha, respectivamente.

El sistema también utiliza dos diccionarios (`my_channel_dict` y `my_boats_dict`) para definir las conexiones de los LEDs en diferentes canales y estados de los barcos. Estos diccionarios permiten que el código asigne correctamente los pines de GPIO a cada canal y los colores correspondientes a los barcos (verde para barcos normales, azul para pesqueros y rojo para patrullas). La función `show_leds()` utiliza esta configuración para actualizar los LEDs en tiempo real según los datos recibidos del servidor.

Además, el código procesa mensajes recibidos a través de un socket, actualizando las variables globales que controlan los LEDs. A medida que se reciben datos del servidor, se procesan y se envían señales a los pines GPIO configurados, lo que garantiza que el sistema de LEDs refleje con precisión el estado actual de los barcos y su dirección. El diagrama refleja cómo la configuración inicial de hardware está vinculada con el procesamiento de mensajes y la visualización en los LEDs [3].

VI. Instrucciones de cómo se utiliza el proyecto

Para ejecutar el proyecto de manera correcta, se deben cumplir con los siguientes requisitos:

- **Python**: Versión 3.6.9 o superior, junto con las bibliotecas **Pillow** y **Tkinter**.
- **CMake**: Versión 3.10.0 o superior.
- **Sistema operativo**: Ubuntu 18.04 o superior.

Para instalar las dependencias, puedes ejecutar el siguiente comando:

```
sudo apt install build-essential cmake ninja-build
```

```
pip install Pillow
```

Para compilar el proyecto, utiliza el siguiente conjunto de comandos desde la raíz del proyecto:

```
rm -rf build
mkdir build 4
cd build
cmake ..
make
```

Este conjunto de comandos eliminará cualquier compilación previa, creará un nuevo directorio `build`, generará los archivos de construcción con CMake y luego compilará el proyecto.

Es importante ejecutar el proyecto desde el directorio `build`, ya que los archivos de configuración necesarios (como `canal/canal.config`) estarán disponibles en este directorio. Para ejecutar el programa principal, usa:

```
cd build/bin
./sheduling-ships
```

Para correr las pruebas unitarias, utiliza el siguiente comando dentro del directorio `build`:

```
cd build
ctest --verbose
```

Este comando ejecutará todas las pruebas definidas en el proyecto, mostrando en detalle los resultados de cada una.

Correr la GUI se tiene que ejecutar primero el projecto en C sheduling-ships y luego ejecutar el siguente comando.

```
cd build/bin
python3 gui.py
```

VII. Conclusiones

- El uso de **CMake** y **CTest** resultó fundamental para la compilación y pruebas del proyecto. Aunque la configuración inicial tomó algunos días, la inversión de tiempo valió la pena, ya que permitió comprobar eficientemente las funcionalidades de la librería `CEThread` y de los algoritmos de calendarización implementados.
- Utilizar un microcontrolador con suficientes pines resulta indispensable para el correcto desarrollo, ya que es factible agotar las cantidades disponibles de pines.
- El canal debe de supervisar constantemente los hilos creados, así como los finalizados, para evitar problemas de segmentación.
- La reimplementación de la biblioteca Pthreads bajo el nombre CEThreads ha demostrado ser efectiva en permitir la ejecución simultánea de múltiples hilos y su sincronización adecuada, asegurando que las tareas se ejecuten en el orden deseado según el calendarizador aplicado.
- La implementación y comparación de diversos algoritmos de planificación, como Round Robin (RR), Prioridad, Shortest Job First (SJF), y First-Come, First-Served (FCFS), revelaron diferencias significativas en la eficiencia con la que se manejaron los hilos.
- La simulación del canal como un medio de tránsito para barcos (hilos) ha probado ser una herramienta útil para visualizar y entender la dinámica de los sistemas de hilos y los algoritmos de planificación.
- Los algoritmos de flujo implementados han evitado con éxito colisiones y han asegurado un tráfico fluido y equitativo bajo diversas condiciones.
- A pesar del éxito en la implementación de la biblioteca y los algoritmos de planificación, el proyecto enfrentó desafíos técnicos, especialmente en la interfaz de usuario y la representación gráfica de datos en tiempo real. Estas áreas representan oportunidades para futuras mejoras y refinamientos.

VIII. Segurencias y recomendaciones

- **Uso de la GUI**: Se recomienda utilizar preferentemente la interfaz gráfica en lugar de implementar el hardware, ya que la GUI permite observar el funcionamiento completo del canal de manera más clara y eficiente. La integración de hardware, aunque exitosa, resultó innecesaria para los objetivos del proyecto.
- **Alternativas de hardware**: Dado el elevado costo de la Jetson Nano, se sugiere utilizar alternativas más económicas como la Raspberry Pi, que también puede cumplir con los requisitos del proyecto.
- **Pruebas unitarias adicionales**: Sería beneficioso generar más pruebas unitarias para validar funciones específicas del canal simulado, mejorando así la cobertura de pruebas y la confiabilidad del sistema.
- **Manejo adecuado de hilos**: Es crucial crear y finalizar correctamente los hilos, asegurándose de liberar la memoria reservada para evitar errores de segmentación. La implementación de mutexes ayudó a gestionar el acceso a recursos compartidos, asegurando que las variables globales no sufrieran modificaciones inconsistentes entre hilos.
- **Mejor uso de recursos compartidos**: El uso de mecanismos de sincronización como mutexes fue clave para evitar conflictos en el acceso a variables compartidas. Se recomienda continuar utilizando esta técnica en futuros proyectos para garantizar un correcto control sobre los recursos compartidos entre hilos.

Referencias

[1] *pthreads(7) - Linux manual page*, 2024. dirección: https://man7.org/linux/man-pages/man7/pthreads.7.html.

[2] Kitware, Inc., *CMake Tutorial*, 2024. dirección: https://cmake.org/cmake/help/latest/guide/tutorial/index.html.

[3] *Jetson Nano 2GB Developer Kit User Guide*, NVIDIA Developer, 2024. dirección: https://developer.nvidia.com/embedded/learn/jetson-nano-2gb-devkit-user-guide.