Universidad Carlos III de Madrid

Grado en Ingeniería Informática

Trabajo Fin de Grado

"Herramienta Didáctica para la Programación Concurrente"

Autor

Adrián Fernández Galán

Tutor

Alejandro Calderon Mateos

Leganés, Madrid, Spain

Junio 2025

Un gran poder conlleva una gran responsabilidad

— El tío Ben

DEDICATORIA

Me gustaría comenzar agradeciendo a mis padres. Sin ellos no me encontraría terminando la carrera de Ingeniería Informática. Me han apoyado en todo momento y me han dado la libertad de elegir mi camino. Me han educado lo mejor que han podido y me han enseñado a esforzarme en todo lo que hago. Gracias a ellos soy la persona que soy hoy.

También quiero agradecer a mis amigos, desde los que llevan acompañandome desde que no sabía andar como son Lorenzo o Raúl hasta los que he conocido en los últimos años como el grupo de *Pan con Pan* o Paula, Calderón, Luis, Irene y muchos otros. He sido capaz de aprender de cada uno de ellos y me han hecho disfrutar cada momento. Sobre todo agradecer a Paula, que me ha hecho equivocarme mil y una veces y aprender de cada una de ellas.

Quería hacer especial énfasis en César. Es un amigo que no solo me ha apoyado en todo lo que ha podido, sino que he podido discutir con él todas aquellas ideas que no terminaba de encajar en este proyecto. Proporcionándome ideas originales y aportándome críticas constructivas constantemente. Sin él este proyecto no tendría la calidad que ha llegado a alcanzar.

Otra persona que merece un agradecimiento es mi tutor Alejandro. Que aunque no ha sido necesario que nos reunamos con demasiada frecuencia, las veces que nos hemos reunido han sido muy proliferas. Siempre ha estado dispuesto a ayudarme, aportando ideas frescas y soluciones a problemas, además de guiarme en el desarrollo de la memoria que ahora mismo estás leyendo.

Para terminar, aunque no por ello menos importante, quería agradecer a Luis Daniel, Alvaro y Jose Antonio. Cada uno de ellos han propuesto soluciones a ciertos problemas que les he ido comentando durante el desarrollo del proyecto. Además Luis Daniel ha sido fundamental gracias a la plantilla que ahora mismo estoy usando para escribir esta memoria.

RESUMEN

La concurrencia es una herramienta muy potente para mejorar el rendimiento de los programas desarrollados, sin embargo tiene una curva de aprendizaje muy pronunciada y su programación puede inducir a errores con facilidad.

Es por esto que proponemos crear un depurador didáctico enfocado a la programación concurrente, que permita a estudiantes entender con más facilidad y desarrollar código más libre de errores.

Esta herramienta estará centrada en que hasta los programadores más primerizos sean capaces de desarrollar código C, centrándose en descubrir las características principales de los programas concurrentes. Para ello el estudiante podrá moverse por libremente por la ejecución del programa desarrollado y observar cómo el estado del sistema cambia durante la ejecución del mismo.

Palabras clave: Concurrencia • Herramienta didáctica • Depurador

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

En este capítulo se presenta el proyecto, desde la motivacion que lo ha generado (Sección 1.1, *Motivación*) hasta los objetivos que se pretenden alcanzar (Sección 1.2, *Objetivos*), además de describir la estructura del documento y cómo se detallan los capítulos que lo componen (Sección 1.3, *Estructura del documento*).

1.1. Motivación

Este proyecto se ve motivado por las dificultades vividas durante el desarrollo y aprendizaje de la asignatura Sistemas Operativos. Durante esta asignatura se nos presentó el concepto de la programación concurrente y se nos evaluó de forma práctica a través de una herramienta basada en un productor-consumidor multihilo escrito en *C*.

Para programar concurrentemente es necesario entender qué son los procesos, los hilos y cómo se relacionan entre sí. Esta última parte es la más complicada de entender y no comprenderla supone caer en problemas de condiciones de carrera, inanición, etc. En lo personal, me costó desarrollar la práctica pedida, debido a ciertos problemas que surgían por no sincronizar correctamente los hilos y no ser capaz de encontrorar la causa de estos problemas.

Durante este periodo estuve ayudando a aquellos compañeros que estaban teniendo los mismos problemas que yo, lo que me permitió ver qué cosas específicas eran, tanto en mi caso como en el de otras personas, más problemáticas y cómo podrían solucionarse o mermarse.

Por todo esto, decidí que sería interesante realizar una herramienta que facilitara el aprendizaje de los principales conceptos de la programación concurrente a través de la visualización de los diferentes estados de la práctica de Sistemas Operativos. Por esta razón, la herramienta debía de permitir escribir código en *C* y poder observar cómo se

comportan los procesos y los hilos, según el código que se haya escrito.

1.2. Objetivos

Teniendo en cuenta la motivación presentada en la Sección 1.1, *Motivación*, se definen unos objetivos principales (**Obj.1** y **Obj.2**) y unos objetivos secundarios (**Obj.1.X** y **Obj.2.X**) que se pretenden alcanzar con este proyecto:

- **Obj.1**: La herramienta debe de compilar, ejecutar y depurar código concurrente en *C*
  - **Obj.1.1**: La depuración debe de permitir al usuario controlar la ejecución de los hilos.
  - **Obj.1.2**: La depuración debe de permitir al usuario observar el estado de los hilos.
  - **Obj.1.3**: La herramienta debe de permitir realizar las acciones básicas de un depurador, tales como *step over*, *step into*, *step out*, *continue*, *breakpoint*, etc.
- **Obj.2**: La herramienta debe de tener un enfoque didáctico
  - **Obj.2.1**: Debe de abstraerse de la complejidades en la compilación, memoria de los procesos y llamadas al sistema, arquitectura de la máquina o sistema operativo.
  - **Obj.2.2**: Debe de tener una interfaz gráfica que permita al usuario interactuar con la herramienta.
  - **Obj.2.3**: Debe de tener las funcionalidadades básicas para permitir al usuario entender cómo se comportan los hilos.

1.3. Estructura del documento

El documento se estructura de la siguiente manera:

- Capítulo 1, *Introducción*: presenta brevemente el proyecto, sus motivaciones y una descripción de la estructura del documento.
- Capítulo 2, *Estado del arte*: describe el estado actual de los depuradores y comparandolos entre si.
- Capítulo 3, *Análisis*: presenta el análisis de los requisitos y la arquitectura del sistema.

- Capítulo 4, *Diseño*: presenta el diseño de la herramienta, a través de las diferentes decisiones tomadas y los diferentes diagramas que describen el sistema.
- Capítulo 5, *Implementación y Despliegue*: presenta la implementación de la herramienta, así como el proceso de despliegue y las herramientas utilizadas.
- Capítulo 6, *Validación y Verificación*: presenta los casos de prueba realizados para comprobar la funcionalidad de la herramienta.
- Capítulo 7, *Planificación y presupuesto*: presenta la planificación del proyecto, así como el presupuesto del mismo.
- Capítulo 8, *Marco regulatorio y entorno socio-económico*: presenta el marco regulador del proyecto y el efecto en el entorno socio-economico.
- Capítulo 9, *Conclusiones y Trabajo Futuro*: presenta las conclusiones del proyecto y el trabajo realizado, así como los futuros trabajos que se pueden realizar.

CAPÍTULO 2

ESTADO DEL ARTE

En este capítulo se describirá el estado actual de las tecnologías y herramientas relacionadas con el proyecto. Se explorará la importancia de la concurrencia y las distintas herramientas de depuración de código que permitan analizar el comportamiento de programa concurrente y cómo sus características casan con los objetivos del proyecto (Sección 1.2, *Objetivos*).

2.1. Concurrencia y su importancia

La concurrencia [1] se define como la capacidad de un sistema para ejecutar múltiples procesos de forma simultánea o intercalada, según lo determine el planificador del sistema operativo. Gracias a este mecanismo, un programa puede gestionar diversas tareas como la interacción con el usuario, el procesamiento de datos y la comunicación en red sin que el bloqueo de una impida el progreso de las demás, lo que optimiza el rendimiento general y reduce el tiempo de respuesta.

Este enfoque no solo mejora la eficiencia en la ejecución, sino que también incrementa la escalabilidad, especialmente en entornos distribuidos, y contribuye a disminuir el consumo energético mediante una distribución adecuada de las tareas y la reducción de las frecuencias de reloj [2]. Además, la desaceleración en el avance de la Ley de Moore [3] ha subrayado la necesidad de explorar otros medios para aumentar el rendimiento y la eficiencia de los sistemas [4].

Sin embargo, diversas investigaciones sobre la paralelización en tiempo de compilación han evidenciado que extraer de forma automática el paralelismo de programas secuenciales resulta un desafio considerable [4]. Esto implica que, para aprovechar plenamente las ventajas de los sistemas multicore, es indispensable que los desarrolladores realicen una reestructuración manual del código para transformarlo en una aplicación

concurrente.

En este contexto, formar nuevos profesionales con conocimientos profundos en concurrencia se presenta como una estrategia clave para mejorar el rendimiento de las aplicaciones y reducir los costes energéticos, consolidándose como un factor competitivo esencial en el desarrollo de software actual y futuro.

2.2. Depuradores de programas concurrentes

El paralelismo presenta nuevos desafíos a la hora de desarrollar programas, ya que la ejecución de múltiples procesos puede dar lugar a errores difíciles de detectar y reproducir. Estos errores pueden incluir condiciones de carrera, interbloqueos y otros problemas relacionados con la sincronización de hilos. Por tanto, es fundamental comprender exactamente qué está ocurriendo en el programa para poder identificar y corregir estos errores. Para esto existen herramientas que permiten analizar el comportamiento de los programas como lo son los depuradores.

Un depurador [5] es un programa usado para probar y depurar otros programas. Los depuradores suelen permitir colocar puntos de interrupción, saltar a través de distintas partes del código e inspeccionar el estado de la memoria, registros de la Central Processing Unit (CPU) y la pila de llamadas. Estos depuradores suelen ser integrados en los entornos de desarrollo, como *Visual Studio* [6] o *Eclipse* [7], aunque también existen depuradores basados en línea de comandos (Command Line Interface (CLI)) como *GDB* [8] o *LLDB* [9].

Sin embargo, los depuradores tradicionales no suelen estar diseñados para programas concurrentes. En un programa concurrente [1] existen múltiples procesos, que pueden ser ejecutados en paralelo o de forma intercalada, que interactuan con los recursos, lo que obliga a que los depuradores tengan funcionalidades específicas para soportar este tipo de ejecuciones.

Los depuradores de programas concurrentes son capaces de controlar e inspeccionar cada uno de los procesos que existen en el programa, aunque el nivel de granularidadd y funcionalidad dependen de la herramienta. A continuación se explorarán algunas de las herramientas usadas en la actualidad.

2.2.1. GDB

GDB es un depurador de código abierto creado por la *Free Software Foundation* en 1986 [8]. Este depurador solo está disponible para sistemas *Unix* [10] , aunque simulando este sistema en otros sistemas operativos se puede utilizar [11]. Es importante destacar que esta herramienta ocupa en disco únicamente en torno a 12.22 MB. GDB una herramienta con la que se interactua a través de su CLI o Text User Interface (TUI) y permite controlar

la ejecución de programas y analizar su estado en tiempo de ejecución. GDB es capaz de depurar programas escritos en C, C++, D, Fortran, Go, Pascal, Rust y ensamblador [8].

GDB permite colocar puntos de interrupción, inspeccionar la memoria, los registros de la CPU y la pila de llamadas e incluso revisar el código ensamblador del programa y modificar el valor de las variables en tiempo de ejecución. Además presenta ciertas características avanzadas como la posibilidad de crear scripts en Python para automatizar tareas [12], utilizar lenguaje máquina para devolver respuestas procesadas y entendibles por las máquinas [13] controlar la ejecución del programa hacia atrás, aunque esta última solo está disponible para programas mono-hilo [14].

Sus puntos fuertes son la multitud de funcionalidades que ofrece para depurar y obtener información sobre el estado del programa y posibilidad de automatizar sus procesos a través de la Application Programming Interface (API) de Python o su interfaz máquina-humano, lo que permite crear nuevas herramientas que extiendan las características de este depurador y se adapten a las necesidades del usuario. Ejemplos de ello son CLion (Subsección 2.2.4, *CLion*) o herramientas menos cononidas como Blink [15], un depurador de entornos mixtos que utiliza la composición para depurar programas que utilizan varios códigos.

CÓDIGO 2.1. Muestra del depurador GDB

```
(gdb) break 50
Breakpoint 1 at 0x1324: file prueba1.c, line 50.
(gdb) break 60
Breakpoint 2 at 0x1376: file prueba1.c, line 60.
(gdb) run
Starting program: /home/adrian/Documents/Trabajo-de-Fin-de-Grado/
   examples/prueba1.o
[New Thread 0x7ffff7a006c0 (LWP 39887)]
[Switching to Thread 0x7ffff7a006c0 (LWP 39887)]

Thread 2 "prueba1.o" hit Breakpoint 1, greet (argumento=0x0) at
   prueba1.c:50
50          printf("Hello, welcome to the program!\n");
(gdb) continue
Continuing.
[New Thread 0x7ffff70006c0 (LWP 39888)]
Hello, welcome to the program!
[Thread 0x7ffff7a006c0 (LWP 39887) exited]
[Switching to Thread 0x7ffff70006c0 (LWP 39888)]

Thread 3 "prueba1.o" hit Breakpoint 2, add (argumento=0x7fffffffdd04
   ) at prueba1.c:60
60      }
(gdb) list
55      void* add(void* argumento) {
56          float flags = 3.14;
57          AddArgs* args = (AddArgs*)argumento;
```

```
58          args->result = args->a + args->b;
59          return NULL;
60      }
(gdb)
```

2.2.2. LLDB

LLDB es un depurador de código abierto creado por LLVM en 2003, siendo construido a través de un conjunto de componentes reusables que son ampliamentes usados en librerías existentes de LLVM, como Clang o el desensamblador LLVM [9], lo que hace que tan solo pese en torno a 4 MB. Aunque LLDB es el depurador por defecto de Xcode en macOS, también está disponible para sistemas Unix y Windows [9].

LLDB es un depurador para programas en C y C++ con una CLI y consta de funcionalidades similares a GDB, como colocar puntos de interrupción, inspeccionar la memoria, etc. Por esta razón existe una correspondencia entre las funcionalidades de GDB y LLDB que permite a los usuarios de GDB (Subsección 2.2.1, *GDB*) cambiar a LLDB sin problemas [9], aunque pueden existir funcionalidades de GDB que no existan en LLDB. Además, al ser un depurador moderno, ofrece funcionalidades como autocompletado de comandos e historial de comandos.

LLDB se caracteriza por tener una API de Python más completa que la de GDB, la cual permite desde crear scripts para automatizar tareas hasta controlar programáticamente la ejecución del programa [16].

Por lo tanto, LLDB destaca por su integración con LLVM y la posibilidad de controlar el depurador a través de Python, lo que permite a los usuarios crear herramientas personalizadas y adaptar el depurador a sus necesidades.

CÓDIGO 2.2. Muestra del depurador LLDB

```
    (lldb) breakpoint set --line 50
Breakpoint 1: where = prueba1.o`greet + 16 at prueba1.c:50:5,
   address = 0x0000000000001324
(lldb) breakpoint set --line 60
Breakpoint 2: where = prueba1.o`add + 60 at prueba1.c:60:1, address
   = 0x0000000000001376
(lldb) run
Process 8704 launched: '/home/adrian/Documents/Trabajo-de-Fin-de-
   Grado/examples/prueba1.o' (x86_64)
Process 8704 stopped
* thread #2, name = 'prueba1.o', stop reason = breakpoint 1.1
    frame #0: 0x0000555555555324 prueba1.o`greet(argumento=0
       x0000000000000000) at prueba1.c:50:5
   47
   48   // Function to print a greeting message
   49   void* greet(void* argumento) {
```

```
-> 50       printf("Hello, welcome to the program!\n");
   51       return NULL;
   52   }
   53
(lldb) continue
Process 8704 resuming
Hello, welcome to the program!
Process 8704 stopped
* thread #3, name = 'prueba1.o', stop reason = breakpoint 2.1
    frame #0: 0x0000555555555376 prueba1.o`add(argumento=0
       x00007fffffffdd64) at prueba1.c:60:1
   57       AddArgs* args = (AddArgs*)argumento;
   58       args->result = args->a + args->b;
   59       return NULL;
-> 60   }
   61
   62   // Function to print an array
   63   void* printArray(void* xd) {
```

2.2.3. RR

RR es un depurador ligero de código abierto creado en 2013 por Mozilla [17]. Este depurador utiliza GDB (Subsección 2.2.1, *GDB*) para depurar programas, sin embargo añade la funcionalidad de rebobinar las ejecuciones de cualquier tipo de programa, siendo uno de ellos un programa concurrente, mediante la grabación de la ejecución del programa y la reproducción de esta grabación. Al grabar una ejecución y poder reproducirla convierte la ejecución concurrente, que es indeterminista por naturaleza, en determinista, lo que facilita la depuración [18].

Para poder grabar la ejecución es necesario ejecutar `rr record <program>` y para reproducir esta grabación `rr replay`, tras esto se mostrará la CLI de la herramienta. RR utiliza la CLI de GDB para interactuar con el usuario, manteniendo los mismos comandos, pero añadiendo comandos específicos para rebobinar.

Esto hace a RR un depurador muy útil por la flexibilidad y las funcionalidades que le ofrece GDB y la capacidad de convertir proceso no deterministas endeterministas, sin embargo añade un *overhead* en la memoria en la ejecución del programa de entre 1.5 y 2 veces y la ralentización que supone ejecutar programas paralelos en una emulación de un solo núcleo[18].

2.2.4. CLion

CLion es un Integrated Development Environment (IDE) de propiedad de JetBrains, enfocado en la programación en C y C++. Esta aplicación necesita de una licencia para su uso de entre 60€a 100€, aunque existe una prueba gratuita de 30 días [19]. Cabe destacar

que CLion es multiplataforma, disponible para Windows, macOS y Linux y ocupa un espacio de 3.5 GB en disco [20].

CLion incluye una opción para depurar a través de GDB (Subsección 2.2.1, *GDB*) o LLDB (Subsección 2.2.2, *LLDB*), elegible a través de un menú contextual, mediante una interfaz gráfica y extendible con Valgrind y ThreadSanitizer. En este modo depuración se pueden colocar puntos de interrupción, ejecutar el programa, parar el programa, y realizar *step over* y *step into* [21].

La depuración muestra multitud de información, organizada en pestañas, como la pila de llamadas, variables locales, variables globales, la salida del programa o inspeccionar la memoria y los registros de la CPU. También es posible ver el valor de las variables en tiempo real, y modificarlas durante la ejecución del programa [22].

Por lo que las claves de CLion son su integración de un depurador con una interfaz gráfica, la variedad de funcionalidades de depuración que permite y la integración con diferentes herramientas de análisis de código.

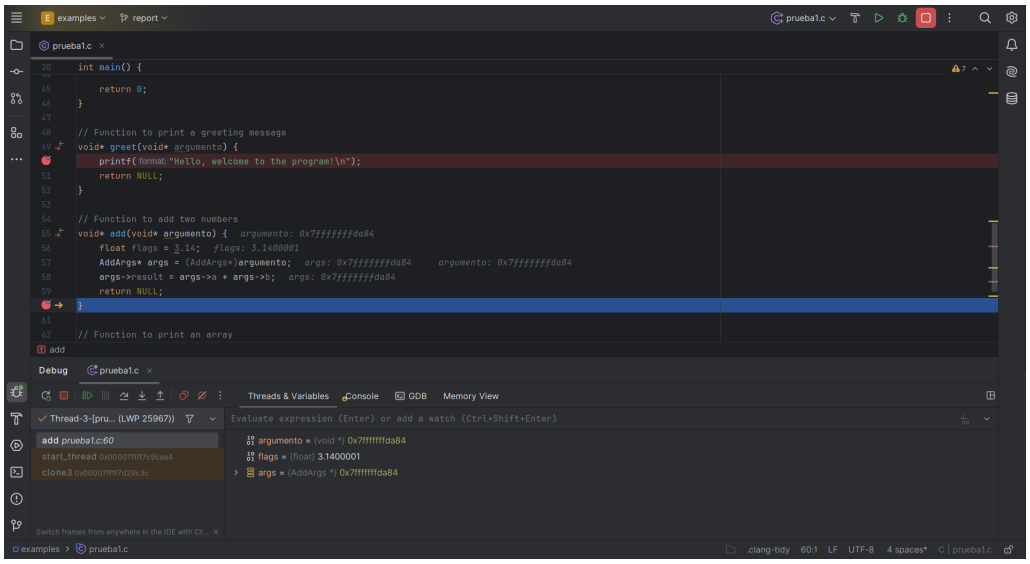

Fig. 2.1. Muestra del depurador de CLion

2.2.5. Seer

Seer es un depurador de código abierto, hecho principalmente en C++ y QTile para la interfaz gráfica, que envuelve las funcionalidades de GDB. Este depurador solo se encuentra disponible en sistemas Unix y ocupa un espacio de 47.4 MB en disco.

Seer se caracteriza por tener una interfaz gráfica que permite visualizar la información de una forma más amigable y accesible que la CLI de GDB. Esta interfaz gráfica tiene una barra de herramientas con botones para las funcionalidades más comunes, como colocar puntos de interrupción, ejecutar el programa, parar el programa, y realizar *step over* y *step into*. Además, tiene diferentes secciones para visualizar información sobre la pila

de llamadas, los distintos hilos en ejecución y las variables locales y globales, pudiendo pulsar en ellos para mostrar más información sobre ellos.

Seer destaca por su ligereza, su interfaz gráfica, su rendimiento y la multitud de funcionalidades que ofrece.

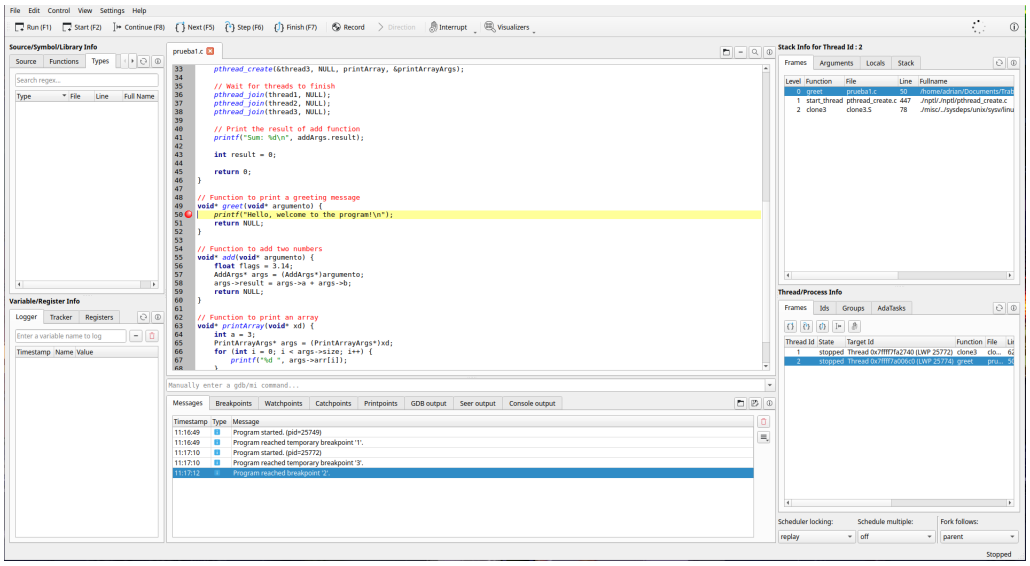

Fig. 2.2. Muestra del depurador de Seer

2.3. Otras Herramientas

Además de los depuradores de programas concurrentes, existen otras herramientas que permiten analizar el comportamiento de un programas concurrentes y detectar errores debidos a el uso incorrecto de la concurrencia. A continuación se explorarán algunas de estas herramientas.

2.3.1. Helgrind

Helgrind es una de las herramientas del grupo Valgrind que detecta errores de sincronización en programas en C y C++ que usan las primitivas POSIX [23]. Estas abstracciones POSIX son: hilos compartiendo un espacio de direccionamiento común, creación de hilos, unión de hilos, salida de hilos, cerrojo (mutex), variable de condición y barreras. Helgrind detecta tres clases de errores: mal uso del API POSIX, potenciales interbloqueos, y condiciones de carrera [24]. Estos resultados suelen ser irreproducibles, lo que complica la tarea de encontrar otras maneras para solucionar los errores.

Helgrind monitorea el orden en el que los distintos hilos adquieren el cerrojo, permitiendo detectar potenciales interbloqueos y condiciones de carrera, lo que resulta útil dado que algunos de ellos pueden haberse reproducido durantes las pruebas del programa. Sin embargo, este monitoreo puede ralentizar la ejecución del programa hasta en un factor de

100 [24].

Para usar esta herramienta se debe de compilar con la opción `-pthread` y ejecutar el programa con la opción `-tool=helgrind`. En el caso en el que Helgrind detecte algún problema en el programa, se mostrará un mensaje de error en la salida estándar, esto se puede observar en la siguiente figura.

CÓDIGO 2.3. Muestra de salida de Helgrind al detectar un error de condición de carrera [25]

```
gcc -Wall -g -pthread helgrind_threads_race.c -o
   helgrind_threads_race valgrind -v  --tool=helgrind  ./
   helgrind_threads_race

==8297== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 1 from
    1)
==8297==
==8297== 1 errors in context 1 of 1:
==8297==
   ------------------------------------------------------------------
==8297==
==8297== Possible data race during write of size 4 at 0xBEE21570 by
   thread #2
==8297== Locks held: none
==8297==    at 0x8048592: increment_counter (helgrind_threads_race.c
   :12)
==8297==    by 0x80485C3: counter_thread (helgrind_threads_race.c
   :20)
==8297==    by 0x402DD35: ??? (in /usr/lib/valgrind/
   vgpreload_helgrind-x86-linux.so)
==8297==    by 0x405AD4B: start_thread (pthread_create.c:308)
==8297==    by 0x415DB8D: clone (clone.S:130)
==8297==
==8297== This conflicts with a previous write of size 4 by thread #1
==8297== Locks held: none
==8297==    at 0x8048592: increment_counter (helgrind_threads_race.c
   :12)
==8297==    by 0x8048682: main (helgrind_threads_race.c:37)
==8297==
```

Helgrind se complementa con las funcionalidades tradicionales de los depuradores dado que se pueden encontrar errores en tiempo de compilación y de forma automática, además de encontrar errores que tal vez aún no se hayan manifestado.

2.3.2. ThreadSanitizer

ThreadSanitizer es una herramienta que detecta condiciones de carrera para código C y C++. Está formado por una módulo de inyección de código y una librería de tiempo de ejecución. Esta herramienta puede ralentizar la ejecución en torno a un factor de 5 a 20 e

introducir un sobrecarga de memoria de un factor de 5 a 10 [26].

A diferencia de Helgrind (Subsección 2.3.1, *Helgrind*), ThreadSanitizer solo detecta condiciones de carrera en tiempo de ejecución, lo que significa que si durante la ejecución no se ha recorrido cierto fragmento de código no va poder detectar condiciones de carrera que corresponda con ese fragmento [27].

Para usar esta herramienta se debe de compilar con la opción `-pthread` y ejecutar el programa con la opción `-fsanitize=thread`. Es importante destacar que esta compilación debe de contener todo el código del programa, incluyendo las librerías que se usen, en caso contrario puede detectar falso positivo. Tras esta compilación la herramienta devolverá en la salida estándar los errores que haya detectado, como se muestra en la siguiente figura [26].

CÓDIGO 2.4. Muestra de salida de ThreadSanitizer al detectar un error de condición de carrera [27]

```
g++ simple_race.cc -pthread -fsanitize=thread -o ./a.out
==================
WARNING: ThreadSanitizer: data race (pid=26327)
  Write of size 4 at 0x7f89554701d0 by thread T1:
    #0 Thread1(void*) simple_race.cc:8 (exe+0x000000006e66)

  Previous write of size 4 at 0x7f89554701d0 by thread T2:
    #0 Thread2(void*) simple_race.cc:13 (exe+0x000000006ed6)

  Thread T1 (tid=26328, running) created at:
    #0 pthread_create tsan_interceptors.cc:683 (exe+0x00000001108b)
    #1 main simple_race.cc:19 (exe+0x000000006f39)

  Thread T2 (tid=26329, running) created at:
    #0 pthread_create tsan_interceptors.cc:683 (exe+0x00000001108b)
    #1 main simple_race.cc:20 (exe+0x000000006f63)
==================
```

2.4. Comparativa de Depuradores

Con el objetivo de encontrar cual de las herramientas descritas en las secciones anteriores es la que mejor se adapta a las necesidades del proyecto, se ha realizado una comparativa de las características de cada una de ellas.

Las características que se tendrán en cuenta para la comparación son las siguientes:

- **Plataforma**: Sistema operativo en el que está disponible la herramienta.
- **Interfaz**: Tipo de interfaz que ofrece la herramienta.

- **Func. Clave**: Funcionalidades clave que se diferencian del resto de herramientas.
- **Integración**: Capacidad de la herramienta para integrarse con otras herramientas.
- **Usabilidad**: Facilidad con la que el usuario puede aprender a usar la herramienta.
- **Coste**: Precio de la herramienta.
- **Almacenamiento**: Espacio que ocupa la herramienta en disco.
- **Arquitecuras**: Arquitecturas de CPU que soporta la herramienta.

Para el caso de la **Usabilidad** se han tenido en cuenta el número de botones, número de pestañas, número de secciones y el acceso a las anteriores características en el caso de Graphical User Interface (GUI). Para el caso de CLI se tendrán en cuenta la complejidad de estos comandos y características adicionales como autocompletado o sugerencias. De esta manera se han definido los siguientes valores:

- GUI
  - Alta: Entre 1 y 5 botones accesibles, 3 o menos secciones y pestañas.
  - Media: Entre 6 y 10 botones accesibles, 4 o 5 secciones y pestañas.
  - Baja: Más de 10 botones accesibles, más de 5 secciones y pestañas.
- CLI
  - Alta: Entre 20 y 30 comandos sencillos, autocompletado y sugerencias.
  - Media: Más de 30 comandos sencillos y autocompletado.
  - Baja: Más de 30 comandos complejos y sin autocompletado.

2.4.1. Comparativa de Depuradores de Programas Concurrentes

TABLA 2.1
COMPARATIVA DE DEPURADORES

Depurador	GDB	LLDB	RR	CLion	SEER	*Propuesta*
Plataforma	Unix	Multiplataforma	Unix	Multiplataforma	Unix	*Web*
Interfaz	CLI	CLI	CLI	GUI + IDE	GUI	*GUI*
Func. Clave	Muy Versátil	Versátil + Moderno	Rebobinado	Integrado en IDE	Visual	*Abstracción*
Integración	MI	API Python	MI	×	×	×
Usabilidad	Baja	Media	Baja	Media	Baja	Alta
Coste	Gratuita	Gratuita	Gratuita	60€- 100€/año	Gratuita	*Gratuita*
Almacenamiento	12.22 MB	4 MB + Módulos	50MB	3.5 GB	5.8 MB	*0 MB*
Arquitecturas	Todas	x86, ARM, MIPS64	ARM y x86 Modernas	Todas	Todas	*No Afecta*

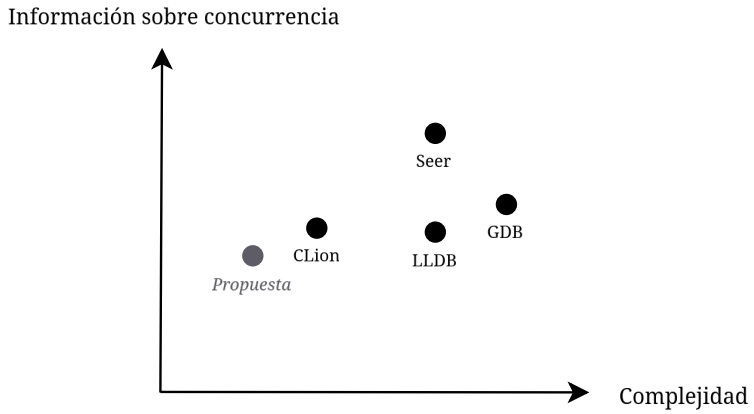

Fig. 2.3. Comparativa de depuradores

2.4.2. Conexión con los Objetivos del Proyecto

A partir del análisis de las herramientas existentes para la depuración de programas concurrentes, se observa que, si bien cada una presenta características distintivas, ninguna satisface completamente el objetivo 2 (9.1). La única excepción parcial es CLion (Subsección 2.2.4, *CLion*), que se acerca más a cumplir este objetivo, aunque carece de una versión gratuita, lo que limita su accesibilidad. Por ello, y basándose en los conocimientos adquiridos durante esta investigación, se plantea el desarrollo de una herramienta que no solo abstraiga las funcionalidades de los depuradores actuales, sino que también incorpore mecanismos didácticos que faciliten el aprendizaje de la depuración en entornos concurrentes de manera intuitiva y accesible para principiantes.

En el capítulo Capítulo 3, *Análisis* se detallará cómo se conseguirán alcanzar estos objetivos y qué herramientas se utilizarán para ello.

CAPÍTULO 3

ANÁLISIS

En este capítulo se detallará el sistema a desarrollar describiendo brevemente la propuesta a desarrollar, sus requisitos y los casos de uso que se han identificado.

3.1. Descripción del proyecto

Este proyecto tiene como objetivo construir una herramienta que permita a los estudiantes que comienzan a crear programas concurrentes depurar su código abstrayendose de ciertas complejidades ajenas a la programación concurrente y enfocandose en la sincronización de los hilos y procesos.

Como se mencionó durante el Capítulo 2, *Estado del arte*, los depuradores que permiten la depuración concurrente no están enfocados en el aprendizaje ni la abstracción, sino en ofrecer todo tipo de funcionalidades al usuario para que él, que entiende al completo todos los factores que pueden inducir a un error, encuentre el problema. Es por esto que se propone una herramienta que permita a los estudiantes entender cómo se comportan los hilos y procesos de forma sencilla y visual.

3.2. Requisitos

En esta sección se describirán los requisitos del sistema. Para la tarea de la especificación de los requisitos se ha seguido el estándar de IEEE [28] y la guía práctica sobre la redacción de requisitos de COSEC [29]. Estas prácticas describen que una buena especificación de requisitos de software debe explicar la funcionalidad del software, rendimiento del sistema, interfaces con otros sistemas y otras características no funcionales.

Para tranformar una necesidad a un requisito se han seguido las siguientes caracterís-

ticas:

- **C1 - Necesario**: Debe definir una capacidad, característica o restricción esencial para satisfacer una necesidad del sistema.
- **C2 - Apropiado**: Debe estar definido con el nivel de detalle adecuado para el contexto en el que se aplicará.
- **C3 - No ambiguo**: Debe ser claro y comprensible para que no haya interpretaciones múltiples.
- **C4 - Completo**: Debe proporcionar toda la información necesaria para ser implementado sin suposiciones adicionales.
- **C5 - Singular**: Debe expresar una única idea o necesidad sin combinar múltiples conceptos.
- **C6 - Factible**: Debe ser posible de implementar dentro de las restricciones de costos, tiempo y tecnología.
- **C7 - Verificable**: Debe ser medible y comprobable mediante inspección, análisis, prueba o demostración.
- **C8 - Correcto**: Debe representar fielmente la necesidad o el requisito de nivel superior del que se deriva.
- **C9 - Conforme**: Debe seguir un formato y un estilo estandarizado, evitando términos vagos o ambiguos.

Para comenzar con la especificación de los requisitos se parte de las necesidades de los usuarios, que se transforman a requisitos, de la manera antes mencionada, para convertirlos en requisitos de usuario, que se detallan en Subsección 3.2.1, *Requisitos de usuario*. Tras especificar los requisitos de usuario se derivarán los requisitos de sistema, que se detallarán en Subsección 3.2.2, *Requisitos de sistema*, que proporcionan una información más concisa sobre el funcionamiento y las diferentes características del sistema. Estos requisitos serán la base para la construcción del sistema y servirán como guía para el desarrollo del mismo.

3.2.1. Requisitos de usuario

En esta sección se detalla la descripción de cada uno de los requisitos de usuario de este proyecto. Estos requisitos reflejan las necesidades y restricciones impuestas por los usuarios finales. Estos requisitos pueden clasificarse en dos tipos:

- **Capacidades**: Representan las funcionalidades que el usuario necesita para lograr un objetivo o resolver un problema.

- **Restricciones**: Representan las limitaciones impuestas por el usuario o el entorno en el que se desarrolla el sistema.

Cada requisito tiene un identificador único que sigue el siguiente formato: UR-YY-XX, donde YY es el tipo de requisito, *CA* para capacidades y *RE* para restricciones, y XX es un número secuencial que identifica el requisito y comienza en 01.

Para la especificación los requisitos se usará la siguiente plantilla:

Cada requisito de usuario se identifica con un ID que sigue el formato *RU-YY-XX*, donde *YY* identifica el tipo del requisito, ya sea capacidad (*CA*) o restricción (*RE*); y *XX* identifica el número secuencial del requisito dentro de su tipo, empezando en *01*.

La tabla 3.1 contiene la plantilla utilizada para la especificación de los requisitos, incluyendo la descripción de cada atributo.

TABLA 3.1
PLANTILLA DE REQUISITO DE USUARIO

RU-YY-XX	
Descripción	Especificación del requisito en un lenguaje claro, conciso y no ambiguo.
Necesidad	Prioridad del requisito para el usuario (*Esencial*, *Conveniente* u *Opcional*).
Prioridad	Prioridad del requisito para el desarrollador (*Alta*, *Media* o *Baja*).
Estabilidad	Indica si el requisito se modifica durante el desarrollo (*No cambia*, *No cambia* o *Muy inestable*).
Verificabilidad	Capacidad de comprobar la validez del requisito (*Alta*, *Media* o *Baja*).

TABLA 3.2
REQUISITO RU-CA-01

RU-CA-01	
Descripción	El sistema debe de mostrar información sobre la ejecución de un programa concurrente escrito en C.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.3
REQUISITO RU-CA-02

RU-CA-02	
Descripción	El usuario debe ser capaz de escribir código.
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.4
REQUISITO RU-CA-03

RU-CA-03	
Descripción	El editor debe resaltar la sintaxis del código en C.
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.5
REQUISITO RU-CA-04

RU-CA-04	
Descripción	El usuario debe ser capaz de compilar el código escrito con anterioridad.
Necesidad	Opcional
Prioridad	Alta
Estabilidad	Cambiante
Verificabilidad	Alta

TABLA 3.6
REQUISITO RU-CA-05

RU-CA-05	
Descripción	El usuario debe ser capaz de ejecutar el código compilado.
Necesidad	Opcional
Prioridad	Media
Estabilidad	Cambiante
Verificabilidad	Media

TABLA 3.7
REQUISITO RU-CA-06

RU-CA-06	
Descripción	El usuario debe ser capaz de depurar el código escrito.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Cambiante
Verificabilidad	Baja

TABLA 3.8
REQUISITO RU-CA-07

RU-CA-07	
Descripción	El sistema debe de avisar al usuario si se encuentran errores de concurrencia.
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	Cambiante
Verificabilidad	Media

TABLA 3.9
REQUISITO RU-CA-08

RU-CA-08	
Descripción	El usuario debe ser capaz de parar la ejecución del programa en cualquier momento.
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.10
REQUISITO RU-CA-09

RU-CA-09	
Descripción	El usuario debe ser capaz de depurar paso a paso.
Necesidad	Conveniente
Prioridad	Media
Estabilidad	Cambiante
Verificabilidad	Alta

TABLA 3.11
REQUISITO RU-CA-10

RU-CA-10	
Descripción	El usuario debe ser capaz de rebobinar la ejecución del programa.
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.12
REQUISITO RU-CA-11

RU-CA-11	
Descripción	El usuario debe ser capaz de ver los hilos y procesos en ejecución.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.13
REQUISITO RU-CA-12

RU-CA-12	
Descripción	El usuario debe ser capaz de ver el estado de las variables de cada uno de los hilos.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.14
REQUISITO RU-CA-13

RU-CA-13	
Descripción	El usuario debe ser capaz de ver la evolución de los hilos y procesos.
Necesidad	Esencial
Prioridad	Media
Estabilidad	Cambiante
Verificabilidad	Alta

TABLA 3.15
REQUISITO RU-RE-01

RU-RE-01	
Descripción	El sistema debe ser capaz de compilar código C.
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.16
REQUISITO RU-RE-02

RU-RE-02	
Descripción	El sistema debe ser capaz de ejecutar código C.
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.17
REQUISITO RU-RE-03

RU-RE-03	
Descripción	El sistema debe ser capaz de depurar código C.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Baja

TABLA 3.18
REQUISITO RU-RE-04

RU-RE-04	
Descripción	El sistema debe de abstraerse de las diferencias al usar la herramienta en distintos sistemas operativos.
Necesidad	Esencial
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.19
REQUISITO RU-RE-05

RU-RE-05	
Descripción	El sistema debe de abstraerse de las diferencias al usar la herramienta en distintas arquitecturas.
Necesidad	Esencial
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Baja

TABLA 3.20
REQUISITO RU-RE-06

RU-RE-06	
Descripción	El sistema debe de abstraerse de la elección del compilador.
Necesidad	Esencial
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.21
REQUISITO RU-RE-07

RU-RE-07	
Descripción	El sistema debe de abstraerse de direcciones de memoria de las variables, hilos y procesos.
Necesidad	Esencial
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta

TABLA 3.22
REQUISITO RU-RE-08

RU-RE-08	
Descripción	El sistema debe de abstraerse de depurar cualquier tipo de fragmento de código que no pertenezca al usuario.
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Media

TABLA 3.23
REQUISITO RU-RE-09

RU-RE-09	
Descripción	Un mal uso de la herramienta no debe de comprometer la seguridad del sistema.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Baja

TABLA 3.24
REQUISITO RU-RE-10

RU-RE-10	
Descripción	El sistema debe de ser multiplataforma.
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta

3.2.2. Requisitos de sistema

Esta sección detalla la descripción de cada uno de los requisitos de sistema de este proyecto. Estos requisitos derivan de los requisitos de usuario, definidos en Subsección 3.2.1, *Requisitos de usuario*, y proporcionan una información más concisa sobre el funcionamiento y las diferentes características del sistema.

Estos requisitos se clasifican en dos tipos:

- **Funcionales**: Representan las funcionalidades que el sistema debe proporcionar.
- **No funcionales**: Representan las restricciones y limitaciones impuestas por el sistema.

Cada requisito tiene un identificador único que sigue el siguiente formato: SR-YY-XX, donde YY es el tipo de requisito, *FN* para funcionales y *NF* para no funcionales, y XX es un número secuencial que identifica el requisito y comienza en 01.

Cada requisito de software se identifica con un ID que sigue el formato *RU-YY-XX*, donde *YY* identifica el tipo del requisito, ya sea funcional (*FN*) o no funcional (*NF*); y *XX* identifica el número secuencial del requisito dentro de su tipo, empezando en *01*.

La tabla 3.25 contiene la plantilla utilizada para la especificación de los requisitos, incluyendo la descripción de cada atributo.

TABLA 3.25
PLANTILLA DE REQUISITO DE SOFTWARE

RU-YY-XX	
Descripción	Especificación del requisito en un lenguaje claro, conciso y no ambiguo.
Necesidad	Prioridad del requisito para el usuario (*Esencial*, *Conveniente* u *Opcional*).
Prioridad	Prioridad del requisito para el desarrollador (*Alta*, *Media* o *Baja*).
Estabilidad	Indica si el requisito se modifica durante el desarrollo (*No cambia*, *No cambia* o *Muy inestable*).
Verificabilidad	Capacidad de comprobar la validez del requisito (*Alta*, *Media* o *Baja*).
Origen	Referencia a los requisitos de usuario que dieron lugar al requisito.

TABLA 3.26
REQUISITO RS-FN-01

RS-FN-01	
Descripción	El sistema debe de tener una GUI que facilite la interacción del usuario con el sistema y le permita visualizar la información sobre la ejecución y depuración del programa
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-01, RU-CA-11, RU-CA-12, RU-CA-13

TABLA 3.27
REQUISITO RS-FN-02

RS-FN-02	
Descripción	El sistema debe de tener un editor de código donde se escriba el código
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-02, RU-CA-03

TABLA 3.28
REQUISITO RS-FN-03

RS-FN-03	
Descripción	El editor debe de permitir al usuario poner breakpoints en el código, donde se parará la ejecución del programa al depurar
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-02, RU-CA-03, RU-CA-08

TABLA 3.29
REQUISITO RS-FN-04

RS-FN-04	
Descripción	El sistema debe de crear un fichero de código que almacene el código escrito por el usuario
Necesidad	Opcional
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-02, RU-CA-04

TABLA 3.30
REQUISITO RS-FN-05

RS-FN-05	
Descripción	El sistema debe de generar un archivo ejecutable a partir del código escrito a través del compilador
Necesidad	Opcional
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-04, RU-RE-01

TABLA 3.31
REQUISITO RS-FN-06

RS-FN-06	
Descripción	El sistema debe de ejecutar el archivo ejecutable generado
Necesidad	Opcional
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-05, RU-RE-02, RU-RE-04, RU-RE-05

TABLA 3.32
REQUISITO RS-FN-07

RS-FN-07	
Descripción	El sistema debe abrir una sesión de depuración sobre el archivo ejecutable generado
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	Cambiante
Verificabilidad	Baja
Origen	RU-CA-06, RU-RE-03, RU-RE-07, RU-RE-08

TABLA 3.33
REQUISITO RS-FN-08

RS-FN-08	
Descripción	El sistema debe de utilizar Helgrind o ThreadSanitizer para detectar errores de concurrencia
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-07

TABLA 3.34
REQUISITO RS-FN-09

RS-FN-09	
Descripción	El sistema debe permitir al usuario depurar paso a paso sin entrar en las funciones que se llaman
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-09

TABLA 3.35
REQUISITO RS-FN-10

RS-FN-10	
Descripción	El sistema debe permitir al usuario depurar paso a paso entrando en las funciones que se llaman
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-09

TABLA 3.36
REQUISITO RS-FN-11

RS-FN-11	
Descripción	El sistema debe permitir al usuario depurar paso a paso saliendo de las funciones que se llaman
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-09

TABLA 3.37
REQUISITO RS-FN-12

RS-FN-12	
Descripción	El sistema debe permitir moverse entre los breakpoints y continuar la ejecución del programa
Necesidad	Conveniente
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-08, RU-CA-09

TABLA 3.38
REQUISITO RS-FN-13

RS-FN-13	
Descripción	El sistema debe permitir al usuario rebobinar la ejecución del programa
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Media
Origen	RU-CA-10

TABLA 3.39
REQUISITO RS-FN-14

RS-FN-14	
Descripción	El sistema debe de mostrar durante la depuración los hilos que se encuentran en ejecución en el punto actual
Necesidad	Esencial
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-11

TABLA 3.40
REQUISITO RS-FN-15

RS-FN-15	
Descripción	El sistema debe de mostrar durante la depuración el estado de las variables de cada uno de los hilos
Necesidad	Esencial
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-12

TABLA 3.41
REQUISITO RS-FN-16

RS-FN-16	
Descripción	El sistema debe de mostrar durante la depuración la evolución de los hilos y procesos a través de un diagrama temporal
Necesidad	Conveniente
Prioridad	Baja
Estabilidad	No cambia
Verificabilidad	Baja
Origen	RU-CA-13

TABLA 3.42
REQUISITO RS-NF-01

RS-NF-01	
Descripción	El sistema debe de tener un compilador de código C
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	Cambiante
Verificabilidad	Alta
Origen	RU-RE-01, RU-RE-06

TABLA 3.43
REQUISITO RS-NF-02

RS-NF-02	
Descripción	El sistema debe de ejecutar el código C compilado
Necesidad	Opcional
Prioridad	Media
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-RE-02

TABLA 3.44
REQUISITO RS-NF-03

RS-NF-03	
Descripción	El sistema debe de ejecutar en un sistema operativo UNIX para soportar la ejecución de hilos POSIX
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-05, RU-RE-02

TABLA 3.45
REQUISITO RS-NF-04

RS-NF-04	
Descripción	El sistema debe de depurar código C
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Baja
Origen	RU-RE-03

TABLA 3.46
REQUISITO RS-NF-05

RS-NF-05	
Descripción	La herramienta debe de tener el mismo comportamiento independientemente del sistema operativo en el que se ejecute
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Media
Origen	RU-RE-04

TABLA 3.47
REQUISITO RS-NF-06

RS-NF-06	
Descripción	La herramienta debe de tener el mismo comportamiento independientemente de la arquitectura que tenga el sistema en el que se ejecute
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Baja
Origen	RU-RE-05

TABLA 3.48
REQUISITO RS-NF-07

RS-NF-07	
Descripción	La depuración no deberá mostrar información sobre direcciones de memoria ni de código que no pertenezca al usuario
Necesidad	Esencial
Prioridad	Media
Estabilidad	Cambiante
Verificabilidad	Baja
Origen	RU-RE-07, RU-RE-08, RU-CA-11, RU-CA-12

TABLA 3.49
REQUISITO RS-NF-08

RS-NF-08	
Descripción	El sistema debe ser estable y no presentar errores al depurar
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-01, RU-CA-06, RU-RE-09

TABLA 3.50
REQUISITO RS-NF-09

RS-NF-09	
Descripción	El sistema debe de ejecutar el código en un entorno aislado para evitar problemas de seguridad
Necesidad	Opcional
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Media
Origen	RU-RE-09

TABLA 3.51
REQUISITO RS-NF-10

RS-NF-10	
Descripción	El sistema debe de ser multiplataforma
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-RE-10

3.3. Casos de uso

En esta sección se describirán los casos de uso del cliente a través de un modelo de casos de uso UML [30]. Los casos de uso son una técnica que se utiliza para capturar los requisitos funcionales de un sistema y se representan como un conjunto de acciones que un sistema realiza en colaboración con uno o más actores. Los actores son entidades que interactúan con el sistema y pueden ser tanto humanos como sistemas externos [31].

Cada caso de uso está identificado por un identificador único. Este identificador sigue el siguiente formato: CU-XX, donde XX es un número secuencial que identifica el caso de uso y comienza en 01.

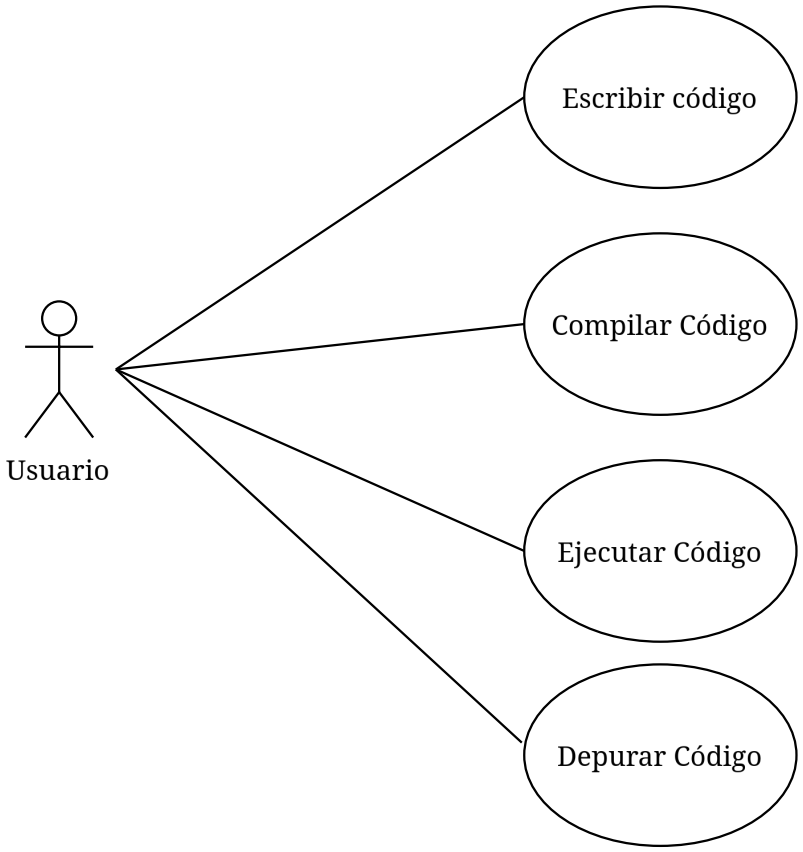

Fig. 3.1. Diagrama de casos de uso del sistema.

Para la especificación de los casos de uso se usará la siguiente plantilla:

Cada caso de uso se identifica con un ID que sigue el formato *CU-XX*, donde *XX* identifica el número secuencial del caso de uso, empezando en *01*.

La tabla 3.52 contiene la plantilla utilizada para la especificación de los casos de uso, incluyendo la descripción de cada atributo.

TABLA 3.52
PLANTILLA DE CASO DE USO

CU-XX	
Nombre	Descripción breve del caso de uso.
Actores	Agente externo que ejecuta el caso de uso.
Objetivo	Propósito del caso de uso.
Descripción	Pasos que debe seguir el actor para ejecutar el caso de uso.
Pre-condición	Condiciones previas que se deben cumplir para ejecutar el caso de uso.
Post-condición	Condiciones que se deben cumplir después de ejecutar el caso de uso.

TABLA 3.53
CASO DE USO CU-01

CU-01	
Nombre	Escribir código
Actores	Usuario
Objetivo	Desarrollar código C concurrente
Descripción	1. Clica en el editor 2. Escribe código C
Pre-condición	N/A
Post-condición	N/A

TABLA 3.54
CASO DE USO CU-02

CU-02	
Nombre	Compilar código
Actores	Usuario
Objetivo	Compilar el código C
Descripción	1. Clica en el botón de compilar 2. Se compila el código 3. En la parte inferior se muestra el resultado de la compilación
Pre-condición	Tener código C escrito dentro del editor de código
Post-condición	Se genera un archivo ejecutable en el backend

TABLA 3.55
CASO DE USO CU-03

CU-03	
Nombre	Ejecutar código
Actores	Usuario
Objetivo	Ejecutar el código C
Descripción	1. Clica en el botón de ejecutar 2. Se ejecuta el código 3. En la parte inferior se muestra el resultado de la ejecución
Pre-condición	Tener código C compilado
Post-condición	Se ejecuta en el backend el archivo ejecutable

TABLA 3.56
CASO DE USO CU-04

CU-04	
Nombre	Depurar código
Actores	Usuario
Objetivo	Depurar el código C
Descripción	1. Desde la sección de edición se selecciona la opción de depuración 2. El usuario elige de qué forma quiere depurar el código 3. Se ejecuta el código paso a paso 4. Se muestra el estado de los hilos y procesos
Pre-condición	Tener código C compilado
Post-condición	Se inicia una sesión de depuración respecto al fichero ejecutable en el backend

3.4. Trazabilidad

En esta sección se detallará la Trazabilidad de los requisitos de usuario y sistema, donde los requisitos funcionales deben de cubrir todos los requisitos de capacidad (Tabla 3.57) y los requisitos no funcionales deben de cubrir todos los requisitos de restricción (Tabla 3.58), y con los casos de uso (Tabla 3.59). La Trazabilidad es la capacidad de seguir y documentar la vida de un requisito, desde su origen hasta su implementación y pruebas. La Trazabilidad es una parte importante de la gestión de requisitos, ya que permite a los desarrolladores y a los interesados comprender la historia y el propósito de un requisito, así como evaluar el impacto de los cambios en los requisitos [28].

TABLA 3.57

TRAZABILIDAD DE LOS REQUISITOS FUNCIONALES CON LOS REQUISITOS DE CAPACIDAD

	RU-CA-01	RU-CA-02	RU-CA-03	RU-CA-04	RU-CA-05	RU-CA-06	RU-CA-07	RU-CA-08	RU-CA-09	RU-CA-10	RU-CA-11	RU-CA-12	RU-CA-13
RS-FN-01	•									•	•	•	
RS-FN-02		•	•										
RS-FN-03		•	•					•					
RS-FN-04		•		•									
RS-FN-05				•									
RS-FN-06					•								
RS-FN-07						•							
RS-FN-08							•						
RS-FN-09									•				
RS-FN-10									•				
RS-FN-11									•				
RS-FN-12								•	•				
RS-FN-13										•			
RS-FN-14											•		
RS-FN-15												•	
RS-FN-16													•

TABLA 3.58

TRAZABILIDAD DE LOS REQUISITOS NO FUNCIONALES CON LOS REQUISITOS DE RESTRICCIÓN

	RU-RE-01	RU-RE-02	RU-RE-03	RU-RE-04	RU-RE-05	RU-RE-06	RU-RE-07	RU-RE-08	RU-RE-09	RU-RE-10
RS-NF-01	•					•				
RS-NF-02		•								
RS-NF-03		•								
RS-NF-04			•							
RS-NF-05				•						
RS-NF-06					•					
RS-NF-07							•	•		
RS-NF-08									•	
RS-NF-09									•	
RS-NF-10										•

TABLA 3.59
TRAZABILIDAD DE REQUISITOS DE USUARIO CON CASOS DE USO

	CU-01	CU-02	CU-03	CU-04
RU-CA-01	•			
RU-CA-02	•			
RU-CA-03	•			
RU-CA-04		•		
RU-CA-05			•	
RU-CA-06				•
RU-CA-07		•		•
RU-CA-08			•	•
RU-CA-09				•
RU-CA-10				•
RU-CA-11				•
RU-CA-12				•
RU-CA-13				•
RU-RE-01		•		
RU-RE-02			•	
RU-RE-03				•
RU-RE-04		•	•	•
RU-RE-05		•	•	•
RU-RE-06		•		
RU-RE-07				•
RU-RE-08				•
RU-RE-09			•	
RU-RE-10	•	•	•	•

CAPÍTULO 4

DISEÑO

En este capítulo se detalla el diseño que tendrá la solución propuesta. Para ello se discutirán sobre las posibles alternativas y cuál de ellas ha sido elegida (Sección 4.1, *Construcción de la solución final*) y posteriormente, se describirá la arquitectura del sistema (Sección 4.3, *Arquitectura del sistema*) en base a la solución final elegida.

4.1. Construcción de la solución final

Para la realización del estudio de la solución final se han tenido en cuenta los requisitos especificados en Sección 3.2, *Requisitos* y las herramientas existentes descritas en Capítulo 2, *Estado del arte*.

4.1.1. Monolítico vs Distribuido

Durante la especificación de los requisitos se han descrito varios requisitos que influyen en la elección del sistema operativo y de la arquitectura del sistema. En concreto los requisitos Requisito RS-NF-05 y Requisito RS-NF-06 especifican que la herramienta debe de tener el mismo funcionamiento aunque la arquitectura del sistema o el sistema operativo sean diferentes. Estos requisitos nos obligan a tomar una decisión respecto a la arquitectura del sistema.

Monolítico

Una aplicación monolítica [32] es aquella que se ejecuta en una única máquina y en un único sistema operativo. Este tipo de aplicaciones son más sencillas de desarrollar y de mantener, ya que no requieren de una comunicación entre diferentes máquinas. Sin embargo, una aplicación monolítica no es capaz de ejecutarse en diferentes sistemas ope-

rativos, ya que el código fuente debe de ser compilado para cada sistema operativo.

Distribuido

Una aplicación distribuida [32] es aquella que se ejecuta en múltiples procesos. Este tipo de aplicaciones tienen una mayor complejidad de desarrollar, debido a la comunicación entre procesos. Sin embargo, una aplicación distribuida que sirva un servicio web podría funcionar en cualquier sistema operativo, dado que solo ejecutaría el código del cliente,

4.1.2. Elección de la arquitectura

Tras exponer las ventajas y desventajas de cada una de las arquitecturas, se ha decidido optar por una arquitectura distribuida. La arquitectura monolítica no es una opción viable, ya que para cumplir con los requisitos Requisito RS-NF-05, Requisito RS-NF-06 y Requisito RS-NF-10 sería necesario compilar el código fuente para cada sistema operativo y arquitectura. Esto supondría un gran esfuerzo de desarrollo y mantenimiento, ya que habría que mantener diferentes versiones del código fuente para cada sistema operativo y arquitectura.

En cambio, una arquitectura distribuida permitiría tener un único código fuente que se ejecute en cualquier sistema operativo y arquitectura, siempre y cuando el cliente sea una aplicación web. Esto facilitaría el desarrollo y mantenimiento del código fuente, ya que solo habría que mantener una única versión del código fuente. Además, la arquitectura distribuida permitiría escalar el sistema de manera más sencilla, ya que se podrían añadir más servidores para soportar más carga de trabajo. Además, la arquitectura distribuida permitiría ejecutar el código fuente en un sistema operativo Unix que soporte la ejecución de hilos POSIX, lo que facilitaría la implementación de la funcionalidad de depuración.

4.1.3. Almacenamiento de estados vs Depuradores Externos

Almacenamiento de estados

Para implementar las operaciones básicas de un depurador es necesario moverse por la ejecución del código fuente. Es por esta razón que se evaluó la posibilidad de inyección de código en el programa cuyo propósito sea almacenar el estado de la ejecución en un momento determinado. Con estos estados almacenados el usuario podría moverse entre cada uno de ellos, facilitando el rebobinado de la ejecución.

Para almacenar estos estados sería necesario introducir código en el programa tras cada sentencia que modifique el estado de la ejecución, es decir, creación de variables, modificación de variables o llamadas al sistema que generasen nuevos hilos. Lo que supone crear un *parser* para encontrar estas sentencias y añadir a continuación una función que almacene la variable y su valor en la función e hilo correspondiente.

Uso de depuradores externos

También se ha evaluado la opción de utilizar un depurador externo. Durante el Capítulo 2, *Estado del arte* se analizaron los depuradores existentes, dentro de estos depuradores tan solo GDB (Subsección 2.2.1, *GDB*), LLDB (Subsección 2.2.2, *LLDB*) y RR (Subsección 2.2.3, *RR*) permitían construir una aplicación sobre ellos. Aunque LLDB tenga una API en Python que permita la interactividad con la sesión de depuración, carece de la capacidad de rebobinar durante la depuración, tal y como indica el requisito Requisito RS-FN-13. Por lo tanto, RR es una opción más viable, que aunque no ofrezca una API en Python que permita la interactividad, sí que ofrece una interfaz máquina que ofrece una salida formateada.

No obstante, RR registra la ejecución del programa para reproducirla posteriormente, garantizando así una ejecución determinista. Esto limitaría la capacidad del usuario para modificar el flujo de ejecución a su conveniencia. Por ello, se ofrecerá la opción de elegir entre grabar la ejecución, permitiendo el rebobinado, o no grabarla (utilizando solo GDB), permitiendo una ejecución libre de los hilos.

Elección de depurador

Analizadas ambas propuestas, se ha decidido optar por la segunda opción, es decir, utilizar un depurador externo. Esta decisión se ha tomado debido a la complejidad que supone implementar un *parser* que permita reconocer dónde se debe inyectar código para almacenar el estado de la ejecución. Además, el uso de un depurador externo como RR permite aprovechar su funcionalidad de rebobinado, lo que facilita la implementación de las funcionalidades del depurador.

4.1.4. Sandbox

Entorno no sandbox

Si se optara por no usar un entorno aislado, el código del cliente se ejecutaría en la propia máquina del servidor. Esto podría suponer un problema de seguridad, ya que el código del cliente, que al ser código C puede acceder fácilmente a la memoria, ficheros o recursos, lo que añadido a una intención maliciosa podría provocar un ataque al servidor. Además, el uso de recursos del servidor podría verse comprometido, ya que el código del cliente podría consumir todos los recursos del servidor, afectando a la disponibilidad del mismo.

Entorno sandbox

Por otro lado, si se opta por un entorno aislado, el código del cliente se ejecutará en un entorno controlado y limitado. Esto evitaría que el código del cliente pudiera acceder a la memoria o a los ficheros del servidor, lo que aumentaría la seguridad del sistema.

4.1.5. Elección de entorno sandbox

Dadas las ventajas que supone el uso de un entorno aislado y la necesidad de cumplir con el requisito Requisito RS-NF-09, "*el sistema debe de ejecutar el código en un entorno aislado para evitar problemas de seguridad o disponibilidad*". Para ello, se optó por utilizar contenedores Docker en los que se ejecutarán los programas. De esta manera cualquier código malicioso no podrá acceder a los archivos del sistema ni atacar a la disponibilidad del servidor. Además se limitarán los recursos de los contenedores para controlar el uso de los mismos.

La principal dificultad que supone incorporar contenedores es controlar la disponibilidad de cada uno de ellos. Las características del proyecto nos obligan a que la comunicación entre cliente y docker sea por sesión, lo que implica que durante la sesión el docker no estará disponible para servir a ningún otro cliente. Una posible solución es que la responsabilidad del servidor sea servir los ficheros estáticos del cliente y redirigir los mensajes del cliente al contenedor asociado a la sesión correspondiente y gestionar las comunicaciones entre clientes y contenedores. Esta solución podría generar un cuello de botella en el componente servidor en el caso de tener muchos clientes activos, dado que todas las comunicaciones pasarían por el servidor.

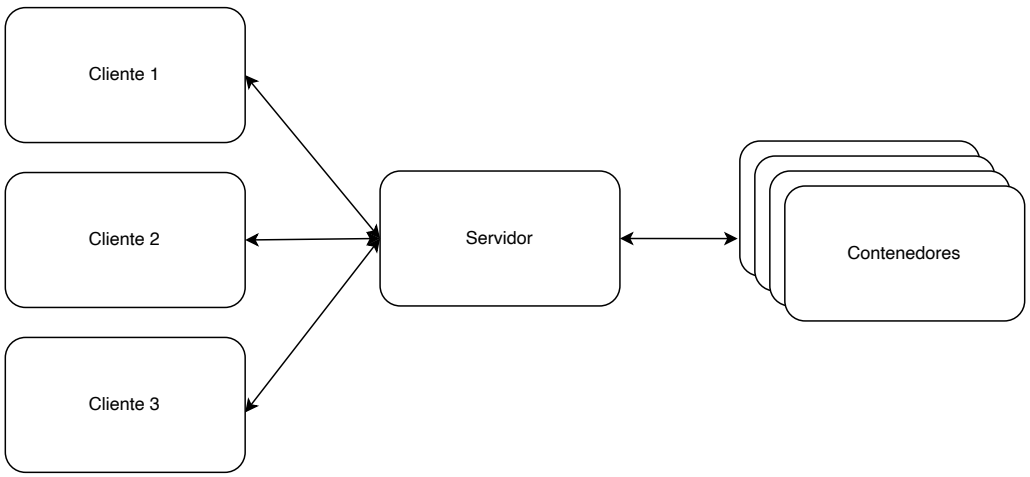

Fig. 4.1. Primera propuesta de arquitectura del sistema

Es por esta razón que se ha tomado la decisión de quitar responsabilidad al servidor e incorporarla en los contenedores. El servidor se convertirá en un *proxy* y se encargará de redirigir la conexión de un cliente a un docker disponible, es decir, solo tendrá información sobre a qué contenedor está conectado cada cliente. Cada docker alojará un pequeño servidor que sirva los ficheros estáticos y las funcionalidades de los requisitos.

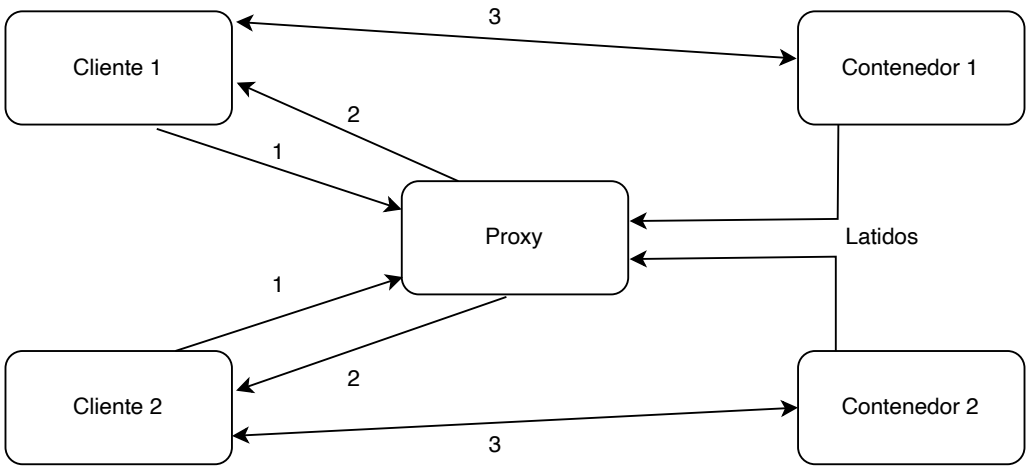

Fig. 4.2. Propuesta final de arquitectura del sistema

Es importante destacar, que como un contenedor puede dejar de estar disponible por un uso malicioso desde el lado cliente, el proxy debe de conocer la disponibilidad de cada uno de los contenedores. Para ello cada uno de los contenedores mandará latidos, de manera que el proxy pueda conocer qué dockers están disponibles.

Para conseguir todo esto se creará una imagen Docker que contenga el código correspondiente a la comunicación tanto con el cliente como con el proxy, el código de las funcionalidades del depurador y los ficheros estáticos a servir. Cada uno de estos contenedores se levantará a través de *Docker Compose*, configurado con un `docker-compose.yml` en el que se indicarán los puertos que cada docker tiene que mapear y las variables de entorno para las comunicaciones.

Es por esto que la arquitectura distribuida final constará de uno o varios clientes, un único *proxy*, al que se conectarán los clientes para ser redirigidos, y varios *contenedores* que se encargarán de toda la lógica del sistema.

4.1.6. Comunicación entre procesos

Dada la existencia de un cliente, un proxy y varios contenedores, es necesario establecer una comunicación entre ellos. Esta comunicación se puede realizar de diferentes maneras, pero se han considerado las dos siguientes opciones:

- **REST**: Permite la comunicación entre procesos a través de peticiones Hypertext Transfer Protocol (HTTP). Esta comunicación es unidireccional, sin control de estados y se establece una conexión por cada petición. Es ideal para controlar una gran cantidad de comunicaciones y para servir unos recursos.
- **Web Sockets**: Esta comunicación es bidireccional, y permite el envío de mensajes en tiempo real. Es ideal para aplicaciones que requieren una comunicación constante.

Dadas las características de la aplicación, se optará por usar REST API para servir los ficheros estáticos y para enviar los latidos de los contenedores, se usará Web Sockets para la comunicación entre el cliente y su correspondiente contenedor, permitiendo entablar la sesión y que exista una comunicación bidireccional.

4.1.7. Lenguajes de programación

Dada la arquitectura distribuida del sistema, se necesita usar un lenguaje de programación para el lado cliente y otro para el lado servidor, correspondiente al proxy y los contenedores.

Tecnologías del lado cliente

Para la construcción de la aplicación web se ha basará en HTML5 para dar estructura al código, CSS3 para dar estilos, y JavaScript para implementar las distintas funcionalidades. Se ha decidido elegir este *Stack* por ser el más básico y el que más experiencia se tiene. Este *Stack* será más que suficiente dado que el lado cliente no será de gran envergadura y no necesitará más que unos pocos botones, un editor de código y una comunicación a través de WebSockets.

Tecnologías del lado servidor

En cuanto al lenguaje de programación del lado servidor es necesario elegir un lenguaje que permita la implementación de WebSockets para la comunicación en tiempo real, una API REST para servir los ficheros estáticos y enviar los latidos, que trabaje con ficheros para la creación de ficheros con el código del usuario, que permita la concurrencia para poder gestionar procesos en paralelo, y que permita crear un proceso en el que ejecutar RR. Dentro de las opciones destacan Python, Go, Rust y NodeJS, por lo que vamos a analizar cada una de ellas.

- **Python**: Un lenguaje de programación interpretado, muy versátil, aunque con un rendimiento moderado. Gracias a frameworks como Flask para implementar APIs REST, aiohttp o FastAPI para WebSockets, y al módulo threading, para la concurrencia, es una opción viable.

- **Go**: Un lenguaje de programación compilado, con un buen rendimiento y orientado en la concurrencia y el procesamiento en la red. Su soporte nativo para WebSockets mediante goroutines, y frameworks como Gin o Echo para APIs REST lo hacen ideal para este tipo de comunicación.

- **Rust**: Un lenguaje de programación compilado, con un gran rendimiento, aunque con una curva de aprendizaje pronunciada. Tiene bibliotecas como Actix-web para APIs REST y WebSockets, pero no es la más recomendable por su complejidad.

- **NodeJS**: Un lenguaje de programación interpretado con excelente soporte tanto para APIs REST (Express.js) como para WebSockets (Socket.io), aunque su concurrencia se basa en un solo hilo utilizando un bucle de eventos y un modelo asíncrono; lo que no es un paralelismo real. Su modelo no es ideal para operaciones intensivas en CPU.

Tanto Go como Python son lenguajes adecuados para el proyecto. Aunque Go ofrece un rendimiento superior, la disponibilidad de la biblioteca Python *pygdbmi*, que facilita la gestión de una sesión RR/GDB y la obtención de la salida MI en formato JSON, convierte a Python en la opción más conveniente.

4.2. Actualizacion de los requisitos del sistema

Tras las decisiones tomadas durante la construcción de la solución final, se han actualizado los requisitos del sistema. Las decisiones tomadas han llevado a generar 3 nuevos requisitos funcionales de software y un nuevo requisito no funcional, que se describen a continuación:

TABLA 4.1
REQUISITO RS-FN-17

RS-FN-17	
Descripción	El sistema debe implementarse como una aplicación distribuida, donde los componentes puedan ejecutarse en diferentes ubicaciones físicas, permitiendo que el código del usuario se ejecute en un entorno controlado separado del cliente.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-01

Tras la actualización de los requisitos, se han actualizado los requisitos de trazabilidad, esto se muestra en las siguientes tablas Tabla 4.5 Tabla 4.6.

TABLA 4.2
REQUISITO RS-FN-18

RS-FN-18	
Descripción	El sistema debe implementarse como una aplicación web accesible desde un navegador, permitiendo su uso sin instalación específica en la máquina del usuario y garantizando la compatibilidad multiplataforma.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-CA-01

TABLA 4.3
REQUISITO RS-FN-19

RS-FN-19	
Descripción	El sistema debe utilizar WebSockets para la comunicación en tiempo real entre el cliente y el servidor, permitiendo la actualización inmediata de la información de depuración sin necesidad de recargar la página.
Necesidad	Conveniente
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Media
Origen	RU-CA-11, RU-CA-12, RU-CA-13

TABLA 4.4
REQUISITO RS-NF-11

RS-NF-11	
Descripción	El sistema debe ejecutarse en un entorno aislado utilizando contenedores Docker, garantizando la seguridad y el control de recursos durante la ejecución del código del usuario.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	No cambia
Verificabilidad	Alta
Origen	RU-RE-09

TABLA 4.5

TRAZABILIDAD DE LOS REQUISITOS FUNCIONALES CON LOS REQUISITOS DE CAPACIDAD

	RU-CA-01	RU-CA-02	RU-CA-03	RU-CA-04	RU-CA-05	RU-CA-06	RU-CA-07	RU-CA-08	RU-CA-09	RU-CA-10	RU-CA-11	RU-CA-12	RU-CA-13
RS-FN-01	•									•	•	•	
RS-FN-02		•	•										
RS-FN-03		•	•					•					
RS-FN-04		•		•									
RS-FN-05				•									
RS-FN-06					•								
RS-FN-07						•							
RS-FN-08							•						
RS-FN-09									•				
RS-FN-10									•				
RS-FN-11									•				
RS-FN-12								•	•				
RS-FN-13										•			
RS-FN-14											•		
RS-FN-15												•	
RS-FN-16													•
RS-FN-17	•												
RS-FN-18	•												
RS-FN-19										•	•	•	

TABLA 4.6

TRAZABILIDAD DE LOS REQUISITOS NO FUNCIONALES CON LOS REQUISITOS DE RESTRICCIÓN

	RU-RE-01	RU-RE-02	RU-RE-03	RU-RE-04	RU-RE-05	RU-RE-06	RU-RE-07	RU-RE-08	RU-RE-09	RU-RE-10
RS-NF-01	•					•				
RS-NF-02		•								
RS-NF-03		•								
RS-NF-04			•							
RS-NF-05				•						
RS-NF-06					•					
RS-NF-07							•	•		
RS-NF-08									•	
RS-NF-09									•	
RS-NF-10										•
RS-NF-11									•	

4.3. Arquitectura del sistema

4.3.1. Componentes del sistema

Tal y como se ha comentado en Sección 4.1, *Construcción de la solución final*, el sistema consta de tres partes: el cliente, el *proxy* y el contenedor docker.

En la figura 4.3 se muestra el diagrama de componentes UML [30] donde se pueden observar los distintos componentes y sus diferentes interfaces.

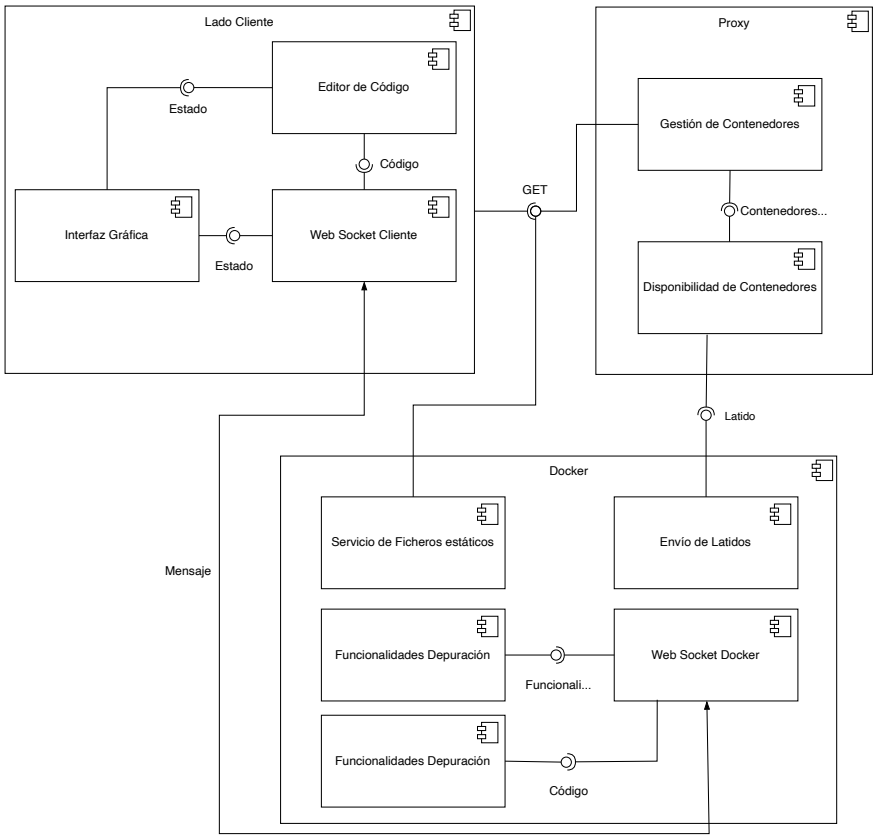

Fig. 4.3. Diagrama de componentes del sistema

A continuación se describen los distintos componentes del sistema y sus responsabilidades.

- **Cliente**: Permite facilitar la interacción del usuario con el depurador, además de visualizar las salidas y cambios ocurridos durante la ejecución. En este componente se podrá escribir código, y comunicará las acciones al servidor para ejecutar las correspondientes funcionalidades. Dentro de la aplicación web, existen dos componentes principales:

  - **Interfaz Gráfica**: Su responsabilidad es actualizar la información que se muestra por pantalla.

    - **Comunicación con el contenedor (Web Socket)**: Se encarga de enviar y recibir mensajes del contenedor.
    - **Editor de Código**: Su responsabilidad es permitir al usuario escribir el código que se va a depurar.

- **Proxy**: Se encargará de gestionar la disponibilidad de los contenedores y los contenedores libres. Además redirigirá los clientes a los contenedores correspondientes. Este componente consta de dos componentes:
    - **Gestión de contenedores**: Al recibir una petición de un cliente se le redirigirá a un contenedor libre, este contenedor será asociado a este cliente hasta que se desconecte, por lo que dejará de estar libre.
    - **Disponibilidad de contenedores**: Alojará un servicio para escuchar los contenedores disponibles. Si tras cierto tiempo deja de recibir latidos de un contenedor lo dará por muerto, por lo que no podrá ser asociado a nuevos clientes.
- **Contenedor Docker**: Se encarga de servir los ficheros estáticos al cliente y de ejecutar las funcionalidades pedidas por este.
    - **Servicio de ficheros estáticos**: El contenedor servirá el html, css y JavaScript del cliente al recibir una petición GET por parte del cliente.
    - **Web Socket Docker**: El contenedor utilizará Web Socket para la comunicación con el cliente.
    - **Funcionalidades de Compilación**: Este componente se encargará de compilar el código recibido y ejecutarlo sin modo depuración.
    - **Funcionalidades de Depuración**: Este componente contendrá la implementación de las funcionalidades de depuración, las cuales serán llamadas a través de la API.
    - **Envío de latidos**: El contenedor enviará latidos cada cierto tiempo.

Para la especificación de los distintos componentes se ha usado la siguiente plantilla:

La tabla 4.7 contiene la plantilla utilizada para la especificación de los componentes, incluyendo la descripción de cada atributo.

TABLA 4.7
PLANTILLA DE COMPONENTE

Nombre	
Rol	Papel del componente en el sistema.
Dependencias	Componentes que dependen de este.
Descripción	Explicación del funcionamiento del componente.
Datos	Datos de entrada (*in*) y salida (*out*) del componente.
Origen	Requisitos software que dieron lugar al componente.

TABLA 4.8
COMPONENTE 'INTERFAZ GRÁFICA'

Interfaz Gráfica	
Rol	Mostrar la información actualizada al usuario por pantalla
Dependencias	COM-Editor de Código, COM-Web Socket Cliente
Descripción	La Interfaz Gráfica debe de gestionar la interacción del usuario con la aplicación y mostrar la información de manera estructurada y clara.
Datos	▪ **in:** Estado actual del sistema ▪ **out:** NA
Origen	RS-FN-01, RS-FN-08, RS-FN-15, RS-FN-14, RS-FN-16, RS-FN-18

TABLA 4.9
COMPONENTE 'EDITOR DE CÓDIGO'

Editor de Código	
Rol	Permitir al usuario escribir el código que se va a depurar
Dependencias	N/A
Descripción	Este componente permitirá al usuario escribir código, colorear la sintaxis y gestionar de forma visual los breakpoints.
Datos	▪ **in:** N/A ▪ **out:** Código escrito, estado del editor de código
Origen	RS-FN-01, RS-FN-02, RS-FN-03

TABLA 4.10
COMPONENTE 'WEB SOCKET CLIENTE'

Web Socket Cliente	
Rol	Punto de comunicación por Web Socket desde el cliente al servidor
Dependencias	COM-Editor de Código, COM-Web Socket y REST API Servidor
Descripción	La comunicación con el servidor se encargará de gestionar la comunicación entre el cliente y el servidor a través de Web Sockets.
Datos	▪ **in:** Mensajes del servidor ▪ **out:** Cambios del sistema ,Mensajes del cliente
Origen	RS-FN-19, RS-FN-17

TABLA 4.11
COMPONENTE 'GESTIÓN DE CONTENEDORES'

Gestión de contenedores	
Rol	Asignar clientes a contenedor libres
Dependencias	COM-Disponibilidad de contenedores
Descripción	Este componente asignará a un nuevo cliente un contenedor libre hasta que se desconecte. Al asignar a un contenedor le redireccionará al contenedor correspondiente
Datos	▪ **in:** Contenedores Disponibles ▪ **out:** URL para la conexión
Origen	RS-FN-19, RS-FN-17

TABLA 4.12
COMPONENTE 'DISPONIBILIDAD DE CONTENEDORES'

Disponibilidad de contenedores	
Rol	Escucha latidos de los contenedores para comprobar su disponibilidad
Dependencias	COM-Gestión de contenedores, COM-Envío de Latidos
Descripción	Registra cada uno de los latidos escuchados por cada contenedor, si tras 10 segundos no ha recibido un latido dará por no disponible al contenedor.
Datos	▪ **in:** Latido ▪ **out:** Contenedores Disponibles
Origen	RS-FN-17, RS-NF-11

TABLA 4.13
COMPONENTE 'WEB SOCKET DOCKER'

Web Socket Docker	
Rol	Gestión de comunicación entre los contenedores y el cliente a través de Web Sockets
Dependencias	COM-Web Socket Cliente, COM-Funcionalidades Depuración
Descripción	La comunicación con los contenedores se encargará de gestionar la comunicación entre el contenedor y el cliente a través de Web Sockets.
Datos	▪ **in:** Mensajes del cliente ▪ **out:** Mensajes del contenedor
Origen	RS-FN-19, RS-FN-17

TABLA 4.14
COMPONENTE 'FUNCIONALIDADES COMPILACIÓN'

Funcionalidades Compilación	
Rol	Se encarga de compilar el código recibido y ejecutarlo sin modo depuración
Dependencias	COM-Web Socket Docker
Descripción	Este componente se encarga de realizar acciones básicas que no están relacionadas directamente con la depuración, como compilar el código y ejecutarlo.
Datos	▪ **in:** Código ▪ **out:** Salida Estándar
Origen	RS-FN-04, RS-FN-05, RS-FN-06, RS-FN-08, RS-NF-11

TABLA 4.15
COMPONENTE 'FUNCIONALIDADES DE DEPURACIÓN'

Funcionalidades de Depuración	
Rol	Gestionar la ejecución de las funcionalidades de depuración
Dependencias	COM-Web Socket Docker
Descripción	Este componente ejecutará las funcionalidades del depurador y devolverá el resultado de aplicar esas funcionalidades.
Datos	▪ **in:** Funcionalidad a realizar ▪ **out:** Estado tras la ejecución de la funcionalidad
Origen	RS-FN-07, RS-FN-09, RS-FN-10, RS-FN-11, RS-FN-12, RS-FN-13

TABLA 4.16
COMPONENTE 'ENVÍO DE LATIDOS'

Envío de Latidos	
Rol	Envío de latidos
Dependencias	COM-Disponibilidad de contenedores
Descripción	Envío de latidos cada 5 segundos a la dirección del *proxy*
Datos	▪ **in:** N/A ▪ **out:** Latido
Origen	RS-FN-17, RS-NF-11

TABLA 4.17
COMPONENTE 'SERVICIO DE FICHEROS ESTÁTICOS'

Servicio de Ficheros Estáticos	
Rol	Envío de ficheros estáticos al cliente
Dependencias	N/A
Descripción	Se encarga de enviar ficheros estáticos tras recibir una petición GET por parte del cliente para que este pueda acceder a su aplicación web.
Datos	▪ **in:** Petición GET del cliente ▪ **out:** Ficheros HTML, CSS y JavaScript
Origen	RS-FN-18, RS-NF-11

Trazabilidad

TABLA 4.18

TRAZABILIDAD DE LOS COMPONENTES CON LOS REQUISITOS FUNCIONALES

	RS-FN-01	RS-FN-02	RS-FN-03	RS-FN-04	RS-FN-05	RS-FN-06	RS-FN-07	RS-FN-08	RS-FN-09	RS-FN-10	RS-FN-11	RS-FN-12	RS-FN-13	RS-FN-14	RS-FN-15	RS-FN-16	RS-FN-17	RS-FN-18	RS-FN-19
Disponibilidad de contenedores																•			
Editor de Código	•	•	•																
Envío de Latidos																•			
Funcionalidades Compilación				•	•	•		•											
Funcionalidades de Depuración							•		•	•	•	•	•						
Gestión de contenedores																•			•
Interfaz Gráfica	•							•						•	•	•		•	
Servicio de Ficheros Estáticos																		•	
Web Socket Cliente																•			•
Web Socket Docker																•			•

4.3.2. Interfaz gráfica del sistema

La interfaz gráfica del sistema se ha diseñado para facilitar la interacción del usuario con el depurador. La figura Figura 4.4 muestra un mockup de la interfaz gráfica.

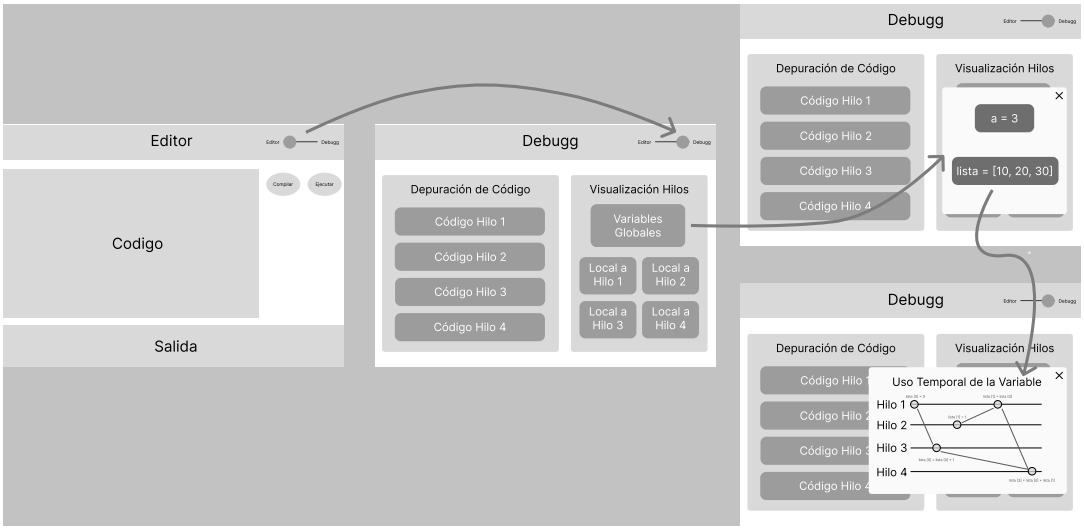

Fig. 4.4. Mockup de la interfaz gráfica del sistema

La interfaz gráfica se divide en dos secciones:

- **Editor de Código**: Permite al usuario escribir el código que se va a depurar. Este editor cuenta con resaltado de sintaxis y permite establecer puntos de interrupción (breakpoints) en el código. Justo a la derecha se encuentran dos botones, para compilar (obligatorio para acceder al resto de funcionalidades) y ejecutar. En la parte inferior del editor se muestra la salida estándar del código, lo que permite al usuario ver los resultados de la compilación y ejecución del código.

- **Panel de Depuración**: En esta sección se mostrará la información de los hilos y las variables del programa. En la parte superior se tendrán botones para realizar las diferentes acciones de depuración. En la parte izquierda se mostrán los hilos activos, y en la sección de código en la que se han encontrado al pararse. En la parte derecha se mostrán las variables de cada hilo, junto con su valor actual. Si se clicka en una variable global se mostrará la evolución de su valor a lo largo de la ejecución del programa.

4.3.3. Diagrama de secuencia del sistema

Con el objetivo de visualizar la interacción entre los distintos componentes del sistema, se han desarrollado varios diagramas de secuencia que muestran la interacción entre los distintos componentes del sistema.

Diagrama de secuencia de la conexión del cliente

En la figura 4.5 se muestra cómo el cliente realiza una petición `GET` al *proxy*, el cual le redirige a un contenedor libre, devolviendo este los ficheros estáticos al cliente.

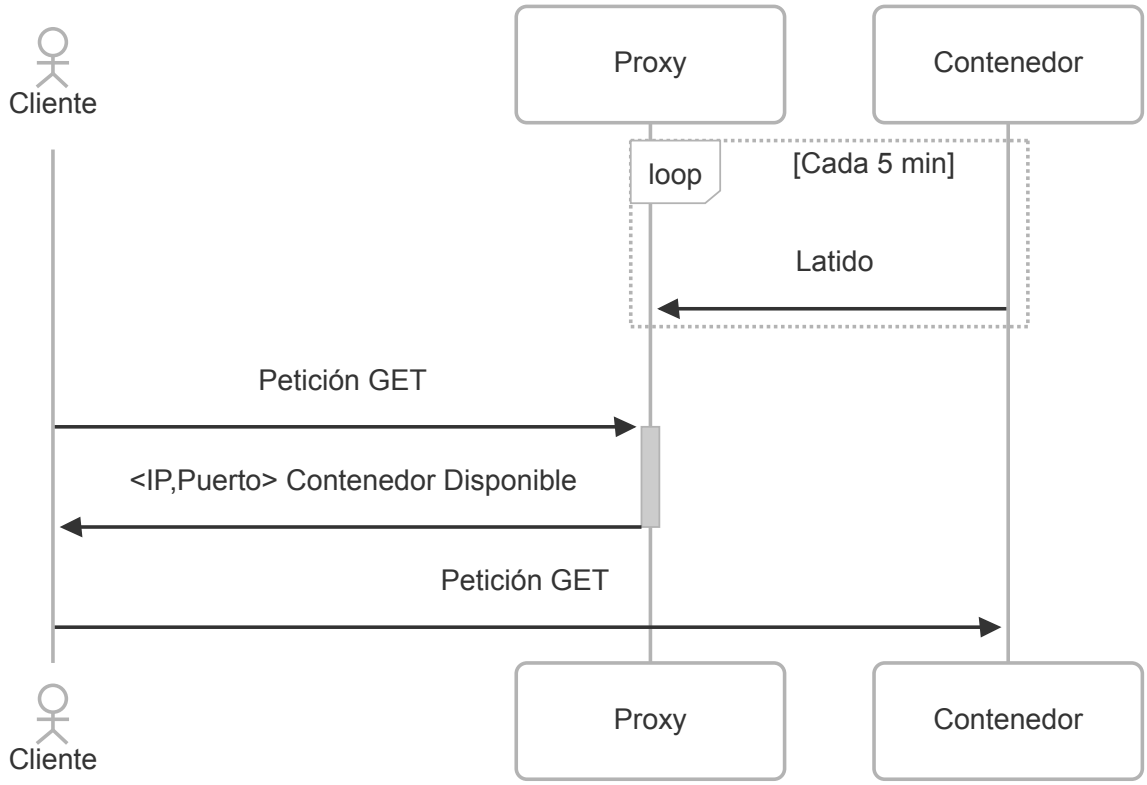

Fig. 4.5. Diagrama de secuencia de la conexión del cliente al contenedor

Diagrama de secuencia de la gestion de contenedores

En la figura 4.6 se muestra cómo los contenedores envían latidos al *proxy* con cierta frecuencia, y cómo el *proxy* gestiona la disponibilidad de los contenedores. Si un contenedor deja de enviar latidos durante un tiempo determinado, el *proxy* lo marca como no disponible.

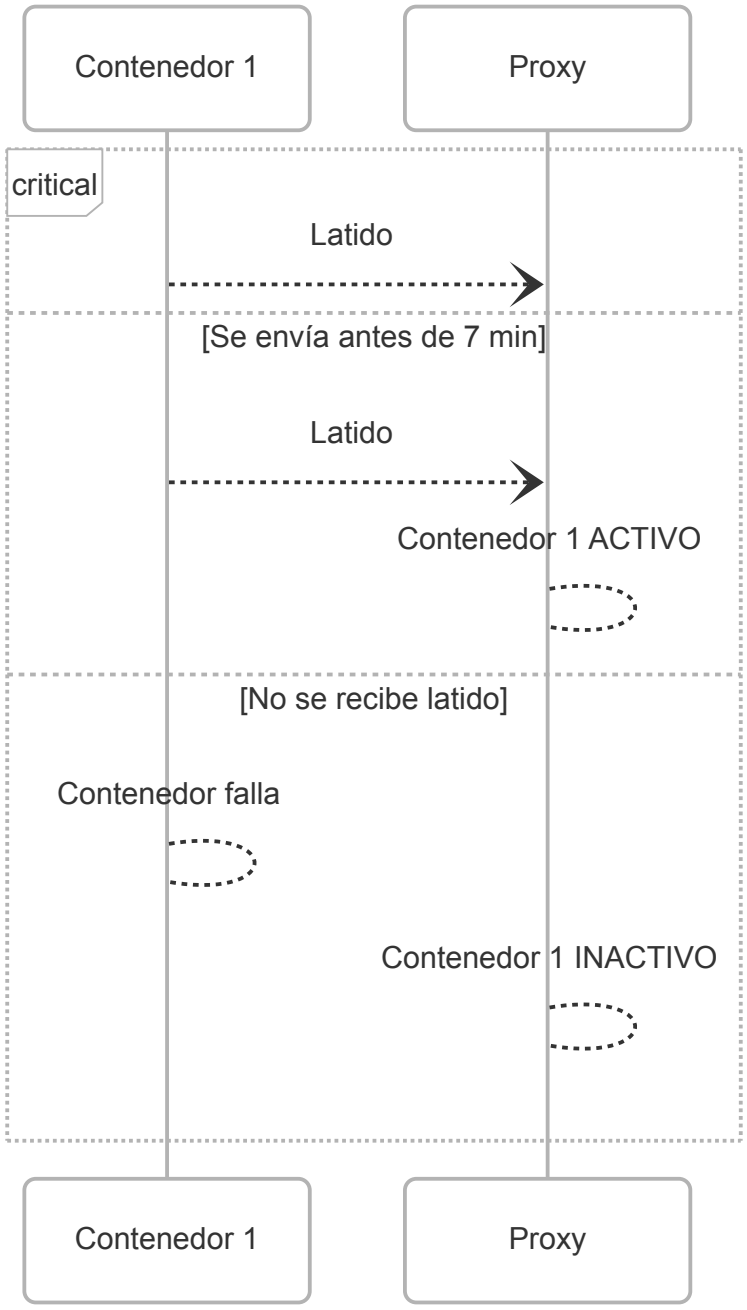

Fig. 4.6. Diagrama de secuencia de la gestión de contenedores

Diagrama de secuencia de la compilación y ejecución

En la figura 4.7 se muestra cómo el cliente envía una petición de compilación al contenedor, enviando el código a compilar. El contenedor compila el código y devuelve la salida estándar de esa compilación al cliente. Tras esto el cliente envía una petición de ejecución al contenedor, el cual ejecuta el código y devuelve la salida estándar de esa ejecución al cliente.

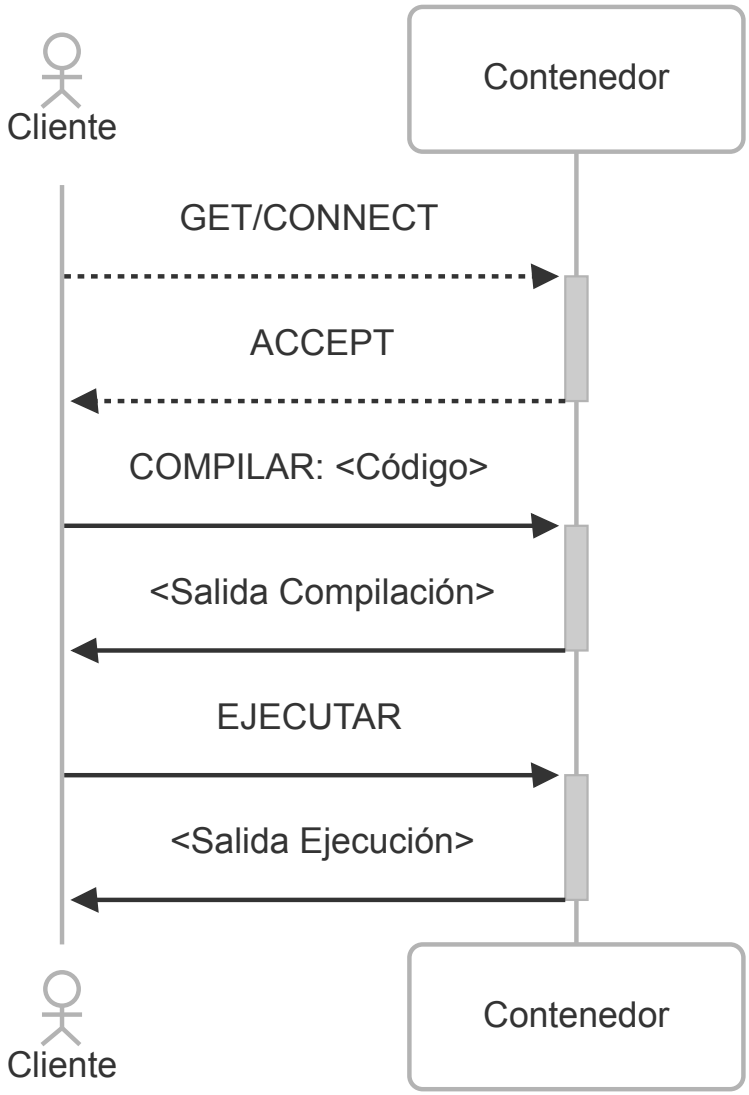

Fig. 4.7. Diagrama de secuencia de la compilación y ejecución del código

Diagrama de secuencia de la depuración

En la figura 4.8 se muestra un posible flujo de depuración. El cliente compila el código, mandando el código y los breakpoints al contenedor. Este contesta con la salida de la compilación. Tras esto el cliente envía una petición de inicio de depuración al contenedor, y el contenedor inicia la depuración hasta el primer breakpoint. El cliente recibe la información de los hilos y las variables, y muestra la información al usuario. Tras esto el cliente manda un `step` y el contenedor ejecuta la siguiente instrucción, actualizando la información de los hilos y las variables. Para finalizar la depuración, el cliente envía una petición de finalización de depuración al contenedor, el cual finaliza la sesión de depuración y devuelve la información final al cliente.

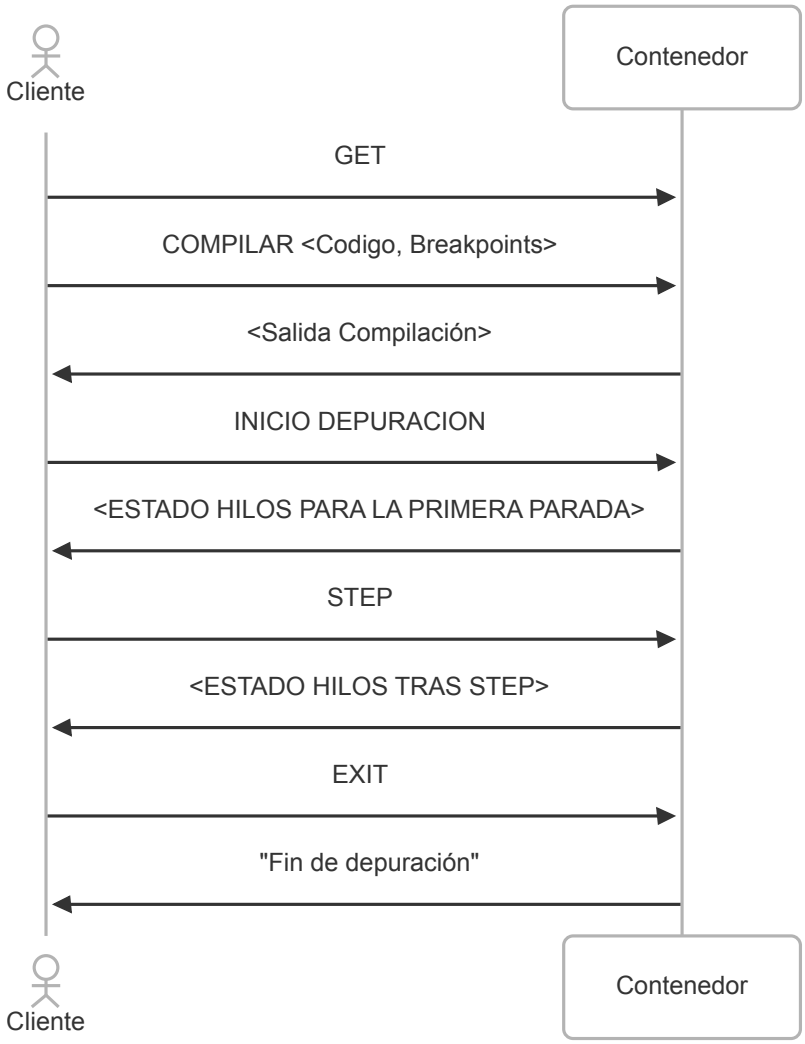

Fig. 4.8. Diagrama de secuencia de la depuración del código

Estructura de los mensajes de depuración

Tal y como se ha desarrollado en Subsección 4.3.3, *Diagrama de secuencia de la depuración*, durante la depuración los contenedores responden a las funcionalidades enviando un mensaje que contienen la información de los hilos y las variables. Esta información se almacena en los contenedores en diccionarios de Python, y se envía al cliente en formato JavaScript Object Notation (JSON). Y la estructura es la siguiente:

Es importante destacar que los hilos se actulizan según el estado de la ejecución, de manera que si un hilo no se ha ejecutado o ha finalizado, no se incluirá en el mensaje.

De esta manera, el cliente puede visualizar la información de los hilos y las variables de manera clara y estructurada. Además, se pueden añadir nuevas funcionalidades que envíen información adicional al cliente, como la información de los breakpoints o el estado de la ejecución.

```
Hilos
├── Hilo-1:
│   ├── function: "main"
│   ├── line: "10"
│   ├── code: "int main() int a = 5;..."
│   └── variables:
│       ├── a: 5
│       └── b: 10
├── Hilo-2:
│   ├── function: "funcion1"
│   ├── line: "47"
│   ├── code: "void funcion1() int c = 1;..."
│   └── variables:
│       ├── b: 10
│       └── c: 1
└── Hilo-3:
    ├── function: "funcion2"
    ├── line: "73"
    ├── code: "void funcion2() int d = 2;..."
    └── variables:
        └── d: 2
```

Fig. 4.9. Estructura de los mensajes de depuración

CAPÍTULO 5

IMPLEMENTACIÓN Y DESPLIEGUE

En este capítulo se describen los aspectos relacionados con la implementación y el despliegue del sistema. Se detallan las herramientas utilizadas (Subsección 5.1.2, *Uso de librerías y dependencias*), la estructura de ficheros del proyecto (Subsección 5.1.1, *Estructura de ficheros del proyecto*), la imagen Docker desarrollada (Subsección 5.1.3, *Construcción de imagen Docker y Docker-compose*) y el proceso de despliegue (Sección 5.2, *Despliegue*).

5.1. Implementación

Tal y como se mencionó en Subsección 4.1.7, *Lenguajes de programación*, el lado cliente será desarrollado con JavaScript, mientras que el lado servidor (*proxy* + contenedores) se desarrollará en Python. Además el lado servidor se comunicará con un contenedor docker para ejecutar de manera aislada el código desarrollado por el cliente (Subsección 4.1.4, *Sandbox*). Con esto en mente se explicará como se han organizado cada parte del proyecto, cómo se han integrado las librerías y cómo se ha construido la imagen Docker y el archivo `docker-compose.yml` para la orquestación de los contenedores.

El código al completo se encuentra disponible en el repositorio de GitHub del proyecto
.

5.1.1. Estructura de ficheros del proyecto

Se ha decidido tener un primer nivel de directorios que separen las distintas partes del proyecto. La estructura se muestra en Figura 5.1.

```
/
├── src/
├── examples/
├── test/
├── README.md
├── doc/
└── run_proyect.py
```

Fig. 5.1. Primer nivel de directorios del proyecto

En el directorio raíz del proyecto se encuentran los ficheros del código fuente en el directorio `src` y el directorio `examples` contiene ejemplos de códigos que podrían escribir los usuarios. En el directorio `test` se encuentran los ficheros necesarios para realizar las pruebas de integración y de unidad. El fichero `README.md` contiene la información necesaria para ejecutar el proyecto y el directorio `doc` se encuentra todo lo relacionado con la memoria sobre el proyecto.

A partir de aquí exploraremos el directorio del código fuente. En el directorio `src` se encuentran los ficheros de código fuente del lado cliente, del lado servidor y del contenedor docker. La estructura de este directorio se muestra en Figura 5.2.

En el directorio `proxy` se encuentra el fichero `proxy.py` que contiene la implementación del *proxy*, el fichero `requirements.txt` que contiene las dependencias necesarias y los ficheros `Dockerfile` y `docker-compose.yml` que contienen la configuración del contenedor *proxy*.

En el directorio del lado cliente `frontend` se encuentra el fichero `index.html` que contiene la estructura de la página web, el fichero `package.json` que contiene las dependencias del lado cliente, y un directorio para cada tipo de recurso (imágenes, scripts y estilos). En el directorio `docker` se encuentran los ficheros necesarios para construir la imagen Docker. En este directorio se encuentran los ficheros `Dockerfile` y `docker-compose.yml` que contiene las instrucciones para construir la imagen. También se incluye un fichero `requirements.txt` que contiene las dependencias necesarias para ejecutar el código del contenedor, además de cada uno de los ficheros `.py` que contienen la implementación de cada uno de los componentes de cada contenedor.

Por último, en el directiorio `src` se encuentra el fichero `run_proyect.sh` que permite desplegar la parte correspondiente al lado servidor.

```
src/
├── frontend/
│   ├── js/
│   │   ├── script.js
│   │   ├── code_editor.js
│   │   └── client_api.js
│   ├── images/
│   ├── css/
│   ├── index.html
│   └── package.json
├── proxy/
│   ├── proxy.py
│   ├── requirements.txt
│   ├── Dockerfile
│   └── docker_compose.yml
├── docker/
│   ├── compiler.py
│   ├── webSocket.py
│   ├── debugger.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── docker_compose.yml
└── run_back_docker.sh
```

Fig. 5.2. Estructura del directorio src

5.1.2. Uso de librerías y dependencias

En esta sección se describen las librerías y dependencias utilizadas en el proyecto, así como su propósito y cómo se integran en el sistema. Las dependencias y librerías se engloban en dos grupos: las del lado cliente y las del lado servidor.

Lado cliente

Dado que el lado cliente se desarrolla en JavaScript, se ha usado NodeJS como entorno de ejecución y gestor de paquetes. Para la gestión de dependencias se ha utilizado `npm` y `package.json` para definir las dependencias necesarias. Las principales dependencias utilizadas son:

- **Parcel** [33]: Parcel me permite empaquetar el código JavaScript y sus dependencias en un solo archivo, facilitando la carga y ejecución en el navegador.Permitiendo que

se puedan servir los ficheros desde el lado servidor a los navegadores.

- **CodeMirror** [34]: CodeMirror es una librería que proporciona un editor de código enriquecido, que permite resaltar la sintaxis y ofrece funcionalidades avanzadas.
- **Socket.io** [35]: Socket.io es una librería que permite la comunicación en tiempo real entre el cliente y el servidor a través de WebSockets. Es una librería fundamental para permitir comunicar el lado cliente con la sesión de depuración que se ejecuta en los diferentes contenedores.

Lado servidor

El lado servidor se desarrolla en Python y está conformado por el *proxy* y los contenedores. Tal y como se ha desarrollado en Subsección 4.1.4, *Sandbox*, el *proxy* se encarga de gestionar la comunicación entre el cliente y los contenedores, mientras que los contenedores se encargan de ejecutar el código del cliente. Para la construcción de los contenedores se ha usado Docker y Docker Compose, que permiten levantar unos sistemas aislados con una serie de características predefinas. Para la gestión de dependencias se ha utilizado `pip` y `requirements.txt` para definir las dependencias necesarias. Como principales librerías utizadas:

- **Flask** [36]: Flask es un microframework para Python que permite crear aplicaciones web de manera sencilla y rápida. Flask permite crear endpoints para la comunicación entre cliente, *proxy* y contenedores, tanto con *API REST*, como con *Web Socket* a través de la extensión `Flask-SocketIO`.
- **PyGDBMI** [37]: PyGDBMI es una librería que facilita el uso y la comunicación con un subproceso que ejecuta GDB. De esta manera se puede usar GDB para depurar la comunicación a través de un método `write`, donde se indica qué acción debe de realizar GDB y se obtien como resultado del método un diccionario con la salida.

5.1.3. Construcción de imagen Docker y Docker-compose

Como se ha indicado con anterioridad, los contenedores se construyen a partir de una imagen Docker, que se define en el fichero `Dockerfile`. Y los diferentes contenedores se orquestan a través de un fichero `docker-compose.yml`. Tanto el *proxy* como los contenedores tienen su propio `Dockerfile` y `docker-compose.yml`. En esta sección se explicará cómo se construyen las imágenes y cómo se orquestan los contenedores.

Proxy

La gestión del *proxy* es sencilla y tan solo es necesario crear un docker que soporte python, instalar las dependencias, abrir el puerto y ejecutar el *proxy*. El `Dockerfile` del *proxy* se muestra a continuación en la figura 5.1.

CÓDIGO 5.1. Dockerfile para el proxy

```
FROM python:3.12

WORKDIR /app

COPY requirements.txt .
COPY proxy.py .
RUN pip install -r requirements.txt

EXPOSE 8080

CMD ["python", "proxy.py"]
```

Para comunicar el *proxy* con el resto de contenedores se ha indicado en el `docker-compose.yml` el nombre del contenedor y el nombre de la red en la que se encuentra, para facilitar la conexión cuando todos los contenedores se encuentran en la misma máquina.

CÓDIGO 5.2. Docker-compose.yml para el contenedor del proxy

```
    services:
  proxy:
    container_name: proxy
    build:
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    volumes:
      - .:/app
    command: python proxy.py
    environment:
      - ALLOWED_HOSTS=*
    networks:
      - debugger_network

networks:
  debugger_network:
    external: true
```

Contendores de Depuración

En cuanto a los contenedores que se encargan de las sesiones de depuración se partirá de un único `Dockerfile`. Este `Dockerfile` se basará en una imagen de Ubuntu y se instalará todo lo necesario para poder ejecutar python y subprocesos con gdb y rr.

Además se copiarán los archivos fuente relacionados con las sesiones de depuración y los ficheros estáticos del lado cliente para servirlos.

CÓDIGO 5.3. Dockerfile para el contenedor de depuración

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
    build-essential \
    python3.12 \
    gdb \
    rr \
    # Entre otros

WORKDIR /app

COPY docker/compiler.py docker/debugger.py docker/requirements.txt
   docker/webSocket.py ./
COPY ../frontend/dist ./frontend/dist

# Configurar Python e instalar dependencias en un entorno virtual
RUN python3.12 -m venv venv && \
    . venv/bin/activate && \
    pip install --upgrade pip && \
    pip install -r requirements.txt

EXPOSE 5000

CMD ["/app/venv/bin/python3", "webSocket.py"]
```

Con la imagen preparada solo es necesario indicar cómo se deben de desplegar. Para ello se usará el fichero `/src/docker/docker-compose.yml`. Este fichero tendrá la configuración de las variables de entorno que se usan en el código, con la información sobre los host y puertos, de la red a la que se tiene que contectar y del mapeo de puertos que tiene que realizar el contenedor. Además se indicarán los recursos que podrán usar cada docker, en este caso se ha decidido que solo puedan usar 4 hilos y 128MB reservados para ese contenedor, con hasta un máximo de 256MB de memoria dinámica. En la figura 5.4 se muestra un fragmento del `docker-compose.yml` que corresponde con la configuración de un contenedor de depuración llamado `debugger1`.

CÓDIGO 5.4. Docker-compose.yml para un contenedor de depuración

```
services:
    debugger1:
        build:
            context: ../
            dockerfile: docker/Dockerfile
        environment:
            - PROXY_URL=http://proxy:8080
            - HOST=debugger1
            - PORT=5000
        ports:
            - "5001:5000"
```

```
        networks:
            - debugger_network
        volumes:
            - ./compiler.py:/app/compiler.py
            - ./debugger.py:/app/debugger.py
            - ./webSocket.py:/app/webSocket.py
            - ../frontend/dist:/app/dist
        deploy:
            resources:
                limits:
                cpus: '4.0'
                memory: 256M
                reservations:
                    memory: 128M
```

5.2. Despliegue

En el siguiente diagrama Figura 5.3 se muestra el diagrama de despliegue del sistema siguiendo el estándar UML [30].

Fig. 5.3. Diagrama de despliegue del sistema

El despliegue del lado puede ser desplagado separando en distintas máquinas el *proxy* y los contenedores, o bien desplegando todo en la misma máquina. En este caso se ha decidido desplegar el *proxy* y los contenedores en la misma máquina, pero se puede desplegar de manera separada. Por esta razón las especificaciones técnicas se han separado en los 3 principales componentes aunque el despliegue se realice en la misma máquina.

Las especificaciones técnicas recomendadas para el despliegue del sistema son las siguientes:

Especificación	**Servicio de Sesiones de Depuración**	**Máquina Proxy**	**Máquina de Lado Cliente**
CPU	4 núcleos × (número de sesiones) Intel Core i7-13700 or superior	Intelő CoreTM i3 CPU 6300	CPU con al menos 2 núcleos
RAM	512 Mb × (número de sesiones) 4 GB o superior	2 GB o superior	2GB o superior
Almacenamiento	2 GB × (número de sesiones)	2 GB o superior	100 MB
Sistema Operativo	Ubuntu 24.04 LTS	Ubuntu 24.04 LTS	Cualquiera

Todas las máquinas deben de estar conectadas a Internet y tener acceso a los puertos usados en los contenedores.

El software necesario debe de ser el siguiente:

- **Lado Servidor** (Proxy + Contenedores):
  - Docker
  - Docker Compose
- **Lado Cliente**:
  - Navegador web moderno (Chrome, Firefox, Edge, etc.)
  - NodeJS
  - npm

Para poder desplegar el proyecto se debe de seguir la guía indicada en Apéndice A. En resumen, para realizar un despligue de prueba se debe de ejecutar el script que se encuentra en la raíz del proyecto `start_proyect.sh`. Sin embargo, si se quiere hacer un despligue más específico, se puede desplegar solo el lado servidor a través del script `run_back_docker.sh` que se encuentra en el directorio `src`. Este script ejecutará el *proxy* y los contenedores de depuración.

CAPÍTULO 6

VALIDACIÓN Y VERIFICACIÓN

En este capitulo se presentará la validación y verificación del sistema desarrollado. El objetivo de esta sección es comprobar que el sistema cumple con los requisitos establecidos en el capitulo Capítulo 3, *Análisis*.

La verificación busca comprobar que el sistema cumple con los requisitos establecidos en la fase de análisis [38], mientras que la validación busca comprobar que el sistema cumple con las expectativas del cliente [38]. Es por esto que se desarrollarán dos tipos de pruebas: pruebas de verificación (Sección 6.1, *Pruebas de verificación*) y pruebas de validación (Sección 6.2, *Pruebas de validación*).

Para ambas pruebas se han diseñado casos de prueba que servirán para comprobar el correcto funcionamiento del sistema. Cada uno de los casos de prueba cuenta con un idenificador único, con el formato *YYY-XX*, donde la primera parte corresponde al tipo de prueba (*VET* para verificación y *VAT* para validación). La siguiente tabla proporciona una plantilla para la descripción de los casos de prueba:

Cada caso de prueba se identifica con un ID que sigue el formato *YYY-XX*, donde *XX* identifica el número de secuencia del caso de prueba, empezando en *01*, y *YYY* representa el tipo, ya sea *VET* (verificación) o *VAT* (validación).

La tabla 6.1 contiene la plantilla utilizada para la especificación de los casos de prueba, incluyendo la descripción de cada atributo.

6.1. Pruebas de verificación

Para conseguir verificar que el sistema cumple con los requisitos establecidos se han diseñado los siguientes casos de prueba.

En la tabla Tabla 6.9 se presenta la tabla de trazabilidad entre los casos de prueba de

TABLA 6.1
PLANTILLA DE PRUEBA

YYY-XX	
Descripción	Descripción del test.
Pre-condición	Condiciones que deben cumplirse para realizar el test.
Procedimiento	Descripción de los pasos a seguir para realizar el test.
Post-condición	Condiciones que deben cumplirse después de realizar el test para pasarlo.
Origen	Requisitos que originaron el test.
Evaluación	Resultado del test (*OK* o *Error*).

verificación y los requisitos de software del sistema. Esta tabla permite comprobar que todos los requisitos de software han sido cubiertos por al menos un caso de prueba.

TABLA 6.2
PRUEBA VET-01

VET-01	
Descripción	Escribir código en C en el editor
Pre-condición	El sistema se encuentra en ejecución
Procedimiento	1. Acceder a la URL del sistema desde el navegador 2. Escribir código en C 3. Comprobar que el código se muestra con resaltado de sintaxis
Post-condición	El editor debe de mostrar el código escrito con resaltado de sintaxis
Origen	RS-FN-01, RS-FN-02, RS-FN-03, RS-FN-18, RS-NF-10
Evaluación	OK

TABLA 6.3
PRUEBA VET-02

VET-02	
Descripción	Compilar y ejecutar código en C
Pre-condición	El sistema se encuentra en ejecución y se accede a la URL
Procedimiento	1. Escribir código en C 2. Compilar el código 3. Consultar la salida de la compilación 4. Ejecutar el código 5. Consultar la salida de la ejecución
Post-condición	El sistema debe de crear un fichero que contenga el código escrito compilado y un fichero ejecutable
Origen	RS-FN-04, RS-FN-05, RS-FN-06, RS-FN-08, RS-FN-18, RS-FN-17, RS-FN-19, RS-NF-01, RS-NF-02, RS-NF-03, RS-NF-05, RS-NF-06, RS-NF-09, RS-NF-10, RS-NF-11
Evaluación	OK

TABLA 6.4
PRUEBA VET-03

VET-03	
Descripción	Depuración con breakpoints
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL y tener un código en C escrito
Procedimiento	1. Colocar un breakpoint en el código 2. Compilar el código 3. Pasar a modo depuración 4. Comenzar la depuración 5. Observar que el programa se ha detenido en el breakpoint 6. Comprobar que se muestra la información al completo del estado del programa en el momento de la parada 7. Continuar la ejecución del programa 8. Comprobar que se ha finalizado la ejecución del programa
Post-condición	N/A
Origen	RS-FN-03, RS-FN-07, RS-FN-12, RS-FN-14, RS-FN-15, RS-FN-17, RS-FN-19, RS-NF-03, RS-NF-04, RS-NF-05, RS-NF-06, RS-NF-07, RS-NF-08, RS-NF-09, , RS-NF-11
Evaluación	OK

TABLA 6.5
PRUEBA VET-04

VET-04	
Descripción	Depuración con paso a paso
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con un breakpoint y encontrarse en modo depuración
Procedimiento	1. Comenzar la depuración 2. Observar que el programa se ha detenido en el breakpoint 3. Realizar un step-over del hilo seleccionado 4. Comprobar que solo ha avanzado el hilo seleccionado 5. Realizar un step-into del hilo seleccionado 6. Comprobar que se ha entrado en la función del código del hilo seleccionado 7. Realizar un step-out del hilo seleccionado 8. Comprobar que se ha salido de la función del código del hilo seleccionado
Post-condición	N/A
Origen	RS-FN-03, RS-FN-07, RS-FN-09, RS-FN-10, RS-FN-11, RS-FN-14, RS-FN-15, RS-FN-17, RS-FN-19, RS-NF-03, RS-NF-04, RS-NF-05, RS-NF-06, RS-NF-07, RS-NF-08, RS-NF-09, RS-NF-10, RS-NF-11
Evaluación	OK

TABLA 6.6
PRUEBA VET-05

VET-05	
Descripción	Depuración con breakpoints con rebobinado
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con varios breakpoints y encontrarse en modo depuración
Procedimiento	1. Iniciar la depuración 2. Observar que el programa se ha detenido en el breakpoint 3. Comprobar que se muestra la información al completo del estado del programa en el momento de la parada 4. Avanzar la ejecución del programa hasta el siguiente breakpoint 5. Observar que el programa se ha detenido en el siguiente breakpoint 6. Comprobar que se muestra la información al completo del estado del programa en el momento de la parada 7. Rebobinar la ejecución, volviendo al breakpoint anterior 8. Comprobar que se la información que se muestra es la misma que en el momento de la anterior parada
Post-condición	N/A
Origen	RS-FN-03, RS-FN-07, RS-FN-13, RS-FN-14, RS-FN-15, RS-FN-17, RS-FN-19, RS-NF-03, RS-NF-04, RS-NF-05, RS-NF-06, RS-NF-07, RS-NF-08, RS-NF-09, RS-NF-10, RS-NF-11
Evaluación	OK

TABLA 6.7
PRUEBA VET-06

VET-06	
Descripción	Depuración paso a paso con rebobinado
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con un breakpoint y encontrarse en modo depuración
Procedimiento	1. Iniciar la depuración 2. Avanzar la ejecución del programa con un step-over 3. Rebobinar la ejecución con un rebobinado del step-over 4. Comprobar que el programa se encuentra en el mismo estado que antes de realizar el step-over 5. Avanzar la ejecución del programa con un step-into 6. Rebobinar la ejecución con un rebobinado del step-into 7. Comprobar que el programa se encuentra en el mismo estado que antes de realizar el step-into 8. Avanzar la ejecución del programa con un step-out 9. Rebobinar la ejecución con un rebobinado del step-out 10. Comprobar que el programa se encuentra en el mismo estado que antes de realizar el step-out
Post-condición	N/A
Origen	RS-FN-03, RS-FN-07, RS-FN-13, RS-FN-14, RS-FN-15, RS-FN-17, RS-FN-19, RS-NF-03, RS-NF-04, RS-NF-05, RS-NF-06, RS-NF-07, RS-NF-08, RS-NF-09, RS-NF-10, RS-NF-11
Evaluación	OK

TABLA 6.8
PRUEBA VET-07

VET-07	
Descripción	Evolución de las variables globales durante la ejecución del programa
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL
Procedimiento	1. Tener un código en C donde varios hilos accedan a una variable global 2. Colocar varios breakpoints en el código 3. Iniciar la depuración 4. Observar que el programa se ha parado en el primer breakpoint y que la variable global ya ha sido modificada por un hilo 5. Avanzar la ejecución del programa hasta el siguiente breakpoint 6. Observar que el programa se ha parado en el segundo breakpoint y que la variable global ya ha sido modificada por otro hilo, mostrando el valor anterior y el nuevo valor
Post-condición	N/A
Origen	RS-FN-07, RS-FN-14, RS-FN-15, RS-FN-16, RS-FN-18, RS-NF-03, RS-NF-04, RS-NF-05, RS-NF-06, RS-NF-07, RS-NF-08, RS-NF-09, RS-NF-10
Evaluación	OK

TABLA 6.9

TRAZABILIDAD DE LOS CASOS DE PRUEBA DE VERIFICACIÓN RESPECTO A LOS REQUISITOS FUNCIONALES

	RS-FN-01	RS-FN-02	RS-FN-03	RS-FN-04	RS-FN-05	RS-FN-06	RS-FN-07	RS-FN-08	RS-FN-09	RS-FN-10	RS-FN-11	RS-FN-12	RS-FN-13	RS-FN-14	RS-FN-15	RS-FN-16	RS-FN-17	RS-FN-18	RS-FN-19	RS-NF-01	RS-NF-02	RS-NF-03	RS-NF-04	RS-NF-05	RS-NF-06	RS-NF-07	RS-NF-08	RS-NF-09	RS-NF-10	RS-NF-11
VET-01	•	•	•														•											•		
VET-02				•	•	•		•									•	•	•	•	•	•		•	•			•	•	•
VET-03			•				•					•		•	•		•		•			•	•	•	•	•	•	•		•
VET-04			•				•		•	•	•			•	•		•		•			•	•	•	•	•	•	•	•	•
VET-05			•				•						•	•	•		•		•			•	•	•	•	•	•	•	•	•
VET-06			•				•						•	•	•		•		•			•	•	•	•	•	•	•	•	•
VET-07							•							•	•	•		•					•	•	•	•	•	•	•	•	

6.2. Pruebas de validación

Para conseguir validar que el sistema cumple con las expectativas del cliente se han diseñado los siguientes casos de prueba.

TABLA 6.10
PRUEBA VAT-01

VAT-01	
Descripción	La herramienta debe de permitir la edición de código en C
Pre-condición	El sistema se encuentra en ejecución
Procedimiento	1. Acceder a la URL del sistema desde el navegador de cualquier dispositivo 2. Escribir código en C 3. Comprobar que el código se muestra con resaltado de sintaxis
Post-condición	N/A
Origen	RU-CA-02, RU-CA-03, RU-RE-03, RU-RE-10
Evaluación	OK

En la tabla Tabla 6.18 se presenta la tabla de trazabilidad entre los casos de prueba de validación y los requisitos de usuario del sistema. Esta tabla permite comprobar que todos los requisitos de usuario han sido cubiertos por al menos un caso de prueba.

TABLA 6.11
PRUEBA VAT-02

VAT-02	
Descripción	La herramienta debe de permitir la compilación y ejecución de código en C
Pre-condición	El sistema se encuentra en ejecución y se accede a la URL
Procedimiento	1. Escribir código en C 2. Compilar el código 3. Consultar la salida de la compilación 4. Comprobar que se muestra un aviso en caso de error de concurrencia 5. Ejecutar el código 6. Consultar la salida de la ejecución 7. Comprobar que se ha ejecutado correctamente
Post-condición	N/A
Origen	RU-CA-04, RU-CA-05, RU-CA-07, RU-RE-01, RU-RE-02, RU-RE-04, RU-RE-05, RU-RE-06
Evaluación	OK

TABLA 6.12
PRUEBA VAT-03

VAT-03	
Descripción	La herramienta debe de avisar en caso de errores de concurrencia
Pre-condición	El sistema se encuentra en ejecución y se accede a la URL
Procedimiento	1. Escribir código en C con errores de concurrencia 2. Compilar el código 3. Comprobar que se muestra un aviso en caso de error de concurrencia
Post-condición	N/A
Origen	RU-CA-04, RU-CA-07, RU-RE-01, RU-RE-04, RU-RE-05, RU-RE-06
Evaluación	OK

TABLA 6.13
PRUEBA VAT-04

VAT-04	
Descripción	La herramienta debe de permitir la depuración de código en C
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL y tener un código en C escrito
Procedimiento	1. Colocar un breakpoint en el código 2. Compilar el código 3. Pasar a modo depuración 4. Comenzar la depuración 5. Observar que el programa se ha detenido en el breakpoint 6. Comprobar que se muestra la información al completo del estado del programa en el momento de la parada 7. Continuar la ejecución del programa 8. Comprobar que se ha finalizado la ejecución del programa
Post-condición	N/A
Origen	RU-CA-01, RU-CA-06, RU-CA-08, RU-CA-11, RU-CA-12, , RU-RE-04, RU-RE-05, RU-RE-07, RU-RE-08
Evaluación	OK

TABLA 6.14
PRUEBA VAT-05

VAT-05	
Descripción	La herramienta debe de permitir la depuración paso a paso de código en C
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con un breakpoint y encontrarse en modo depuración
Procedimiento	1. Comenzar la depuración 2. Observar que el programa se ha detenido en el breakpoint 3. Continuar la ejecución del programa avanzando un paso 4. Comprobar que se ha avanzado un paso en la ejecución del programa para el hilo seleccionado
Post-condición	N/A
Origen	RU-CA-01, RU-CA-06, RU-CA-09, RU-CA-11, RU-CA-12, , RU-RE-04, RU-RE-05, RU-RE-07, RU-RE-08
Evaluación	OK

TABLA 6.15
PRUEBA VAT-06

VAT-06	
Descripción	La herramienta debe de permitir retroceder la ejecución del programa durante la depuración
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con varios breakpoints y encontrarse en modo depuración
Procedimiento	1. Iniciar la depuración 2. Observar que el programa se ha detenido en el primer breakpoint 3. Avanzar la ejecución del programa hasta el siguiente breakpoint 4. Observar que el programa se ha detenido en el segundo breakpoint 5. Rebobinar la ejecución, volviendo al breakpoint anterior 6. Comprobar que la información que se muestra es la misma que en el momento de la anterior parada
Post-condición	N/A
Origen	RU-CA-01, RU-CA-06, RU-CA-11, RU-CA-12, RU-CA-10, , RU-RE-04, RU-RE-05, RU-RE-07, RU-RE-08
Evaluación	OK

TABLA 6.16
PRUEBA VAT-07

VAT-07	
Descripción	La herramienta debe de permitir ver la evolución de las variables globales durante la ejecución del programa
Pre-condición	El sistema se encuentra en ejecución, acceder a la URL, tener un código en C compilado con un breakpoint y encontrarse en modo depuración
Procedimiento	1. Tener un código en C donde varios hilos accedan a una variable global 2. Colocar varios breakpoints en el código 3. Iniciar la depuración 4. Observar que el programa se ha parado en el primer breakpoint y que la variable global ya ha sido modificada por un hilo 5. Avanzar la ejecución del programa hasta el siguiente breakpoint 6. Observar que el programa se ha parado en el segundo breakpoint y que la variable global ya ha sido modificada por otro hilo, mostrando el valor anterior y el nuevo valor
Post-condición	N/A
Origen	RU-CA-01, RU-CA-06, RU-CA-11, RU-CA-12, RU-CA-13, , RU-RE-04, RU-RE-05, RU-RE-07, RU-RE-08
Evaluación	OK

TABLA 6.17
PRUEBA VAT-08

VAT-08	
Descripción	Un mal uso de la herramienta no debe de afectar al sistema
Pre-condición	La herramienta se encuentra en ejecución y se accede a la URL
Procedimiento	1. Se debe de escribir un código malicioso en el editor, como por ejemplo una *bomba fork* 2. Compilar el código 3. Ejecutar el código 4. Comprobar que el sistema no se ve afectado por el código malicioso
Post-condición	N/A
Origen	RU-CA-01, RU-CA-04, RU-CA-05, RU-RE-09
Evaluación	OK

TABLA 6.18

TRAZABILIDAD DE LOS CASOS DE PRUEBA DE VALIDACIÓN RESPECTO A LOS REQUISITOS DE USUARIO

	RU-CA-01	RU-CA-02	RU-CA-03	RU-CA-04	RU-CA-05	RU-CA-06	RU-CA-07	RU-CA-08	RU-CA-09	RU-CA-10	RU-CA-11	RU-CA-12	RU-CA-13	RU-RE-01	RU-RE-02	RU-RE-03	RU-RE-04	RU-RE-05	RU-RE-06	RU-RE-07	RU-RE-08	RU-RE-09	RU-RE-10
VAT-01		•	•												•								•
VAT-02				•	•		•							•	•		•	•	•				
VAT-03				•			•							•			•	•	•				
VAT-04	•					•		•			•	•					•	•		•	•		
VAT-05	•					•			•		•	•					•	•		•	•		
VAT-06	•					•				•	•	•					•	•		•	•		
VAT-07	•					•					•	•	•				•	•		•	•		
VAT-08	•			•	•																	•	

CAPÍTULO 7

PLANIFICACIÓN Y PRESUPUESTO

En este capítulo se presentará una visión general del desarrollo y la logística del proyecto. En la sección Sección 7.1, *Planificación* se presentará la planificación del proyecto incluyendo cronograma y la distribución de tareas. Además, se describirá el presupuesto y costes del proyecto (Sección 7.2, *Presupuesto*)

7.1. Planificación

7.1.1. Metodología

Dadas las características del proyecto, se ha decidido utilizar una metodología en cascada [39] para la planificación y desarrollo del proyecto. Esta metodología se adapta correctamente a un proyecto de una envergadura mediana, ya que permite centrarse en cada una de las fases del proyecto de forma secuencial, y permite una mayor claridad en la planificación y desarrollo del proyecto.

El proyecto se dividirá en 4 grandes etapas:

1. **Planificación**: Se obtienen los requisitos de usuario y se evaluan para obtener los objetivos de esa iteración.

2. **Análisis**: Se identifican y evaluan las alternativas que pueden satisfacer los objetivos fijados.

3. **Desarrollo y Pruebas**: Se diseña, desarrolla y prueba la arquitectura propuesta.

4. **Evaluación**: El cliente evalua el sistema aportado *feedback*, que se usará como información valiosa en las siguientes iteraciones.

Para cada una de las etapas se realizará el respectivo desarrollo de los siguientes componentes:

1. **Sesión de Depuración**: Se desarrolla el módulo que permite depurar código concurrente a través de GDB/RR, con cada una de las funcionalidades necesarias.
2. **Depuración en Contenedores**: Se desarrolla la capacidad de levantar contenedores Docker para ejecutar las sesiones de depuración.
3. **Proxy**: Se desarrolla el proxy que permite la redirección del cliente a las sesiones de depuración, así como la gestión de sesiones de depuración.
4. **Comunicaciones**: Se desarrolla la comunicación entre el cliente y el servidor a través de WebSocket.
5. **Lado Cliente**: Se desarrolla la interfaz web que permite al usuario interactuar con el sistema.

7.1.2. Estimación de tiempo y cronograma

La estimación de tiempo se ha realizado diseñando un diagrama de Gantt (Figura 7.1) que permite visualizar la planificación del proyecto. Este diagrama muestra las diferentes fases del proyecto, así como el desarrollo de los distintos componentes para cada una de ellas. Además, se han establecido fechas de inicio y fin para cada tarea, lo que permite tener una visión clara del tiempo necesario para completar el proyecto.

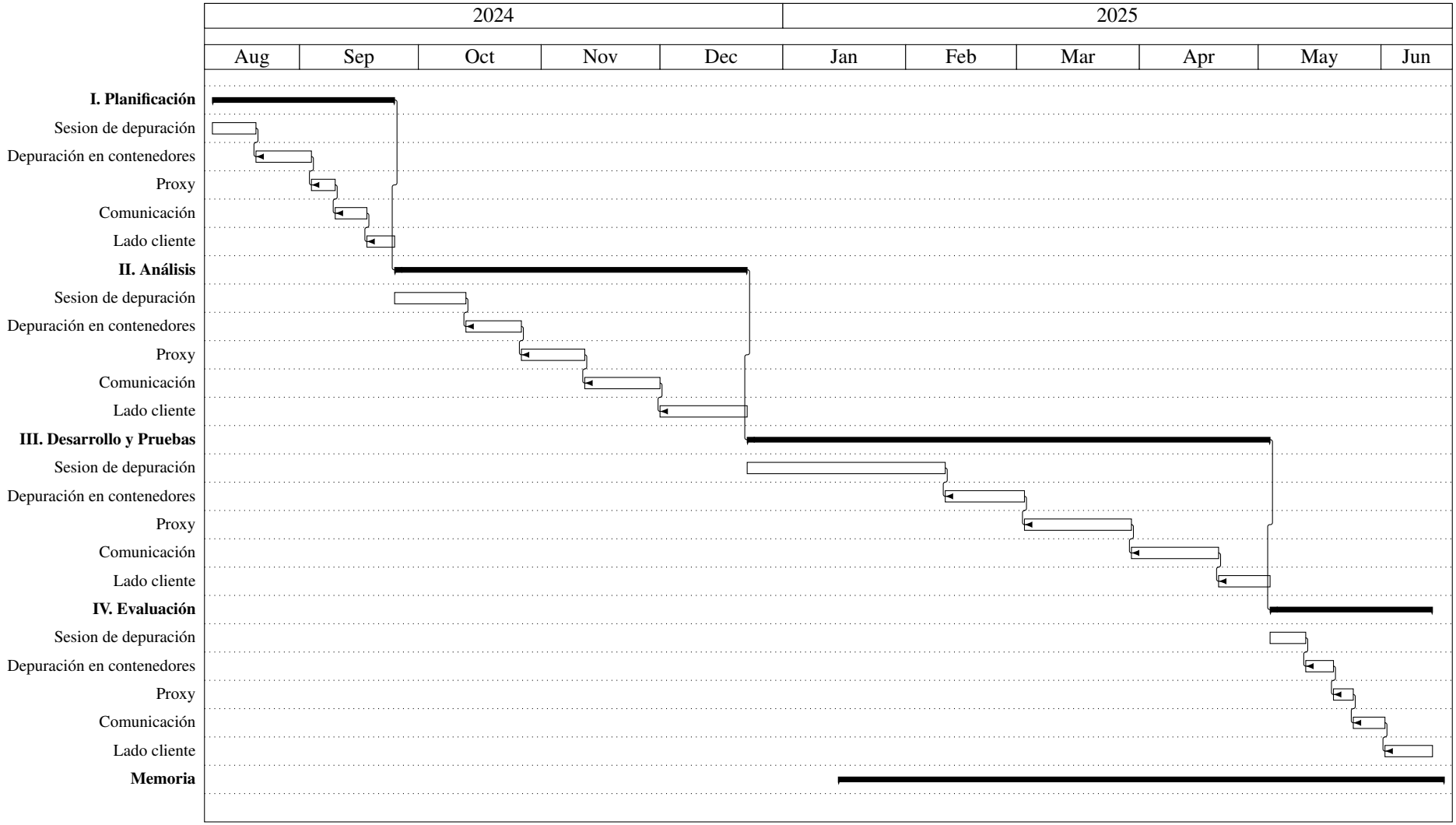

Fig. 7.1. Diagram de Gantt del proyecto.

7.2. Presupuesto

En esta sección se presentará el presupuesto del proyecto, basado en la estimación de tiempo y planificación presentada en la sección Sección 7.1.2, *Estimación de tiempo y cronograma*.

TABLA 7.1
RESUMEN DEL PRESUPUESTO DEL PROYECTO

Titulo	Herramienta Didáctica pra la Programación Concurrente
Autor	Adrián Fernández Galán
Fecha de Inicio	10-08-2024
Fecha Final	16-06-2025
Duración	10 meses
Presupuesto	12.702,30€

El desglose de los costes se dividirá en dos costes: costes directos y costes indirectos. Estos costes no incluyen impuestos, sino que se incluirán en Subsección 7.2.4, *Oferta propuesta*

7.2.1. Costes directos

Los costes directos son aquellos que están directamente relacionados con el desarrollo del proyecto. Estos se dividen en dos subgrupos: costes de personal y costes de equipamiento.

- **Costes de personal**: Estos costes dependerán de los roles, las responsabilidades y el tiempo de desarrollo de cada uno de los miembros
- **Costes de equipamiento**: Estos costes están relacionados con las herramientas usadas durante el desarrollo del proyecto

Costes de personal

Durante el desarrollo del proyecto han sido necesarios 4 roles diferentes:

- **Jefe de Proyecto**: Su tarea principal es planificar y coordinar el proyecto.
- **Analista**: Se encarga de analizar los requisitos de usuario y diseñar la arquitectura del proyecto.
- **Desarrollador**: Su tarea es implementar las funcionalidades indicadas en los requisitos de software.

- **Evaluador**: Se encarga de diseñar y ejecutar las pruebas del sistema, así como de evaluar el sistema y aportar *feedback* al equipo de desarrollo.

En la tabla Tabla 7.2 se presenta el desglose de los costes de personal del proyecto. Estos costes se han calculado en base a las horas estimadas de trabajo y el coste por hora de cada uno de los roles. El coste total del personal del proyecto es de 10.650€.

TABLA 7.2
COSTES DE PERSONAL DEL PROYECTO

Rol	Horas	Coste por Hora	Total
Jefe de Proyecto	45h	60€	2.700€
Analista	70h	35€	2.450€
Desarrollador	150h	32€	4.800€
Evaluador	35h	20€	700€
Total	300h		**10.650€**

Costes de equipamiento

Para el desarrollo del proyecto se han necesitado diferentes herramientas y recursos. Tabla 7.3 muestran los costes de equipamiento del proyecto. Estos costes se han calculado en base a la (7.1).

$$C = \frac{c \cdot u \cdot t}{a} \tag{7.1}$$

Donde:

- C: Coste de amortización del recurso
- c: Coste del recurso
- u: Uso del recurso (horas)
- t: Tiempo de amortización del recurso (horas)
- a: Amortización del recurso (horas)

7.2.2. Costes indirectos

Los costes indirectos son aquellos que ocurren durante el proyecto, pero no pueden ser directamente relacionados con un producto o servicio específico.

Para el consumo de electricidad, se ha revisado la factura de la vivienda durante los 10 meses de desarrollo del proyecto, la cual ha sido de 2,13€el día. Teniendo en cuenta

TABLA 7.3
COSTES DE EQUIPAMIENTO DEL PROYECTO

Concepto	**Coste (c)**	**Uso (u)**	**Tiempo usado (t)**	**Amortización (a)**	**Coste Amortizado (C)**
Portátil	900,00€	45 %	10 meses	48 meses	84,38€
Monitor	125,95€	60 %	10 meses	60 meses	12,60€
Teclado	50,00€	60 %	10 meses	36 meses	8,33€
Silla Oficina	120,00€	40 %	10 meses	60 meses	8,00€
Software	0,00€	40 %	10 meses	120 meses	0,00€
Total	1.195,95€				**113,30€**

que el proyecto ha durado 10 meses, y que se han trabajado 30 horas al mes, el coste total de electricidad es de 2,13€$día \cdot 30\ días \cdot 10\ meses$ = 639€.

En cuanto a los costes de internet, se tiene contratada en la vivienda una tarifa de 50€al mes. Y en cuanto al transporte, se ha estimado en torno a 80€al mes para los viajes realizados al centro educativo.

El resultado de todos los costes indirectos se puede ver en la tabla Tabla 7.4.

TABLA 7.4
COSTES INDIRECTOS DEL PROYECTO

Recurso	Coste Unitario	Tiempo	**Total**
Electricidad	2,13€/día	10 meses	639€
Internet	50€	10 meses	500€
Transporte	80€	10 meses	800€
Total			1.939€

7.2.3. Costes totales

Los costes totales del proyecto constan de la suma de los costes directos, tanto de los costes de personal como de los costes de equipamiento, y los costes indirectos. El resultado de la suma de todos los costes se puede ver en la tabla Tabla 7.5.

TABLA 7.5
COSTES TOTALES DEL PROYECTO

Personal	10.650,00€
Equipamiento	113,30€
Indirecto	1.939,00€
Total Proyecto	**12.702,30€**

Es por esto que el coste total del proyecto es de **12.702,30€**.

7.2.4. Oferta propuesta

Conociendo el coste total del proyecto, se ha decidido realizar una oferta al cliente. Esta oferta contempla un riesgo del 15 % sobre el coste total y un margen de beneficio del 20 %. Tras esto se le aplicará el IVA correspondiente del 21 %. El resultado de la oferta se puede ver en la tabla Tabla 7.6.

TABLA 7.6
OFERTA PROPUESTA AL CLIENTE

Concepto	**Incremento**	**Coste Parcial**	**Coste Agregado**
Coste del Proyecto	-	12.702,30€	12.702,30€
Riesgo	15 %	1.905,35€	14.607,65€
Beneficio	20 %	2.921,53€	17.529,18€
Impuestos	21 %	3.681,13€	21.210,31€
Total	56 %		**21.210,31€**

CAPÍTULO 8

MARCO REGULATORIO Y ENTORNO SOCIO-ECONÓMICO

En este capítulo se describirá cómo el proyecto se ve afectado por el marco regulatorio y el entorno socio-económico.

8.1. Marco regulatorio

En esta sección se van a exponer los estándares de las herramientas usadas en el proyecto, así como las licencias bajo las que se distribuyen los distintos softwares utilizados en el proyecto.

8.1.1. Estándares

Tal y como se ha discutido en la sección Subsección 4.1.7, *Lenguajes de programación*, el proyecto se ha desarrollado principalmente con Python y JavaScript.

Python es un lenguaje de programación que sigue el estándar PEP 8 [40], que es una guía de estilo para escribir código Python. Este estándar establece convenciones sobre la indentación, el uso de espacios en blanco, la longitud de las líneas, los nombres de las variables y funciones, entre otros aspectos. El objetivo de PEP 8 es mejorar la legibilidad del código y facilitar su mantenimiento.

JavaScript, por su parte, sigue el estándar ECMAScript [41], que es un estándar de scripting que define la sintaxis, tipos de datos, estructuras de control, objetos y funciones del lenguaje. Este estándar es fundamental para garantizar la interoperabilidad entre diferentes implementaciones de JavaScript en navegadores y entornos de ejecución.

8.1.2. Licencias de software

Tal y como se ha mencionado en las secciones Subsección 4.1.7, *Lenguajes de programación* y Subsección 5.1.2, *Uso de librerías y dependencias*, el proyecto principalmente en Python, JavaScript y Docker.

Las principales librerías que se han usado en Python son:

- Flask: Es un módulo de Python que usa la licencia la licencia *3-Clause BSD* [36].
- PyGDBMI: Es un módulo de Python que usa la licencia *MIT* [37].

La principal librería que se ha usado en JavaScript son:

- Parcel: Es una herramienta que usa la licencia MIT [33].
- CodeMirror: Esta librería usa la licencia MIT [34].
- Socket.io: Esta librería usa la licencia MIT [35].

Además, docker es un software que usa la licencia Apache 2.0 [42]. El depurador *GDB* es un software que usa la licencia GPLv3 [8] y el depurador *RR* usa una combinación de varias licencias, pero la mayor parte del código tiene la licencia MIT [43].

Dado que todas las librerías y softwares usados en el proyecto tienen licencias permisivas, a excepción de *GDB* (que al no integrar parte del código fuente en el proyecto no se considera obra derivada), no es necesario incluir ninguna mención especial en el proyecto.

El proyecto se distribuye bajo la licencia *MIT* [44], que es una licencia permisiva que permite a los usuarios usar, copiar, modificar y distribuir el software bajo ciertas condiciones. Esta licencia es ampliamente utilizada en la comunidad de software libre y de código abierto.

8.1.3. Normativas

En este apartado se van a exponer las normativas que afectan al proyecto. En este caso, dado que el proyecto se tiene un alcance reducido estará sujeto al *Reglamento General de Protección de Datos* (RGPD) [45].

Dado el carácter intrínseco del proyecto, debe de existir una vinculación entre el usuario y las sesiones de depuración. Para ello, el navegador busca una cookie de sesión y en el caso de no existir se crea un nuevo identificador unívoco. Este identificador se almacena en el navegador y se envía al servidor cada vez que se realiza una petición `GET` al proxy. Este proxy utiliza el identificador para conocer la sesión de depuración a la que pertenece la petición. En el caso de que el usuario cierre el navegador, la cookie de sesión

se elimina tras 10 min, lo que hará que el identificador de sesión se vuelva inválido. En el caso de que el usuario cierre la sesión, la cookie de sesión se elimina inmediatamente y el identificador de sesión se vuelve inválido.

Este proceso encaja con el artículo 6.1.b del RGPD, que establece que el tratamiento de datos personales es lícito si es necesario para la ejecución de un contrato en el que el interesado es parte o para la aplicación a petición de este de medidas precontractuales [46].

Además, el artículo 5.1 del RGPD establece que los datos personales deben ser tratados de manera lícita, leal y transparente en relación con el interesado [47]. En este caso, el tratamiento de datos personales se realiza de manera lícita y transparente, ya que el usuario es informado de la creación de la cookie de sesión y del uso del identificador de sesión para la gestión de la depuración.

8.2. Entorno socio-económico

El análisis del entorno socio-económico es fundamental para comprender la relevancia y el impacto potencial del proyecto. En esta sección, se examinará el impacto socio-económico del proyecto, así como su alineación con los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas, y se desarrollará un plan de explotación del proyecto.

8.2.1. Impacto socio-económico

El impacto socio-económico del proyecto se manifiesta en varios niveles, desde la educación y la formación de los estudiantes hasta la mejora de la calidad del software desarrollado en el ámbito profesional. A continuación, se detallan los principales aspectos del impacto esperado:

Educación y formación

El proyecto tiene como objetivo principal mejorar la enseñanza y el aprendizaje de la programación concurrente, una habilidad esencial en el desarrollo de software moderno. Al proporcionar una herramienta didáctica que facilita la comprensión de conceptos complejos, se espera que los estudiantes adquieran competencias más sólidas en esta área. Esto no solo mejora su formación académica, sino que también aumenta su empleabilidad en un mercado laboral cada vez más demandante de profesionales con habilidades en programación concurrente.

Mejora de la calidad del software

La programación concurrente es crucial para el desarrollo de aplicaciones eficientes y escalables. Al capacitar a los estudiantes en esta disciplina, se espera que los futuros

profesionales sean capaces de desarrollar software de mayor calidad, con menos errores y mejor rendimiento. Esto no solo beneficia a los desarrolladores individuales, sino que también contribuye a la mejora general de la industria del software, lo que puede tener un efecto positivo en la economía digital.

8.2.2. Objetivos de desarrollo sostenible (ODS) de las naciones unidas

Tras analizar el impacto socio-económico del proyecto, se emplearán los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas [48] como marco de referencia. Estos objetivos representan un llamamiento universal a la acción para poner fin a la pobreza, proteger el planeta y mejorar las vidas y las perspectivas de las personas en todo el mundo.

Dentro de este marco global, este proyecto se alinea con varios ODS, entre los que destacan:

ODS 4: Educación de calidad [49]

Este objetivo busca garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje para todos a lo largo de la vida. El proyecto contribuye directamente a este fin al centrarse en mejorar la enseñanza y el aprendizaje de la programación concurrente. Mediante una aproximación práctica y visual, se facilita la comprensión de conceptos complejos y se potencia la capacidad de los estudiantes para desarrollar software concurrente de manera más efectiva y robusta, lo cual es fundamental en el desarrollo tecnológico actual.

ODS 8: Trabajo decente y crecimiento económico [50]

Este objetivo promueve el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo y el trabajo decente para todos. La programación concurrente es un área de conocimiento y una habilidad técnica con una demanda creciente en el mercado laboral. El constante aumento en el número de núcleos de procesamiento en los dispositivos informáticos subraya la necesidad de profesionales capacitados en el desarrollo de aplicaciones que puedan explotar el paralelismo. Al fortalecer la formación en esta disciplina, el proyecto contribuye a preparar a los futuros profesionales para acceder a empleos de calidad en sectores tecnológicos clave y fomenta la productividad en la economía digital.

ODS 9: Industria, innovación e infraestructura [51]

Este objetivo persigue construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación. Las competencias en programación concurrente son esenciales para el desarrollo de sistemas de software de alto rendimiento y escalabilidad, necesarios para la construcción y operación de infraestructuras digitales

modernas (como la computación en la nube o los sistemas de gestión de grandes datos) y para la modernización de procesos industriales. Asimismo, son fundamentales para impulsar la innovación en campos emergentes como la inteligencia artificial, el análisis de big data y la computación científica. Por tanto, al mejorar la capacitación en esta área, el proyecto sienta las bases para el avance tecnológico y la innovación futura.

8.2.3. Plan de explotación

Este proyecto tiene como objetivo principal implantarse como una herramienta didáctica en el ámbito académico, específicamente en la enseñanza de la programación concurrente. Para lograrlo, se desarrollará un plan de explotación que contemple las siguientes acciones:

Colaboración con instituciones educativas

Se buscará establecer colaboraciones con universidades y centros de formación técnica para integrar la herramienta en sus programas de estudio. Esto incluirá la realización de talleres, seminarios la adaptación a las clases y trabajos.

Actualizaciones y mantenimiento continuo

Se establecerá un plan de mantenimiento y actualización de la herramienta para garantizar su funcionamiento óptimo y su adaptación a las necesidades cambiantes del entorno educativo. Esto incluirá la corrección de errores, la incorporación de nuevas funcionalidades y la mejora de la interfaz de usuario en función de los comentarios recibidos por parte de los usuarios.

Evaluación del impacto

Para medir el impacto de la herramienta en la enseñanza de la programación concurrente, se llevarán a cabo evaluaciones periódicas que incluyan encuestas a estudiantes y docentes, análisis de resultados académicos y estudios de caso. Estos datos permitirán ajustar el enfoque pedagógico y mejorar continuamente la herramienta.

Sostenibilidad financiera

Para garantizar la sostenibilidad financiera del proyecto, se explorarán diversas fuentes de financiación, como subvenciones gubernamentales y patrocinios de empresas tecnológicas. Aunque el foco estará en una distribución gratuita de la herramienta, se considerará la posibilidad de ofrecer servicios adicionales, como formación especializada, que puedan generar ingresos para el mantenimiento y desarrollo continuo del proyecto.

CAPÍTULO 9

CONCLUSIONES Y TRABAJO FUTURO

En este capitulo se prensentan las conclusiones del proyecto (Sección 9.1, *Conclusiones del Proyecto*) y las conclusiones personales (Sección 9.2, *Conclusiones Personales*). También se describirán posibles mejoras al proyecto y diferentes frentes en los que se podría avanzar en próximos proyectos.

9.1. Conclusiones del Proyecto

Este documento ha presentado el desarrollo de una herramienta didáctica para la programación concurrente. Para ello se han ido describiendo las diferentes etapas del proyecto, desde el análsis, pasando por el diseño y la implementación, hasta la validación del mismo.

El objetivo principal era crear una herramienta que permitiese a estudiantes interiorizar los conceptos característicos de la concurrencia de manera práctica. Recuperando los objetivos planteados en la Sección 1.2, *Objetivos* podemos observar que todos los objetivos, tanto los principales como los secundarios, han sido cumplidos.

- **Obj.1**: La herramienta compila, ejecuta y depura código concurrente en *C*.
  - **1.1**: La depuración permite al usuario controlar la ejecución de los hilos.
  - **1.2**: La depuración permite al usuario observar el estado de los hilos.
  - **1.3**: La herramienta permite realizar las acciones básicas de un depurador, tales como *step over*, *step into*, *step out*, *continue*, *breakpoint*, etc.
- **Obj.2**: La herramienta tiene un enfoque didáctico

- **2.1**: Se abstrae de la complejidades en la compilación, memoria de los procesos y llamadas al sistema, arquitectura de la máquina o sistema operativo.
- **2.2**: Tiene una interfaz gráfica que permite al usuario interactuar con la herramienta.
- **2.3**: Tiene las funcionalidadades básicas para permitir al usuario entender cómo se comportan los hilos.

Los principales problemas que se han encontrado durante el desarrollo del proyecto han sido encontrar las diferentes herramientas y librerías que permitieran no tener que implementar todo el código desde cero, finalmente decantandose por el uso de un depurador externo como es *gdb* y *RR* y la librería *pygdbMI*, y encontrar una solución que cumpla los requisitos de utilizar un entorno *sandbox* (Requisito RS-NF-09) y sea capaz de alojar varios clientes.

9.2. Conclusiones Personales

Este proyecto ha sido una experiencia muy enriquecedora, ya que me ha permitido aprender a utilizar herramientas y librerías que no conocía, así como mejorar mis conocimientos sobre depuración y programación concurrente. También he aprendido a trabajar con *Docker* y a utilizarlo como herramienta de desarrollo, lo que me ha permitido entender mejor cómo funcionan los contenedores y cómo se pueden utilizar para crear entornos de desarrollo aislados.

He adquirido muchos conocimientos sobre sistemas distribuidos, sobre todo enfocado en servicios web, debido al uso de API REST y WebSockets. Adicionalmente, ha sido necesario aprender a implementar un servicio de monitoreo de disponibilidad basado en latidos (*heartbeat*) y a utilizar herramientas de orquestación como *Docker Compose*.

El hecho de enfrentarme a un proyecto de estas dimensiones sin tener un conocimiento previo y unos objetivos claros ha sido un reto, pero también una oportunidad para mejorar mis capacidades de investigación, análisis y diseño, para poder tomar las mejores decisiones. Características como la escalabilidad, la reutilización y la modularidad han sido puntos clave que se han tenido que tener en cuenta para afrontar un proyecto de esta magnitud.

9.3. Trabajo Futuro

Aunque durante el desarrollo del proyecto se han cumplido con todos los objetivos planteados y se ha tratado de abarcar todos los frentes posibles, siempre hay margen de mejora y nuevas funcionalidades que se pueden implementar. A continuación se presentan los diferentes frentes que se podrían explorar en el futuro y de qué manera se podrían

implementar:

- **Mejorar la calidad de la intefaz gráfica**
  - Tener un diseño *responsive* que se adapte a diferentes tamaños de pantalla.
  - Buscar soluciones para mejorar la accesibilidad de la herramienta.
  - Mejorar la experiencia de usuario, añádiendo *drag and drop* para seleccionar los hilos a visualizar.
- **Aumentar las capacidades distribuidas**
  - Tener un sistema DNS que permita acceder a los diferentes servicios de forma más sencilla.
  - Utilizar TLS para cifrar las comunicaciones entre el cliente y las sesiones de depuración.
  - Implementar un sistema de monitoreo y alertas para detectar problemas en el sistema.
  - Implementar un sistema para que el proxy pueda reiniciar los contenedores que se caen.
- **Nuevas funcionalidades para la depuración**
  - Añadir una opción para añadir un planificador de hilos, que permita simular diferentes políticas de planificación.
  - Tener soporte para un proyecto completo, en vez de un único archivo.
  - Añadir la opción de depurar código ensamblador.

BIBLIOGRAFÍA

[1] J. L. Hennessy y D. A. Patterson, «Chapter 3: Instruction-Level Parallelism and Its Exploitation,» en *Computer Architecture: A Quantitative Approach*, 5th. Morgan Kaufmann, 2011, cap. 3.

[2] S. Holmbacka, E. Nogues, M. Pelcat, S. Lafond y J. Lilius, «Energy efficiency and performance management of parallel dataflow applications,» pp. 1-8, 2014. DOI: 10.1109/DASIP.2014.7115624.

[3] G. E. Moore, «Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.,» *IEEE Solid-State Circuits Society Newsletter*, vol. 11, n.º 3, pp. 33-35, 2006. DOI: 10.1109/N-SSC.2006.4785860.

[4] T. Rauber y G. Rünger, «1. Introduction,» en *Parallel programming*. Springer, 2013, cap. 1.

[5] Wikipedia contributors. «Debugger,» Acceso: 9 de mar. de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Debugger&oldid=1278914632.

[6] Microsoft. «First look at the debugger - Visual Studio,» Acceso: 9 de mar. de 2025. [En línea]. Disponible en: https://learn.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2022.

[7] Eclipse Foundation. «Help - Eclipse Platform,» Acceso: 9 de mar. de 2025. [En línea]. Disponible en: https://help.eclipse.org/2024-12/index.jsp.

[8] GDB developers. «GDB Documentation,» Acceso: 9 de mar. de 2025. [En línea]. Disponible en: https://sourceware.org/gdb/.

[9] LLDB developers. «LLDB,» Acceso: 9 de mar. de 2025. [En línea]. Disponible en: https://lldb.llvm.org/.

[10] Bell Labs. «The Creation of the UNIX* Operating System,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://web.archive.org/web/20140328184730/http://www.bell-labs.com/history/unix/.

[11] GDB developers. «GDB Download,» Acceso: 11 de mar. de 2025. [En línea]. Disponible en: https://sourceware.org/gdb/download/.

[12] GDB developers. «GDB Python API (Debugging with GDB),» Acceso: 11 de mar. de 2025. [En línea]. Disponible en: https://sourceware.org/gdb/current/onlinedocs/gdb.html/Python-API.html#Python-API.

[13] GDB developers. «GDB/MI (Debugging with GDB),» Acceso: 11 de mar. de 2025. [En línea]. Disponible en: https://sourceware.org/gdb/current/onlinedocs/gdb.html/GDB_002fMI.html.

[14] GDB developers. «Reverse Execution (Debugging with GDB),» Acceso: 11 de mar. de 2025. [En línea]. Disponible en: https://sourceware.org/gdb/current/onlinedocs/gdb.html/Reverse-Execution.html.

[15] B. Lee, M. Hirzel, R. Grimm y K. S. McKinley, «Debug all your code: portable mixed-environment debugging,» *SIGPLAN Not.*, vol. 44, n.º 10, pp. 207-226, oct. de 2009. DOI: 10.1145/1639949.1640105. [En línea]. Disponible en: https://doi.org/10.1145/1639949.1640105.

[16] LLDB developers. «LLDB Python API - LLDB,» Acceso: 12 de mar. de 2025. [En línea]. Disponible en: https://lldb.llvm.org/python_api.html.

[17] Mozilla. «Home ů rr-debugger/rr Wiki,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://github.com/rr-debugger/rr/wiki.

[18] Mozilla. «rr: lightweight recording & deterministic debugging,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://rr-project.org/.

[19] JetBrains. «Buy Clion: Pricing and Licesing,» Acceso: 10 de mar. de 2025. [En línea]. Disponible en: https://www.jetbrains.com/clion/buy/?section=commercial&billing=yearly.

[20] JetBrains. «Download CLion: A Smart Cross-Platform IDE for C/C++,» Acceso: 10 de mar. de 2025. [En línea]. Disponible en: https://www.jetbrains.com/clion/download/#section=windows.

[21] JetBrains. «Debug | CLion Documentation,» Acceso: 10 de mar. de 2025. [En línea]. Disponible en: https://www.jetbrains.com/help/clion/debugging-code.html#starting_debug_session.

[22] JetBrains. «Debug Tool Window | CLion Documentation,» Acceso: 10 de mar. de 2025. [En línea]. Disponible en: https://www.jetbrains.com/help/clion/debug-tool-window.html.

[23] IEEE and The Open Group. «What is POSIX,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://servicenow.iu.edu/kb?id=kb_article_view&sysparm_article=KB0024334.

[24] Valgrind developers. «Valgrind,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://valgrind.org/docs/manual/hg-manual.html.

[25] Universidad Carlos III de Madrid. «17.2 Helgrind tool,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://www.it.uc3m.es/pbasanta/asng/course_notes/helgrind_tool_en.html.

[26] LLVM. «ThreadSanitizer - Clang,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://clang.llvm.org/docs/ThreadSanitizer.html.

[27] Google. «ThreadSanitizerCppManual - Google/sanitizers Wiki,» Acceso: 13 de mar. de 2025. [En línea]. Disponible en: https://github.com/google/sanitizers/wiki/threadsanitizercppmanual.

[28] «IEEE Guide for Software Requirements Specifications,» *IEEE Std 830-1984*, pp. 1-26, 1984. DOI: 10.1109/IEEESTD.1984.119205.

[29] International Council on Systems Engineering (INCOSE), «INCOSE Guide to Writing Requirements (Version 4) - Summary Sheet,» INCOSE, inf. téc. INCOSE-TP-2010-006-04, jun. de 2023, Requirements Working Group. [En línea]. Disponible en: https://www.incose.org/docs/default-source/working-groups/requirements-wg/guidetowritingrequirements/incose_rwg_gtwr_v4_summary_sheet.pdf.

[30] S. Cook et al., «Unified Modeling Language Specification,» Object Management Group, Standard, ver. 2.5.1, dic. de 2017. [En línea]. Disponible en: https://www.omg.org/spec/UML/2.5.1.

[31] Wikipedia contributors. «Caso de Uso - Wikipedia, la enciclopedia libre,» Acceso: 21 de mar. de 2025. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Caso_de_uso#cite_ref-2.

[32] Chapman y Hall/CRC, *Distributed Systems*, 2nd. Chapman y Hall/CRC, 2014.

[33] Parcel. «parcel-bundler/parcel: The zero configuration build tool for the web,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/parcel-bundler/parcel.

[34] CodeMirror, *codemirror/dev: Development repository for the CodeMirror editor proyect*, 2024. Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/codemirror/dev.

[35] Socket.IO. «socketio/socket.io: Realtime application framework (Node.JS server),» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/socketio/socket.io.

[36] Flask. «pallets/flask: The Python micro framework for building web applications,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/pallets/flask/tree/main.

[37] Pygdbmi. «pygdbmi/pygdbmi: Python GDB MI interface,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/cs01/pygdbmi?tab=MIT-1-ov-file#readme.

[38] «IEEE Standard for System and Software Verification and Validation,» *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)*, pp. 1-223, 2012. DOI: 10.1109/IEEESTD.2012.6204026. Acceso: 20 de abr. de 2025.

[39] H. D. Benington, «Production of Large Computer Programs,» *Annals of the History of Computing*, vol. 5, n.º 4, pp. 350-361, 1983. DOI: 10.1109/MAHC.1983.10102.

[40] P. S. Foundation. «PEP 8 – Style Guide for Python Code,» Acceso: 29 de mayo de 2025. [En línea]. Disponible en: https://peps.python.org/pep-0008/.

[41] E. International. «ECMAScript Language Specification,» Acceso: 29 de mayo de 2025. [En línea]. Disponible en: https://tc39.es/ecma262/.

[42] moby. «The Moby Project - a collaborative project for the container ecosystem to assemble container-based systems,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/moby/moby.

[43] RR. «rr-debugger/rr: Record and replay framework for debugging,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://github.com/rr-debugger/rr.

[44] MIT. «MIT License,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://opensource.org/licenses/MIT.

[45] E. Union. «General Data Protection Regulation (GDPR),» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://gdpr-infto.eu/.

[46] E. Union. «Art. 6 GDPR - Lawfulness of processing - General Data Protection Regulation (GDPR),» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://gdpr-info.eu/art-6-gdpr/.

[47] E. Union. «Art. 5 GDPR - Principles relating to processing of personal data - General Data Protection Regulation (GDPR),» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://gdpr-info.eu/art-5-gdpr/.

[48] ONU. «Desarrollo Sostenible,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/.

[49] ONU. «ODS 4: Educación de calidad,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/education/.

[50] ONU. «ODS 8: Trabajo decente y crecimiento económico,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/economic-growth/.

[51] ONU. «ODS 9: Industria, innovación e infraestructura,» Acceso: 1 de mayo de 2025. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/infrastructure-industrialization/.

GLOSARIO

A

API

Conjunto de funciones y procedimientos que permiten la comunicación entre distintos componentes de software. 7

aplicación distribuida

Aplicación que se ejecuta en múltiples procesos y permite la ejecución de múltiples tareas de forma simultánea. 50

aplicación monolítica

Aplicación que se ejecuta como un único proceso y no permite la ejecución de múltiples tareas de forma simultánea. 49

aplicación web

Aplicación que se ejecuta en un servidor y se accede a través de un navegador web. 50

B

barrera

Mecanismo de sincronización que permite a un hilo esperar a que otros hilos alcancen un punto de sincronización antes de continuar su ejecución. 11

breakpoint

Puntos de interrupción que se colocan con el objetivo de parar el programa en el punto indicado plural. 30, 34, 69, 73

C

cerrojo

Mecanismo de sincronización que permite controlar el acceso a recursos compartidos entre múltiples procesos. 11

CLI

Tipo de interfaz basada en texto que permite la interacción con un programa a través de comandos escritos en la consola. 6

compilación

Proceso de traducción de un programa escrito en un lenguaje de programación a código máquina. XIII, 13, 69, 71, 72

compilador

Programa que traduce un programa escrito en un lenguaje de programación a código máquina. 31

concurrencia

Propiedad de un sistema que permite la ejecución de múltiples tareas de forma simultánea. 5, 6, 11, 22, 32, *véase* proceso

condición de carrera

Situación en la que el resultado de un programa depende del orden de ejecución de sus instrucciones. 6, 11–13

contenedor

Entorno aislado que permite ejecutar aplicaciones de forma independiente del sistema operativo subyacente. 52–54, 61, 70–73, 75, 76, 78–80, 82

CPU

Unidad de procesamiento central, componente de un ordenador que interpreta y ejecuta instrucciones. 6

cuello de botella

Situación en la que el rendimiento de un sistema se ve limitado por un componente específico, lo que impide alcanzar su máximo potencial. 52

código abierto

Software cuyo código fuente es accesible y modificable por cualquier usuario. 6, 8–10

código fuente

Conjunto de instrucciones escritas en un lenguaje de programación que define el comportamiento de un programa. 50, 76

D

depurador

Programa que tiene como objetivo probar y depurar otros programas, a través del control de la ejecución del programa y la inspección del estado del mismo. 6–12, 17, 50, 51, *véase* proceso

depurar

Proceso de identificar y corregir errores en un programa. 5–7, 9, 21, 22, 25, 38, 45, 69, *véase* depurador

desensamblador

Programa que traduce código máquina a código ensamblador. 8

determinista

Propiedad de un sistema que garantiza un resultado predecible. 9, *véase*

E

ensamblador

Lenguaje de programación de bajo nivel que representa instrucciones en código máquina. 7

F

falso positivo

Resultado de una prueba que indica que un error existe cuando en realidad no es así. 13

G

GUI

Tipo de interfaz que permite la interacción con un programa a través de elementos visuales, como botones, menús y ventanas. 14

H

hilo

Unidad de ejecución más pequeña que un proceso, que comparte recursos con otros hilos de un mismo proceso. 6, 11, 17, 23, 24, 26, 35–37, 45, 50, 51, 69, 72, 73, 80

HTTP

Protocolo de transferencia de hipertexto, utilizado para la comunicación entre un cliente y un servidor web. 53

I

IDE

Entorno de desarrollo integrado, un software que combina herramientas para facilitar la programación, como un editor de texto, un compilador, un depurador, entre otros. 9

imagen Docker

Plantilla que se utiliza para crear contenedores Docker, incluyendo el código y las configuraciones necesarias para ejecutar una aplicación. 53, 75, 76

indeterminista

Propiedad de un sistema que no garantiza un resultado predecible. 9, *véase* determinista

interbloqueo

Situación en la que dos o más procesos quedan bloqueados esperando a que se liberen recursos que se están utilizando. 6, 11

interfaz gráfica

Tipo de interfaz que permite la interacción con un programa a través de elementos visuales, como botones, menús y ventanas. 2, 10, 11

interfaz máquina-humano

Conjunto de elementos que permiten la interacción entre un ser humano y una máquina. 7

inyección de código

Técnica de programación que consiste en insertar código en un programa en tiempo de ejecución. 12

inyección de código

Técnica de programación que consiste en insertar código en un programa en tiempo de ejecución. 50

J

JSON

Notación de objetos de JavaScript, un formato ligero de intercambio de datos que es fácil de leer y escribir para los humanos, y fácil de analizar y generar para las máquinas. 73

L

lenguaje máquina

Lenguaje de programación que representa instrucciones en código binario. 7

llamada al sistema

Interacción entre un programa y el sistema operativo para solicitar servicios del mismo. 50

M

multiplataforma

Software que puede ejecutarse en diferentes sistemas operativos y arquitecturas de hardware. 27, 41

O

overhead

Tiempo adicional que se necesita para realizar una tarea, que no contribuye directamente al resultado final. 9

P

paralelismo

Técnica de programación que consiste en ejecutar múltiples tareas de forma simultánea para mejorar el rendimiento de un programa. 5

parser

Programa que analiza el código fuente de un programa para identificar su estructura y sintaxis. 50, 51

planificador

Software responsable de gestionar y asignar recursos a los procesos en un sistema operativo para optimizar su ejecución. 5, *véase* proceso

proceso

Entidad que representa una tarea en ejecución en un sistema operativo. 5–7, 9, 17, 50, 53, 54

programa concurrente

Programa que utiliza múltiples procesos en ejecución ,que pueden ser ejecutados en paralelo o de forma intercalada, para realizar tareas de forma simultánea. 5, 6, 9, 11, 15, 17, 20, *véase* proceso

proxy

Servidor que actúa como intermediario entre un cliente y otro servidor, permitiendo la comunicación entre ambos. 52–54, 60, 66, 70, 75, 76, 78, 79, 82

R

registros

Memoria de alta velocidad que almacena datos y direcciones de memoria temporales para mejorar el rendimiento de un procesador. 7, 10

REST API

Interfaz de programación de aplicaciones que permite la comunicación entre diferentes sistemas a través de peticiones HTTP. 54

S

salida estándar

Canal de comunicación que permite la salida de datos de un programa. 12, 13

script

Programas que automatizan tareas repetitivas. 7, 8

sentencia

Instrucción que se ejecuta en un programa. 50

servicio web

Aplicación que permite la comunicación entre diferentes sistemas a través de la web. 50

sistemas multicore

Sistemas que contienen múltiples núcleos de procesamiento en un solo chip, lo que permite ejecutar múltiples tareas de forma simultánea. 5

software

Conjunto de programas y datos que permiten el funcionamiento de un sistema informático. 6, 17

step into

Comando de depuración que permite avanzar una instrucción en el código fuente y entrar en las funciones que se llaman. 10, *véase* depurar

step over

Comando de depuración que permite avanzar una instrucción en el código fuente sin entrar en las funciones que se llaman. 10, *véase* depurar

T

tiempo de compilación

Período durante el cual un programa se traduce a código máquina. 5, 12, *véase* compilación

tiempo de ejecución

Período durante el cual un programa se ejecuta en un sistema. 12, 13

TUI

Tipo de interfaz basada en texto que permite la interacción con un programa a través de elementos visuales, como botones, menús y ventanas. 6

U

Unix

Familia de sistemas operativos multiusuario y multitarea, desarrollados en los años 70. 50

V

variable de condición

Mecanismo de sincronización que permite a un hilo esperar a que se cumpla una condición antes de continuar su ejecución. 11

variables de entorno

Conjunto de variables que almacenan información sobre el entorno en el que se ejecuta un programa, como la configuración del sistema o las rutas de búsqueda de archivos. 80

SIGLAS

A

API

Application Programming Interface. 7, 8, 51, *Glossary:* API

C

CLI

Command Line Interface. 6, 8–10, 14, *Glossary:* CLI

CPU

Central Processing Unit. 6, 7, 10, 14, *Glossary:* CPU

G

GUI

Graphical User Interface. 14, 29, *Glossary:* GUI

H

HTTP

Hypertext Transfer Protocol. 53, *Glossary:* HTTP

I

IDE

Integrated Development Environment. 9, *Glossary:* IDE

J

JSON

JavaScript Object Notation. 73, *Glossary:* JSON

T

TUI

Text User Interface. 6, *Glossary:* TUI

APÉNDICE A

MANUAL DE USUARIO

Este apéndice contiene el manual de usuario para la herramienta de depuración de programas concurrentes desarrollada como parte de este Trabajo de Fin de Grado.

La *Herramienta Didáctica para la Programación Concurrente* es una aplicación web que facilita el aprendizaje y comprensión de los conceptos de concurrencia, permitiendo a estudiantes y profesores visualizar el comportamiento de programas multihilo en tiempo real.

La herramienta permite ejecutar y depurar código C con múltiples hilos, observando visualmente el comportamiento de cada hilo y el estado de las variables durante la ejecución, lo que facilita la comprensión de conceptos complejos relacionados con la programación concurrente.

A.1. Requisitos del sistema

Para ejecutar esta herramienta, es necesario contar con:

- **Docker** y **Docker Compose**: Para gestionar los contenedores aislados
- **Node.js** (versión 14 o superior): Para la construcción del frontend

A.1.1. Instalación de requisitos previos

Antes de poder utilizar la herramienta, es necesario instalar los siguientes componentes en su sistema operativo:

Instalación de Docker

Docker es esencial para el funcionamiento de la herramienta, ya que proporciona los entornos aislados donde se ejecutará el código.

- **Linux (Ubuntu/Debian)**:

```
sudo apt update
sudo apt install docker.io docker-compose
sudo systemctl enable --now docker
sudo usermod -aG docker $USER
```

- **macOS**: Descargar e instalar Docker Desktop desde: https://www.docker.com/products/docker-desktop/

Instalación de Node.js

Node.js es necesario para construir y ejecutar el frontend de la aplicación.

- **Linux (Ubuntu/Debian)**:

```
sudo apt install -y nodejs
```

- **macOS** (usando Homebrew):

```
brew install node
```

A.2. Instalación y ejecución

A.2.1. Obtención del código fuente

Para comenzar, se debe clonar el repositorio de GitHub:

```
git clone https://github.com/Adri-Extremix/Trabajo-de-Fin-de-Grado
```

A.2.2. Preparación de scripts

Es necesario dar permisos de ejecución a los scripts del sistema:

```
cd Trabajo-de-Fin-de-Grado
chmod +x start_proyect.sh
chmod +x close_proyect.sh
```

A.2.3. Inicio de la aplicación

Para iniciar todos los componentes del sistema:

```
./start_proyect.sh
```

Este script realizará las siguientes acciones:

- Detectar la dirección IP local
- Crear una red Docker para la comunicación entre contenedores
- Construir el frontend
- Iniciar el proxy
- Lanzar los contenedores de depuración
- Abrir el navegador con la URL del sistema

La aplicación estará accesible automáticamente en el navegador, o alternativamente en la dirección `http://<IP_LOCAL>:8080`.

A.2.4. Cierre de la aplicación

Para detener todos los servicios cuando se termine de utilizar la herramienta:

```
./close_proyect.sh
```

A.3. Guía de uso

A.3.1. Características principales

La herramienta incluye las siguientes funcionalidades:

- **Ejecución de código C**: Compila y ejecuta programas C con múltiples hilos

- **Depuración interactiva**: Utiliza comandos de depuración como step-in, step-over, y breakpoints
- **Visualización en tiempo real**: Observa el estado y la evolución de los hilos durante la ejecución
- **Entorno seguro**: El código se ejecuta en contenedores Docker aislados
- **Arquitectura escalable**: Múltiples contenedores Docker pueden servir a diferentes usuarios
- **Comunicación WebSockets**: Actualización en tiempo real del estado de la depuración

A.3.2. Interfaz principal

La interfaz de la aplicación está dividida en dos secciones:

- **Sección de código**: Donde se escribe y edita el código C
  - **Editor de código**: Donde se escribe el programa C
  - **Panel de control**: Con botones para compilar y ejecutar
  - **Terminal de salida**: Muestra la salida del programa y mensajes del sistema
- **Sección de depuración**: Donde se gestionan los puntos de interrupción y se observa el estado de la ejecución
  - **Panel de depuración**: Contiene botones para iniciar, detener y controlar la depuración
  - **Vista de variables**: Muestra las variables activas y sus valores
  - **Vista de hilos**: Visualiza el estado de los diferentes hilos

A.3.3. Flujo de trabajo básico

1. Escribir o cargar código C en el editor de la interfaz web
2. Añadir puntos de interrupción (breakpoints) haciendo clic junto al número de línea
3. Compilar el código con el botón *Compilar*
4. Si hay errores de compilación, corregirlos y volver a compilar
5. Para ejecución normal, presionar *Ejecutar*
6. Para depuración, cambiar a la sección de depuración

7. Iniciar la depuración con el botón *Run*
8. Utilizar los controles de depuración:
    - **Step Over**: Avanza una instrucción sin entrar en funciones
    - **Reverse Step Over**: Retrocede una instrucción sin entrar en funciones
    - **Step Into**: Avanza entrando en las funciones llamadas
    - **Reverse Step Into**: Retrocede entrando en las funciones llamadas
    - **Step Out**: Sale de la función actual
    - **Reverse Step Out**: Retrocede saliendo de la función actual
    - **Continue**: Ejecuta hasta el siguiente breakpoint
    - **Reverse Continue**: Retrocede hasta el siguiente breakpoint
    - **Stop**: Detiene la ejecución del programa
9. Observar el estado de variables y la evolución de los hilos en tiempo real
10. La depuración puede detenerse en cualquier momento con *Stop*

Importante

Cada vez que se realicen cambios en el código o en los puntos de interrupción, es necesario recompilar el código para que los cambios surtan efecto.