ScienceDirect
Disponible en www.sciencedirect.com
Revista Iberoamericana de Automática e Informática industrial 10 (2013) 465–476
www.elsevier.es/RIAI

Herramienta Web Ligera para La Programación en C-Concurrente

Pablo Basanta-Val*, Marisol García-Valls y Pablo López-Anastasio

Departamento Ingeniería Telemática, Universidad Carlos III de Madrid, Edificio Torres Quevedo, Avda. de la Universidad, 30, 28911 Leganés, Madrid, España.

Resumen

El uso de herramientas a la hora de enseñar una determinada disciplina aporta múltiples beneficios desde el punto de vista de la actividad docente pues permite enfatizar o ilustrar determinados cuestiones que a veces resultan difíciles de enfatizar sin tal apoyo. Ese es también el caso de las herramientas que permiten detectar si ha habido algún tipo de problema en un programa escrito en C-concurrente. Dichas herramientas ofrecen interfaces que pueden complementar la información dada por un compilador con información adicional sobre diferentes tipos de condiciones de carrera o fugas de memoria que aparecen en el código. El presente trabajo tiene por objetivo ver cómo se ha integrado un núcleo de validación para C ya existente como aplicación web, lo que le permite estar accesible a través de la red. Dicha herramienta ha sido evaluada en un curso de programación ya existente, donde ha mostrado que es capaz aportar información adicional de utilidad para el discente y el docente. También se han realizado una serie de mediciones de rendimiento para establecer los límites operativos de la herramienta diseñada dentro de los límites de una asignatura donde se enseña C concurrente.

Palabras Clave:
Herramientas, Informática Industrial, Sistemas Concurrentes, Educación, C.

1. Introducción

El uso de herramientas que sirven de apoyo a la impartición de una determinada materia cuenta con una larga tradición (Caspi et al. 2005)(Sifakis 2011)(Bouyssounouse, Sifakis 2005)(Rodriguez-Andina, Gomes 2013)(Estevez-Avres, Basanta-Val P. & García-Valls 2004) y una orientación actual hacia entornos donde el acceso web está ganando importancia (Basanta-Val .P et al. 2012)(Garcia-Valls, Basanta-Val 2012)(Santana et al. 2013). Dependiendo del dominio de aplicación, existen plataformas generales con acceso web mayoritario y que permiten el aprendizaje con cierta componente lúdica (Merino et al. 2012)(Kim, Jeon 2009)(Pinto, Moreira & Matos 2012)(Alonso, Pastor & Álvarez 2004)(Hamblen, Bekkum 2013)(Crenshaw 2013) De forma más específica, también existen otras plataformas que están más orientadas hacia un dominio en particular como la programación en lenguaje-C (Pardo, Kloos 2011), el middleware de distribución en su conjunto (Garcia-Valls, Basanta-Val 2012) y las aplicaciones de control y automática industrial (Salido, Lillo, A. Déniz Suárez, O., Bueno, G. 2011)(Weber, Rehkopf 2009) con interfaz web, lo que les permite atender a los aspectos particulares referentes a la docencia que se pretende impartir. Es en este último tipo, en la provisión de herramientas para cierto entorno educativo específico, donde se sitúa la contribución de este artículo.

* Autor en correspondencia.
Correos electrónicos: pbasanta@it.uc3m.es (Pablo Basanta Val), mvalls@it.uc3m.es (Marisol García Valls) y pk.pablo@gmail.com (Pablo López Anastasio)

Más específicamente, la herramienta diseñada es para sistemas empotrados programados en lenguaje de programación C concurrente mediante POSIX. En este entorno, la existencia de herramientas que permitan detectar errores de programación puede ayudar no sólo al diseño y desarrollo de software empotrado (Monzón, Fernández & de la Puente 2012)(Caspi et al. 2005) sino también al soporte a la formación específica tal y como muestran (Basanta-Val .P et al. 2012) o (Ihantola 2006).

Algunas de estas herramientas (Havelund, Pressburger 2000) (Nethercote, Seward 2007) pueden ofrecer información sobre la existencia o no de *condiciones de carrera* o de *interbloqueo*, o fugas de memoria, en un cierto código de forma automática. Dicha información puede ser usada en el entorno docente para dar información adicional sobre la causa o el origen de dicho error y que a su vez sería utilizado (Sierra et al. 2012, Lee, Kester & Schulzrinne 2011) para mejorar la enseñanza de la concurrencia en C.

Internamente, estas herramientas pueden estar soportadas con métodos formales que validen exhaustivamente un trozo de código, e.g. Pathfinder (Havelund, Pressburger 2000), y/o con una validación de pasada-única que comprueba sólo los caminos que están ejecutándose, e.g. Valgrind (Nethercote, Seward 2007), que aplique comprobaciones. Típicamente, los métodos formales son exhaustivos y requieren un gran consumo de CPU, incluso cuando se ejecutan sobre ejercicios relativamente sencillos (Basanta-Val .P et al. 2012). Las herramientas que comprueban un único camino, en cambio, son opciones computacionalmente más económicas (con menor sobrecarga) pues no exploran la totalidad de estados de la aplicación, lo cual puede no ser necesario en un ejercicio de programación en C-concurrente.

http://dx.doi.org/10.1016/j.riai.2013.05.010

El presente artículo explora una herramienta del segundo tipo llamada Valgrind en un contexto educativo con acceso web. Esta se utiliza en cursos de programación en C para la detección dinámica de fugas de memoria. Pero también puede ser usada para detectar problemas de concurrencia como son el *abrazo mortal* o *la condición de carrera* pues implementan algoritmos para ello (Savage et al. 1997). En dicho contexto, este artículo realiza las siguientes contribuciones principales:

- Propone una interfaz web que controla el acceso hecho sobre la plataforma. Dicha interfaz permite controlar de forma anticipada problemas en un envío en el cliente.
- Evalúa el tándem resultante de la integración entre la interfaz web y la herramienta en un curso de programación C-POSIX (Committee 2003) preexistente. Esto permite evaluar el beneficio en término de información aportada sobre errores como el coste temporal de la plataforma.

Con el fin de presentar los resultados obtenidos, el resto del este artículo cubre tanto aspectos relativos a la implementación de la plataforma como al rendimiento en una sesión de ejercicios de detección de errores en código C concurrente. La Sección 2 presenta la plataforma de forma técnica y un caso de uso. La Sección 3 añade información evalúa el beneficio y el coste de la plataforma. La Sección 4 conecta el presente trabajo con otras iniciativas similares a la propuesta. Por último, la Sección 5 remarca conclusiones y describe el trabajo en curso más relacionado con la integración realizada.

2. Acceso Web Implementado Para Java EE

Esta sección contiene una descripción detallada del software implementado. Describe el proceso de requisitos del software desarrollado, diagramas de flujo que describen en proceso en alto nivel y detalles de alto nivel relativos a la implementación (que comprenden aspectos arquitectónicos de alto nivel como estructurales). Se completa la sección con un ejemplo de uso sencillo pero significativo.

2.1. Especificación de Requisitos

La herramienta-web de validación diseñada tiene seis requisitos principales. Dichos requisitos han sido obtenidos mediante el análisis de plataforma simulares, principalmente tras analizar el rendimiento de validadores formales en (Basanta-Val .P et al. 2012):

- Requisito 1. La aplicación ha de ser accesible a través de la Web sencilla e intuitiva.
- Requisito 2. Ha de soportar código fuente en el lenguaje de programación C.
- Requisito 3. Ha de soportar múltiples ficheros comprimidos validados (parcialmente) en el lado cliente.
- Requisito 4. Devolver los resultados de forma asíncrona, mediante correo electrónico.
- Requisito 5. Ser capaz de ofrecer soporte general para la herramienta Valgrind y soporte particular para el módulo Helgrind, encargado del control de la concurrencia.
- Requisito 6: Ha de aportar información que pueda permitir mejorar la programación concurrente-C en un curso tipo.
- Requisito 7: Ser capaz de ofrecer un rendimiento eficaz en una clase tipo de 1,5 horas con 20 alumnos.

A la hora de traducir estos requisitos a casos de uso de alto nivel, se opta por un esquema con un caso único de validación, siguiendo un razonamiento similar al utilizado en (Basanta-Val .P et al. 2012). La idea es la de facilitar la funcionalidad al máximo reduciendo el número de actores del sistema a un único actor. En el modelo desarrollado en (Basanta-Val .P et al. 2012) la existencia de dos actores uno usuario y otro administrador se solapa con los mecanismos de gestión ofrecidos por la propia plataforma Java EE. Es por este motivo, que el administrador se ha quitado de la infraestructura de ejecución.

Así, hay un único actor (Figura 1) con un único caso de uso general que permite validar un código escrito en C. Ese caso de uso consta de los siguientes grandes bloques: *extracción*, *compilación*, *validación* y *notificación*. Extracción se encarga de manipular los ficheros comprimidos y de acceder a su contenido. Compilación se encarga de compilar dicho código mediante un compilador de C. Por otro lado, validación se encarga de validar (mediante el micro-núcleo de validación) el código generado con una herramienta general de validación. Por último, está el caso de notificación, el cual comprende las notificaciones mediante el uso de correo electrónico del resultado de la validación al usuario.

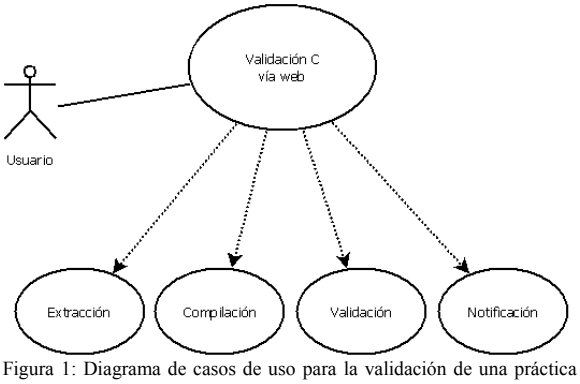

Figura 1: Diagrama de casos de uso para la validación de una práctica en C mediante la infraestructura propuesta

2.2. Diseño de Alto Nivel

A alto nivel, las entidades participantes en el proceso de verificación son las siguientes (Figura 2):

- *Acceso Web:*

Este módulo tiene dos subcomponentes. El primero es el encargado de proporcionar al usuario la posibilidad de introducir la información necesaria de forma visual. Dicha información incluye los datos identificativos del usuario, el código fuente de su programa comprimido en un fichero y las opciones relacionadas con la compilación y el modo de ejecución de la herramienta de validación (Valgrind en este caso).

El segundo subcomponente se encarga de comprobar que la información sea coherente y de almacenarla en un directorio específico. Después, envía la petición al gestor de peticiones.

- *Gestor de peticiones*

Este módulo recibe las peticiones de los usuarios y las encola hasta que el servidor puede procesarlas. De esta manera se evita que, si el servidor está saturado, el usuario reciba una negativa de servicio. Cuando el servidor puede procesar otra petición, el gestor de peticiones envía los datos de la petición correspondiente al módulo extractor.

- *Extractor de archivos*

Este módulo se encarga de extraer el código fuente comprimido en el fichero que el usuario ha enviado. Dicho código es almacenado en un directorio específico.

- *Compilador de archivos*

Este módulo compila el código fuente del usuario con las opciones que éste haya incluido. Si la compilación es exitosa, se guarda el ejecutable en un directorio específico. Si no, se informará al usuario del error para facilitarle la corrección.

- *Ejecutor de código*

Este módulo ejecuta la herramienta de la suite Valgrind que el usuario haya elegido sobre el código fuente ya compilado. El resultado será enviado al usuario

Se impondrá un límite de tiempo para que programas con bucles infinitos no saturen el servidor. Si el programa en cuestión supera dicho límite, se informa al usuario para que actúe consecuentemente.

- *Gestor de correos*

Este módulo se encarga de enviar el correo electrónico con la información pertinente. Podrá incluir el resultado de la validación o el informe de error de alguno de estos módulos: extracción, compilación o ejecución.

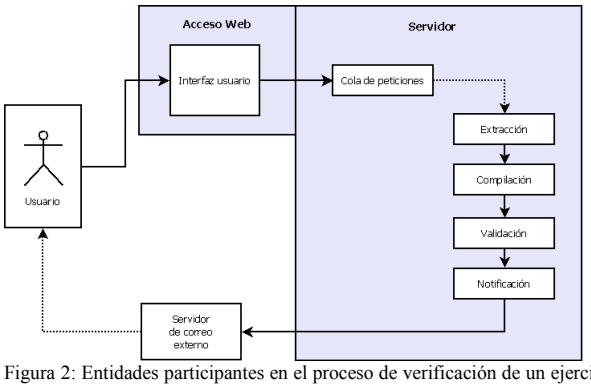

Figura 2: Entidades participantes en el proceso de verificación de un ejercicio. Esquema de alto nivel.

Hay comunicación entre los diferentes elementos del esquema de validación. Así, el acceso web envía datos comprimidos junto a opciones de compilación y ejecución. Esos mismos datos son enviados al gestor de peticiones que realiza un control de los recursos disponibles. Dichos datos y opciones se envían al módulo de extracción de archivos, el cual descomprime los datos. Tras ello, se envían los datos descomprimidos al módulo de compilación, el cual se encarga de generar un fichero ejecutable. Dicho fichero ejecutable es enviado al validador junto a sus opciones de ejecución. El validador produce como salida el resultado de la validación, información que es enviada de vuelta al cliente.

Por último en el diseño de alto nivel, está la arquitectura de despliegue utilizada. El software diseñado está pensado para que haya una relación entre tres nodos físicos: el primero es el equipo usuario; el segundo es el servidor de la plataforma y el tercero es el servidor externo. El primero sirve de interfaz con el usuario que accede con su navegador. El segundo hospeda la interfaz de usuario y el mecanismo de gestión web. Por último, hay un tercer nodo externo que se encarga del envío de los correos y que puede residir en un servidor diferente al servidor dónde reside la plataforma.

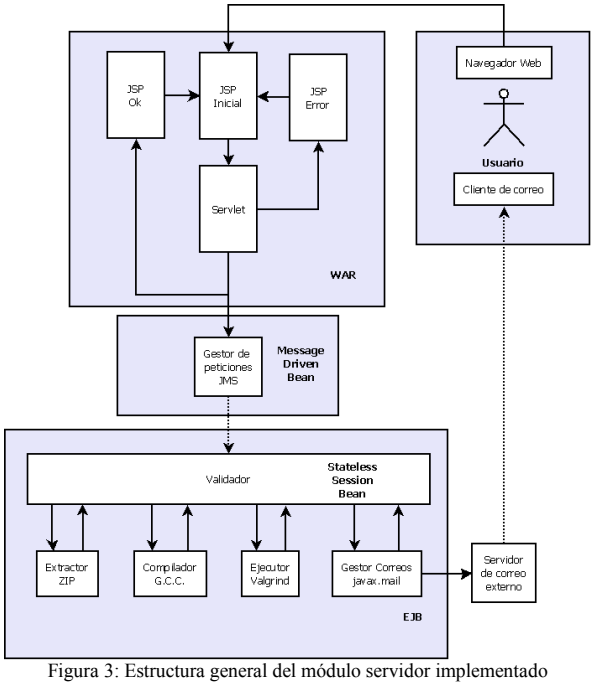

Figura 3: Estructura general del módulo servidor implementado

2.3. Aspectos de Implementación de Alto nivel

La visión de alto nivel (Figura 2) se refina con aspectos o detalles de más bajo nivel (Figura 3) referidos a aspectos tecnológicos sobre la implementación realizada. Dichos detalles cubren aspectos de la interfaz web diseñada, de los módulos hospedados en el servidor de la plataforma y de la estructura de datos utilizada en el servidor para almacenar temporalmente ejercicios.

La aplicación web empresarial diseñada (ver Figura 3), se divide en cuatro grandes bloques:

- *Presentación y tratamiento de la información*

Este bloque se empaqueta en un fichero WAR como establece la especificación de Java EE (Sun Microsystems 2005). También se encarga de ofrecer al usuario la interfaz para que introduzca sus datos y estos sean recogidos y analizados para comprobar que sean coherentes. Utiliza JSPs (Java Server Pages) y un *servlet* (Sun Microsystems 2005).

- *Gestor de peticiones*

Es bloque se implementa con un MDB (Message Driven Bean) que utiliza JMS (Java Messaging Service) (Sun Microsystems 2005) para implementar la cola de peticiones. Su misión es desacomplar el funcionamiento síncrono de la interfaz del navegador del resto del servidor.

- *Lógica de negocio*

Este bloque se encarga de procesar el código del usuario para llegar al resultado final. Está compuesto por un SSB (Stateless Session Bean) que se encarga de ir a llamando a los componentes para realizar las distintas tareas. Estos componentes, implementados como clases Java, son el extractor, compilador, ejecutor y gestor de correos.

- *Usuario*

En el bloque del usuario se encuentran el navegador web desde el que accede a la aplicación y el cliente de correo que recibe el

resultado de la validación. Su misión es la de comunicarse con los módulos anteriores de una forma eficaz.

2.3.1. *Interfaz Gráfica Web*

El acceso a la aplicación se hace mediante un cliente web. Éste consta de un grupo de tres páginas JSP; la principal tiene un formulario para introducir la información; las otras sirven para informar al usuario de que la petición se ha hecho con éxito o para advertirle de algún tipo de error. Los errores pueden deberse a que faltan datos en la petición o a que ha habido un error interno en el servidor.

Como se puede observar en Figura 4y Figura 5, hay tres modos de uso:

- **Memcheck.** Este modo utiliza la herramienta Memcheck de la suite Valgrind para analizar el programa en busca de problemas de memoria. No es necesario especificar ningún detalle más, salvo las opciones de compilación si hicieran falta.
- **Helgrind.** Este modo hace uso de la herramienta Helgrind para detectar problemas de concurrencia en el programa analizado. Si se elige este modo, significa que se está usando programación C concurrente y, por tanto, en las opciones de compilación habrá que poner como mínimo la opción `-lpthread`.
- **Experto**. Este modo brinda al usuario la oportunidad de utilizar la suite Valgrind como si estuviera instalada en su propio equipo. Al elegir este modo se activará una caja de texto usando código *javascript*, como se puede ver en la Figura 5, en la que el usuario deberá escribir el comando con el que quiere ejecutar Valgrind, pudiendo elegir la herramienta, opciones y argumentos deseados.

Figura 4: Interfaz gráfica básica

La tecnología *javascript* también se utiliza en el cliente para comprobar que el fichero que se intenta enviar al servidor tiene formato `.zip`. Si se intenta introducir otro tipo de archivo, se avisa al usuario con la ventana de información que se ve en la Figura 6. En caso de que el explorador utilizado por el usuario no soporte *javascript* y el acceso web permita enviar un fichero que no sea `zip`, será el módulo de extracción el que detecte el error.

Cuando todo el proceso de validación ha funcionado correctamente, se presentará al usuario una página como la de la Figura 7. En ella, se detallan los datos de la petición que incluyen el identificador del alumno, el nombre del fichero enviado, la dirección de respuesta, el modo de validación, las opciones de compilación y las de la herramienta de validación utilizada.

2.3.2. *Lógica de Presentación en el Servidor*

Este componente se encarga de recibir los datos del usuario codificados en la petición post (HTTP) enviada por el formulario (Figura 4). Una vez comprobada la información, se envía al módulo de encolado para esperar allí a ser procesada.

La Figura 8 muestra el funcionamiento más detallado del código del módulo de acceso. Como se puede ver, el JSP inicial hace uso de *javascript* para ver si el fichero adjuntado por el usuario es Zip, si es así pasa al *servlet* y si no lo indica al usuario para que rectifique.

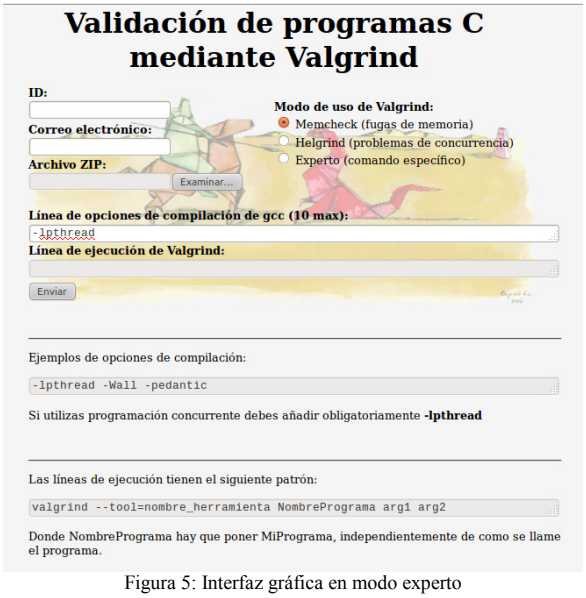

Figura 5: Interfaz gráfica en modo experto

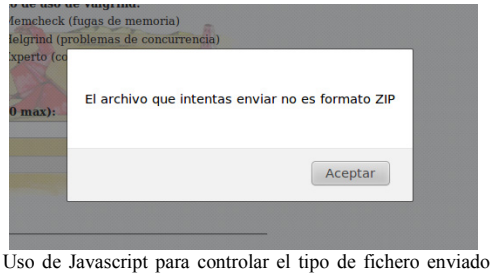

Figura 6: Uso de Javascript para controlar el tipo de fichero enviado por el usuario

Figura 7: Respuesta tipo ofrecida por la plataforma

El *servlet* se encarga de comprobar que no falten datos como ID, el correo electrónico, ni el comando de ejecución en caso de

haber elegido el modo *experto*. En caso de que el servlet no encuentre faltas en los datos, se enviará al usuario a un JSP informando del envío de la petición y se envía la petición a la cola. Si la información no es completa, se envía al usuario a un JSP indicándole el problema y dándole la oportunidad de que vuelva a intentarlo.

El *servlet*, por otro lado, se encarga de crear un directorio temporal (ver sección 2.3.4) para almacenar la petición, guardar en dicho directorio correspondientes el fichero comprimido y crear el fichero de texto con los datos resultado.

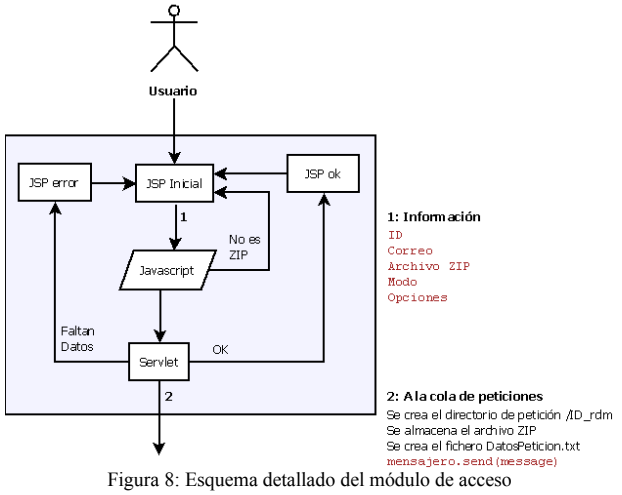

Figura 8: Esquema detallado del módulo de acceso

2.3.3. *Modulo de Desacople*

Está implementado utilizando la tecnología JMS que aporta Java EE y facilita la gestión de colas. Cuando el sistema está saturado y no puede aceptar más peticiones temporalmente, la cola evita negar el servicio al usuario. Lo que hace es almacenar la petición hasta que pueda ser procesada. Cuando llega el momento, la petición es enviada al siguiente módulo de ejecución. En este momento, la función de la cola para esa petición concreta llega a su fin.

La Figura 9 se observa el diagrama de flujo de este subcomponente. Se observa que cuando llega el momento, procesa los datos del mensaje y llama al método correspondiente del módulo encargado de procesar la información.

2.3.4. *Lógica de Negocio en el Servidor*

El módulo de gestión del servidor (Figura 10), en primer lugar, hace una llamada al extractor para que el archivo comprimido en `.zip` que el usuario envió pase a ser un directorio con el código fuente del programa. Este código tiene que ser accesible por los siguientes módulos y se almacenará en un directorio específico. Después, se llama al compilador para que, utilizando el código fuente extraído anteriormente, genere un archivo ejecutable. Seguidamente llama al ejecutor, que es el encargado de ejecutar la herramienta de Valgrind especificada sobre el archivo ejecutable generado. Por último, hará una llamada al gestor de correos. Tanto si ha habido un error durante el procesado, en cuyo caso habrá pasado aquí directamente, como si el proceso ha sido satisfactorio, el gestor de correos será el encargado de enviar el informe a la dirección que indicó el usuario.

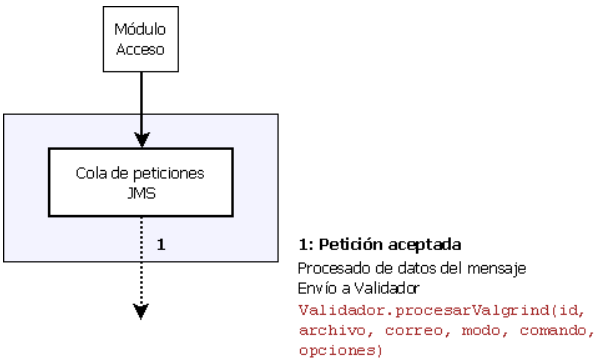

Figura 9: Relación entre el módulo de acceso a la información y el módulo de pruebas controlado mediante JMS

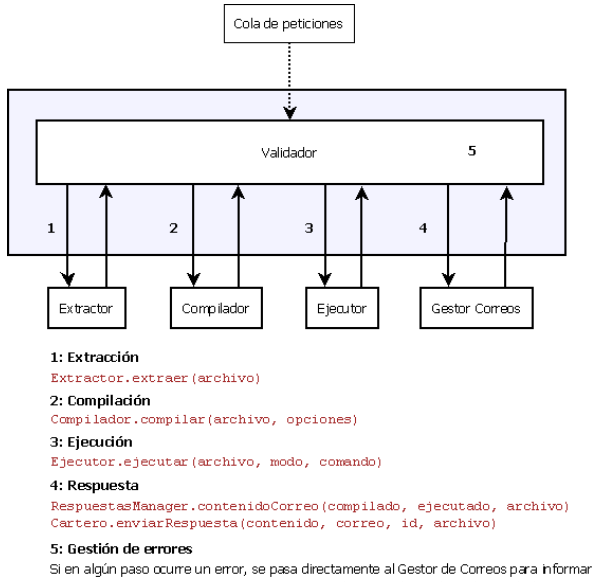

Figura 10: Esquema general correspondiente al núcleo del validador

El módulo extractor (Figura 11) se encarga de descomprimir el fichero enviado en formato `.zip`. La primera acción realizada es comprobar el tipo de fichero y después se llama al extractor correspondiente. En caso de que haya algún tipo de error, se devuelve un error al cliente.

Valgrind necesita tener el fichero ejecutable del programa a analizar. Por lo tanto, se compila el código fuente obtenido para llegar a ese punto en el lado del servidor. Además hay que hacerlo con las opciones adecuadas, como pueden ser la de depuración (obligatoria) o la de programación multihilos (opción del usuario). El módulo de compilación implementado (Figura 12) ejecuta un comando en un terminal de Linux en el que se utiliza la herramienta `gcc` con la opción de depuración `-d`, se incluyen todos los ficheros `.c` encontrados en el directorio de la petición y las opciones adicionales que el usuario ha indicado. El resultado es un fichero denominado `Ejecutable.out` almacenado en un directorio específico. En caso de error de compilación, se crea un

fichero de texto con la salida del comando `gcc` que es enviado al usuario como informe.

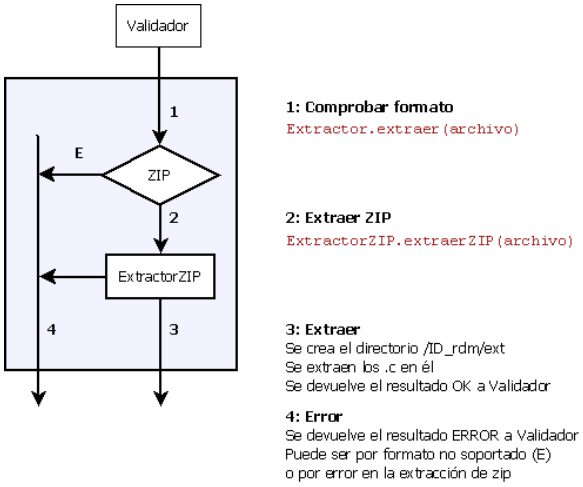

Figura 11: Módulo extractor perteneciente al núcleo del validador

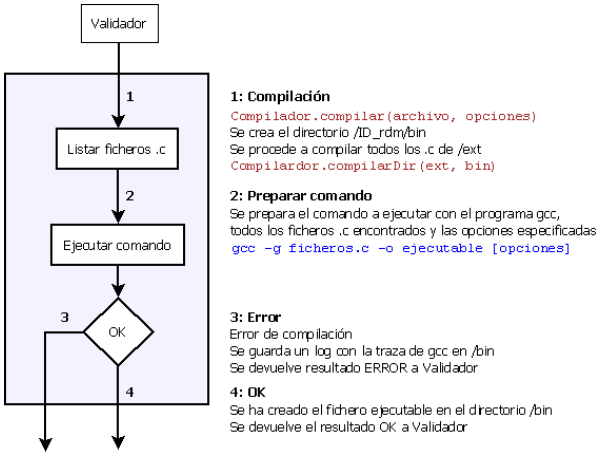

Figura 12: Esquema del módulo de compilación implementado

Este módulo de validación (Figura 13) ejecuta en un terminal de Linux el comando correspondiente con el fichero (i.e. `Ejecutable.out`). Dicho comando es compuesto en el código del módulo atendiendo al modo de ejecución elegido y a la línea introducida por el usuario en el formulario inicial. Si el modo es uno de los preestablecidos, Memcheck o Helgrind, se ejecutará el comando siguiente: `valgrind –tool=name Ejecutable`. (Donde en `name` estaría el nombre de la herramienta). Si el modo es Experto, se analiza la línea de ejecución introducida por el usuario, comprobando por cuestión de seguridad que lo que se quiere ejecutar es Valgrind y no otro comando. Con este modo, el usuario puede explotar todas las posibilidades que ofrece la suite Valgrind.

Además, este módulo dispone de un temporizador para evitar saturar el servidor con programas que tienen problemas como bucles infinitos. El límite de tiempo es un parámetro configurable por el administrador según sus requisitos. Si dicho tiempo se agota, la ejecución de Valgrind será abortada y se generará el consiguiente informe de error.

Como resultado de la ejecución se obtiene un fichero de texto que será enviado al usuario por correo electrónico. Dicho fichero contendrá el tiempo de ejecución empleado y la salida del comando ejecutado, ya sea el resultado de Valgrind sobre el programa del usuario, o un error de ejecución como, por ejemplo, un comando mal escrito en el modo Experto.

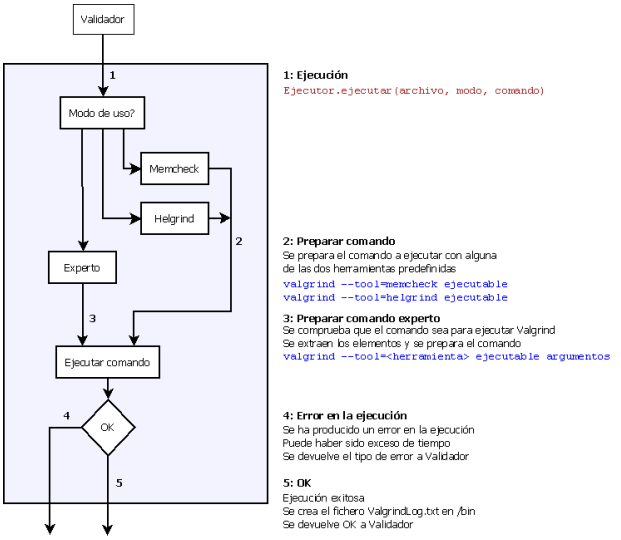

Figura 13: Módulo encargado de realizar la validación con la herramienta Valgrind

El módulo de notificación de usuario es el último de la plataforma (ver esquema interno en Figura 14). Es el encargado de enviar al servidor de correo externo el email con la dirección del usuario utilizando la librería `javax.mail` perteneciente al entorno Java EE. Los tipos de errores que maneja son:

- *Error de extracción*. Avisa de errores en la extracción del fichero.
- *Error de compilación*. El mensaje muestra todo el log generado por el GCC.
- *Error de ejecución*. Informa sobre problemas de tiempo excedido y sobre los resultados obtenidos por Valgrind.

2.3.5. Estructura de Almacenamiento Local

Cuando el servidor recibe una petición, éste guarda la información que el usuario le ha facilitado y organiza una estructura de directorios interna donde almacenar los distintos recursos que se generan en el procesado del código del usuario. El esquema de esta estructura se ve en la Figura 15.

En primer lugar, hay un directorio en el servidor donde se guardarán todas las peticiones que se reciben. De él cuelga un directorio por cada petición recibida y que tendrá como nombre el ID del usuario más un número aleatorio único que evite colisiones entre las peticiones.

Dentro del directorio de cada petición se guardará el fichero `.zip` enviado por el usuario y un fichero de texto con los datos de la petición (fecha y hora, nombre y correo del usuario, nombre del fichero `.zip`, modo de uso, opciones de compilación y opciones de ejecución). Además existe un directorio `ext` donde se guardará el código `.c` extraído del fichero comprimido y un

directorio `bin` donde se almacena el fichero ejecutable y un fichero de texto con el resultado de Valgrind.

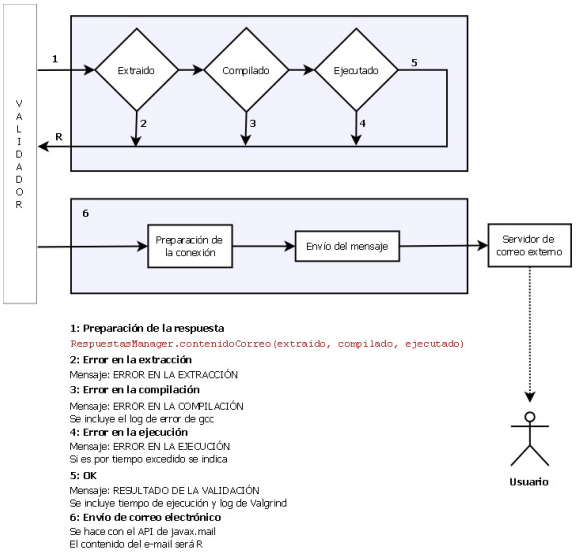

Figura 14: Módulo de notificación dentro del esquema de validación

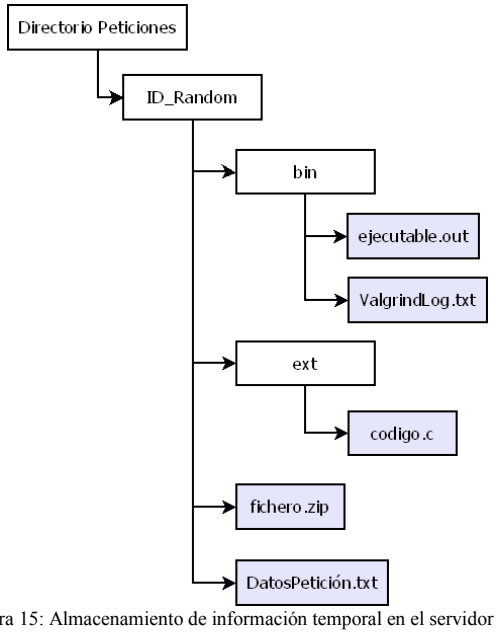

Figura 15: Almacenamiento de información temporal en el servidor

En primer lugar, hay un directorio en el servidor donde se guardarán todas las peticiones que se reciben. De él cuelga un directorio por cada petición recibida y que tendrá como nombre el ID del usuario más un número aleatorio único que evite colisiones entre las peticiones.

Dentro del directorio de cada petición se guardará el fichero `.zip` enviado por el usuario y un fichero de texto con los datos de la petición (fecha y hora, nombre y correo del usuario, nombre del fichero `.zip`, modo de uso, opciones de compilación y opciones de ejecución). Además existe un directorio `ext` donde se guardará el código `.c` extraído del fichero comprimido y un directorio `bin` donde se almacena el fichero ejecutable y un fichero de texto con el resultado de Valgrind.

2.4. Ejemplo de Uso

A fin de ilustrar el funcionamiento, se muestra un ejemplo sencillo del tipo de aplicación que la plataforma puede validar. Se trata un ejemplo básico de dos hilos que comparte una variable que modifican en lectura/escritura. Dicho ejemplo es sencillo y sirve para explicar lo que es una condición de carrera. El ejemplo completo se muestra en Figura 16.

```
#include <stdio.h>
#include <pthread.h>

int contador_compartido=0;

void* hilo_fn( void* arg ) {
   contador_compartido++;
   return NULL;
}

int main ( ) {
   contador_compartido=0;
   pthread_t pth1;
   pthread_t pth2;
   pthread_create(&pth1, NULL, hilo_fn, NULL);
   pthread_create(&pth2, NULL, hilo_fn, NULL);
   pthread_join(pth2, NULL);
   return 0;
}
```

Figura 16: Ejemplo con condición de carrera en el acceso a una variable contador compartido

```
RESULTADO DE LA VALIDACION:

Tiempo empleado en la ejecución: 1920 milisegundos

==12990== Helgrind, a thread error detector
==12990== Copyright (C) 2007-2011, and GNU GPL'd, by OpenWorks LLP et al.
==12990== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==12990== Command: ./riai
==12990== ---Thread-Announcement----------------------------------------
==12990== Thread #3 was created
==12990==    at 0x4153D28: clone (clone.S:111)
==12990== ---Thread-Announcement----------------------------------------
==12990==
==12990== Thread #2 was created
==12990==    at 0x4153D28: clone (clone.S:111)
==12990== ------------------------------------------------------------
==12990==
==12990== Possible data race during read of size 4 at 0x804A020 by thread #3
==12990== Locks held: none
==12990==    at 0x8048497: hilo_fn (in 2012-RIAI-EnterpriseModule4C/riai)
==12990==    by 0x402DD35: ??? (in x86-linux.so)
==12990==    by 0x4050D4B: start_thread (pthread_create.c:308)
==12990==    by 0x4153D3D: clone (clone.S:130)
==12990== This conflicts with a previous write of size 4 by thread #2
==12990== Locks held: none
==12990==    at 0x804849F: hilo_fn (in  2012-RIAI-EnterpriseModule4C/riai)
==12990==    by 0x402DD35: ??? (in …)
==12990==    by 0x4050D4B: start_thread (pthread_create.c:308)
==12990==    by 0x4153D3D: clone (clone.S:130)
---
==12990== Possible data race during write of size 4 at 0x804A020 by thread #3
==12990== Locks held: none
==12990==    at 0x804849F: hilo_fn (in /riai)

==12990==
==12990== For counts of detected and suppressed errors, rerun with: -v
==12990== Use --history-level=approx or =none to gain increased speed, at
==12990== the cost of reduced accuracy of conflicting-access information
==12990== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Figura 17: Informe generado por la plataforma para el ejemplo de Figura 20 con condición de carrera.

En un principio tiene condición de carrera y si se manda a la plataforma de entrega en un `.zip`, la herramienta devuelve un correo donde dice que ha habido un error tal y como se muestra en la Figura 17. Esta condición de carrera se puede solventar usando un cerro POSIX (mutex) tal y como muestra la Figura 18. Si se vuelve a mandar al servidor este fichero comprimido, la herramienta indicaría que no ha encontrado el error (Figura 19).

```
#include <stdio.h>
#include <pthread.h>

int contador_compartido=0;
pthread_mutexattr_t    attr;
pthread_mutex_t       mutex;

void* hilo_fn( void* arg ) {
   pthread_mutex_lock(&mutex);
   contador_compartido++;
   pthread_mutex_unlock(&mutex);
   return NULL;
}

int main ( ) {
   contador_compartido=0;
   pthread_mutexattr_settype(&attr,
   PTHREAD_MUTEX_RECURSIVE);
   pthread_mutex_init(&mutex, &attr);

   pthread_t pth1;
   pthread_t pth2;
   pthread_create(&pth1, NULL, hilo_fn, NULL);
   pthread_create(&pth2, NULL, hilo_fn, NULL);
   pthread_join(pth2, NULL);
   return 0;
}
```

Figura 18: Ejemplo con condición de carrera en el acceso a contador_compatido

```
RESULTADO DE LA VALIDACION:

Tiempo empleado en la ejecución: 1320 milisegundos

==13178== Helgrind, a thread error detector
==13178== Copyright (C) 2007-2011, and GNU GPL'd, by OpenWorks LLP et al.
==13178== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==13178== Command: ./riai
==13178==
==13178==
==13178== For counts of detected and suppressed errors, rerun with: -v
==13178== Use --history-level=approx or =none to gain increased speed, at
==13178== the cost of reduced accuracy of conflicting-access information
==13178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 8)
```

Figura 19: Resultados mostrado con la plataforma cuando no hay problemas. Se corresponde con el código de la Figura 18.

3. Rendimiento de la Plataforma Evaluada en un Curso de Concurrencia para C-POSIX

Los objetivos de la experimentación realizada con la plataforma desarrollada son:

- Evaluar la utilidad de la herramienta dentro de un curso dedicado a la concurrencia. Esta evaluación se hace viendo la información que la herramienta aporta en curso real, siguiendo la estrategia de evaluación descrita en (Basanta-Val .P et al. 2012) para entornos Java concurrente. También se evalúa la utilidad sobre un curso de concurrencia ajeno a los autores del artículo, viendo las capacidades de la herramienta.
- Evaluar si la herramienta es capaz de ser integrada en un curso de programación C-concurrente. Experiencias previas con validación formal (Basanta-Val .P et al. 2012) arrojaron como resultado que la herramienta no era directamente integrable dentro de una sesión de 1,5 horas de laboratorio con 20 alumnos.

3.1. Escenarios de Prueba

Para ello se han utilizado dos cursos. El primero de ellos impartido por parte de los autores de este artículo en la Universidad Carlos III de Madrid (UC3M). Es curso de concurrencia donde los docentes enseñan a alumnos de 3er curso programación concurrente en C sobre la base previa de otro curso donde ya ha aprendido a programar en C. Este curso cuenta con 56 alumnos que realizan las prácticas típicamente en parejas, siguiendo metodologías adatadas al EEES (Espacio Europeo de Educación Superior). El tipo de ejercicio propuesto es de desarrollo de una aplicación concurrente en C, haciendo uso de diferentes mecanismos de gestión de la concurrencia en C. Durante el curso se realizan prácticas introductorias sobre los mecanismos POSIX básicos para la compartición de hilos (ver Tabla 1).

Tabla 1: Ejercicios utilizados en el curso de la UC3M

	Descripción del problema de laboratorio	Complejidad
`UC3M.2.1.c`	Cuenta bancaria con hilo lector y escritor	Baja
`UC3M.2.2.c`	Practica de pasillo	Alta
`UC3M.3.c`	Practica del Santa Claus	Media/Alta
`UC3M.4.1.c`	p1: mutex	Media
`UC3M.4.2.c`	p2: mutex y variables de condición	Media
`UC3M.4.3.c`	Suma: Ejercicios de suma con mutex y variable de condición	Media

Tabla 2: Características del curso en la Lawrence Livermore National Laboratory

Denominación Del fichero	Breve Descripción	Opciones GCC
`Bug1.c`	Cuatro hilos que se bloquen a la espera de una señal. Sólo uno continúa.	
`Bug1_fix.c`	Ejercicio bug1 arreglado.	
`Bug2.c`	No reserva explícitamente el espacio de memoria que luego usa	
`Bug2_fix.c`	Ejercicio bug_2 arreglado.	
`Bug3.c`	Problema de paso de parámetros	
`Bug4.c`	Problema de sincronización que causa bloqueo.	-lm
`Bug4_fix.c`	Ejercicio bug_4 arreglado	-lm
`Bug5.c`	El programa finaliza antes que los hilos que tiene asociado.	
`Bug6.c`	Condición de carrera sobre variable local	
`Bug6_fix.c`	Solución de la condición de carrera de bug_6	

El segundo tipo de curso evaluado ha sido un curso de programación concurrente para C-POSIX disponible en Internet. Dicho material se ha obtenido del Lawrence Livermore National Laboratory (LLNL), un centro de computación de alto rendimiento de EE.UU. Dichos ejercicios se pueden encontrar en el apartado de tutoriales y ejemplos de la página de su página web (Lawrence Livermore National Laboratory). Una particularidad

de estos ejercicios es que muchos ofrecen ejemplos de programas tipo con solución. Para cada programa (ver Tabla 1) se le pide al alumno que encuentre un error y lo solucione. Para aquellos casos donde hay que modificar el código sustancialmente, el material de la unidad didáctica también provee el ejercicio solucionado (que aparece con el sufijo `fix`).

Tabla 3: Resultados arrojados por la herramienta sobre el curso de la UC3M. 1 GHZ- 2 Gigabytes de Memoria- GlassFish 3.2

	Resultados de Herramienta				
	Bucle Inf (20 seg.)	Memcheck	Error en la entrega (Helgrind)	Condiciones de carrera (data race)	Mal usos del API de POSIX
UC3M.2.1.c	0%	0%	3,70%	3,70%	3,70%
UC3M.2.2.c	12%	8%	54,17%	25,00%	33,33%
UC3M.3.c	100%	0%	37,93%	17,24%	34,48%
UC3M.4.1.c	7%	21%	20,69%	0,00%	24,14%
UC3M.4.2.c	0%	0%	26,92%	3,85%	26,92%
UC3M.4.3.c	9%	14%	36,36%	22,73%	36,36%

En el grupo de 29 parejas se ha visto que la herramienta ha detecta problemas en alguna de las entregas de todos los ejercicios. En el caso de la práctica más sencilla (UC3M.2.1.c) es del 3,7% y aumenta con la complejidad del ejercicio. A mayor complejidad del ejercicio más sencillo que exista algún tipo de problema con la concurrencia de las entregas. De esta manera, en la práctica de la los pasillos, la más compleja, más de la mitad de las entregas se podrían beneficiar de la información ofertada por la herramienta, si esta estuviese disponible de forma online para la realización de la práctica. También resulta beneficioso para el profesor que puede utilizar dicha información como parte del proceso de evaluación.

Los resultados obtenidos de la aplicación de la herramienta como complemento en el curso del LLNL se muestran en Tabla 4. En este caso se contempla cuantos de los ejercicios cortos propuestos basados en la detección de errores, pueden beneficiarse del soporte de la herramienta. En dicho proceso, se ha analizado la información que ofrece el propio ejercicio propuesto donde el programa i) se cuelga, ii) genera fallos de segmento (*core dumps*) o iii) produce una respuesta incorrecta. Dicha información se compara con la que ofrece la herramienta web implementada. La herramienta web puede devolver los siguientes errores: i) vencimiento de temporizador, ii) problema de acceso a memoria, y iii) problemas de condición de carrera o interbloqueo de procesos.

El primer resultado observado es que hay correlación entre la información de temporización y las aplicaciones que se cuelgan. Los dos ejercicios (`Bug1` y `Bug4`) donde se cuelga la ejecución, hacen que venza el temporizador.

La integración dentro de la plataforma de la herramienta Memcheck permite detectar un problema de acceso a memoria no inicializado en el ejercicio `Bug2` donde el programa produce un *coredump* que detecta el problema en la compartición de datos de los hilos. Asimismo, la inclusión de la suite del Helgrind permite detectar problemas de condición de carrera correspondientes al ejercicio `Bug6`. Es de destacar que las soluciones de los ejercicios (`Bug1_fix`, `Bug2_fix`, `Bug4_fix` y `Bug6_fix`) están libres de errores de temporización, memoria y problemas de concurrencia, de acuerdo a lo esperado. Esa información extra es aportada por la herramienta de validación-web implementada.

En el caso de `Bug5` y `Bug3` la información aportada por las herramientas (i.e., que no hay errores ni de memoria ni de concurrencia) también es útil para el usuario de la plataforma. El que no existan problemas de concurrencia ni de acceso a memoria le ayuda a detectar el origen de la muerte prematura y los datos erróneos.

En el código original, la herramienta-web implementada ha encontrado un de la solución `Bug2_fix`. En la solución propuesta por los autores del material originales se introduce un problema de gestión de memoria adicional (de tipo assign.). Consultado el código se ha visto que la herramienta web estaba en lo cierto y se ha solucionado.

Tabla 4: Características de los ejercicios usados. Velocidad de ejecución. 1 GHZ- 2 gigabytes de memoria- GlassFish 3.2

	Sin soporte	Con soporte de plataforma		
		Temp.	`Mem.ck (error)`	`Helgrind (errors)`
`Bug1.c`	Bucle inf.	Sí	0	0
`Bug1_fix.c`	OK	No	0	0
`Bug2.c`	Core dump.	No	Assign.	0
`Bug2_fix.c`	OK	No	0	0
`Bug3.c`	Datos erróneos	No	0	0
`Bug4.c`	Bucle inf.	Sí	0	0
`Bug4_fix.c`	OK	No	0	0
`Bug5.c`	Muerte de hilo	No	0	0
`Bug6.c`	Datos erróneos	No	0	Race
`Bug6_fix.c`	OK	No	0	0

Tabla 5: Tiempos de ejecución de la herramienta (medidos en segundos) para las pruebas en UC3M y LLNL. 1 GHZ- 2 Gigabytes de Memoria- GlassFish 3.2

	Sin soporte	Con soporte de plataforma	
		`Memcheck`	`Helgrind`
`UC3M.2.1.c`	0,40	0,50	0,70
`UC3M.2.2.c`	2,00	3,00	1,00
`UC3M.3.c`	20,00	20,00	20,00
`UC3M.4.1.c`	6,00	10,00	11,00
`UC3M.4.2.c`	6,00	10,00	11,00
`UC3M.4.3.c`	6,00	11,00	12,00
`Bug1.c`	20,00	20,00	20,00
`Bug1_fix.c`	10,01	10,56	10,77
`Bug2.c`	3,05	5,82	7,75
`Bug2_fix.c`	3,05	5,87	7,73
`Bug3.c`	1,00	3,14	1,78
`Bug4.c`	20,00	20,00	20,00
`Bug4_fix.c`	1,19	3,07	3,81
`Bug5.c`	0,01	2,09	5,13
`Bug6.c`	0,03	1,05	6,78
`Bug6_fix.c`	0,02	1,00	7,02

3.2. Sobrecarga de la Herramienta y Coste Computacional

El caso de pruebas utilizado es el mismo descrito con anterioridad (Tabla 1 y Tabla 2). Los resultados obtenidos se muestran en la Tabla 5 y se corresponden a un caso donde hay un único usuario (alumno) utilizando el sistema. Como se puede ver, los tiempos obtenidos colocan a la herramienta en tiempo de respuesta en el peor de los casos cercanos a los 10 segundos e

inferiores a los 17 milisegundos en el mejor de los casos. Por último, hay un caso (marcado con – en la Tabla 3) especial. Es el caso en el cual la aplicación se queda bloqueada hasta que ha vencido su temporizador.

Para completar el estudio sobre el rendimiento se ha medido la sobrecarga introducida en el coste total debida a la validación temporal realizada en el servidor (ver Figura 20). Los resultados muestran una gran variación porcentual, que va desde el un mínimo del 5% a un máximo de 1780%. Estos máximos se obtienen con programas muy rápidos que ejecutan Helgrind. Es en estos casos donde las diferencias son más marcadas, pero aún así son bajas en términos absolutos (por ejemplo 6 segundos para Bug_6).

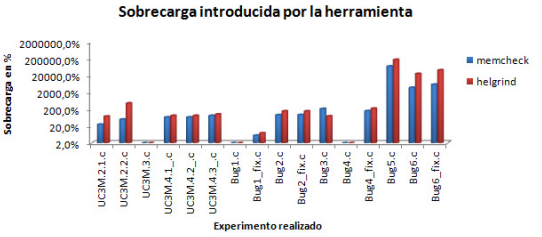

Figura 20: Sobrecarga de ejecución introducida por la herramienta. 1 GHZ- 2 Gigabytes de Memoria- GlassFish 3.2

3.2.1. Aplicabilidad en un Curso Adaptado a Bolonia

La última parte de la evaluación considera el despliegue de esta herramienta en un curso real impartido. Típicamente, ese curso tipo tiene un máximo de 20 parejas que utilizan concurrente el sistema de validación C diseñado. El experimento consiste en establecer la cota máxima de tiempo de respuesta del sistema cuando todos los alumnos usan el servidor.

Dicho tiempo se corresponde con el caso en el que todos los alumnos envían algo al servidor para que realice la validación. Se supone que un alumno no vuelve a enviar una práctica mientras no recibe contestación de ésta.

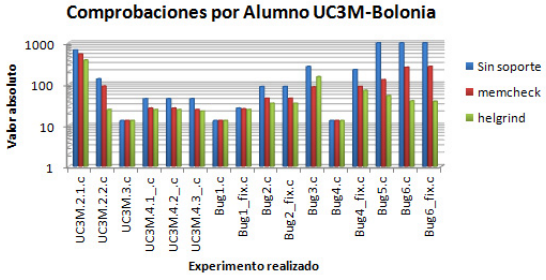

Figura 21: Comprobaciones máximas realizables por ejercicio. 1 GHZ- 2 gigabytes de memoria- GlassFish 3.2

Los resultados obtenidos en el curso de la UC3M y del curso del LLNL se muestran en la Figura 21. Dicha figura representa muestra el número de veces máximo de comprobaciones que todos los alumnos haciendo peticiones al mismo tiempo contra el servidor. En los casos de prueba estudiados este valor va desde un mínimo de 13 pruebas cada 1,5 horas por cada alumno, cuando se envían bucles infinitos, a las 267 cada 1,5 horas cuando son programas cortos donde dos hilos comparten información.

Este rango temporal hace útil a la herramienta a la hora de realizar las comprobaciones dentro de lo que podría ser una clase tipo de 1,5 horas para los casos estudiados. Sin embargo, los tiempos absolutos muestran que sigue siendo bastante interesante la notificación por correo, especialmente en aquellos casos que la máquina esté bloqueada hasta que venza el temporizador de la práctica.

Por tanto, no se debería por tanto prescindir de un sistema de notificación asíncrono al usuario. Esta afirmación es especialmente cierta en aquellos casos donde la práctica bloquea el servidor hasta que vence el temporizador.

4. Trabajo Relacionado

En el trabajo relacionado se analizan diferentes plataformas y herramientas que han influido en el desarrollo realizado. También se analizan diferentes experiencias docentes en C que utilizan núcleos de ejecución similares a los propuestos.

La primera herramienta está descrita en (Basanta-Val .P et al. 2012). Dicha herramienta opta por el uso de validadores formales (Basanta-Val .P et al. 2012) como núcleo de apoyo a la docencia concurrente. En la experiencia realizada con validadores formales se había visto que los tiempos de respuesta eran muy altos y que podían superar con creces el tiempo disponible para una sesión de laboratorio (1,5 horas en la universidad Carlos III de Madrid). La plataforma propuesta en este artículo (Secciones 3 y 4) es mucho más ligera pues tiene un motor de ejecución en el servidor más ligero, lo que le permite ejecutar varios centenares de prácticas en una sola sesión de laboratorio. La validación de una práctica puede ser realizada en la plataforma propuesta en 20 segundos como mucho, dicha ventaja viene principalmente de la utilización de una técnica de validación más sencilla. El modelo previamente propuesto en (Basanta-Val .P et al. 2012) también ha sido simplificado, eliminándose casos de uso de gestión; y mejorado, integrándose tecnología de apoyo en el cliente. Dicho apoyo en el cliente incluye validación mediante javascript de que los parámetros introducidos son los adecuados y que el formato del archivo también lo es.

Otra plataforma con la cual guarda una gran relación la plataforma propuesta es la propuesta en para sistemas de control y middleware descrita en (Garcia-Valls, Basanta-Val 2012) para entornos industriales. La presente herramienta de validación es integrable como componente Java EE dentro de la arquitectura propuesta por (Garcia-Valls, Basanta-Val 2012). Dicha integración permite que (Garcia-Valls, Basanta-Val 2012) pueda acceder a los módulos de validación de forma general; que ahora no posee, remotamente como si fuesen locales.

Por último, es de destacar la infraestructura descrita en (Pardo, Kloos 2011) que permite que los alumnos de comunidades grandes puedan compartir información a través de Internet de forma escalable. Estas estrategias son integrables con la herramienta propuesta en este artículo a nivel de sistema de ficheros. Ambas plataformas pueden ser integradas a dicho nivel para compartir información entre varios alumnos.

La integración de herramientas genéricas como Valgrind en cursos de programación en C es bastante común al aportar una serie de ventajas como son detección de fugas de memoria o de problemas de concurrencia. Como muestra se citan (Lee, Kester & Schulzrinne 2011) y (Sierra et al. 2012). En (Lee, Kester & Schulzrinne 2011) los profesores encargados del curso penalizan

los fallos de los alumnos detectados mediante la herramienta Valgrind, considerando que aporta una información valiosa. Por otro lado en (Sierra et al. 2012) la herramienta Valgrind forma parte de la validación hecha en los laboratorios para que el alumno sea consciente de la calidad de su proyecto software. En el caso del presente artículo es una herramienta web sirve de apoyo al material de aprendizaje de la concurrencia en C.

5. Conclusiones y Líneas Futuras

Como conclusión general se extrae que la docencia se puede beneficiar de herramientas que permitan visualizar y detectar los problemas existentes en cierta disciplina. Ello es especialmente cierto en casos como es el de la enseñanza concurrente en C donde estos problemas resultan especialmente complejos. En este entorno, una infraestructura como la presentada en este artículo permite aliviar parte de los problemas derivados de dicha problemática mediante la integración web de núcleos de compilación y ejecución ya preexistentes en una arquitectura multi-capa típica de Internet.

Los resultados empíricos obtenidos tras la evaluación de la plataforma propuesta en las condiciones de trabajo de un curso dedicado a la docencia concurrente en el lenguaje C muestran que la relación coste beneficio de la herramienta. Resultados sobre trazas disponibles de alumnos muestran que la herramienta aporta en un gran número de casos información que susceptible de ser integrada en el proceso de revisión y de aprendizaje, sobre todo en casos con cierta complejidad. Por otro lado, la elección de un núcleo de validación ligero permite que los alumnos puedan obtener unos tiempos de validación mejores que los ofertados por validadores formales.

A partir de este trabajo, donde la plataforma aporta información valiosa en un tiempo aceptable, se plantea el reto futuro de medir el aprendizaje cuando la herramienta se deja en abierto a los alumnos y el beneficio para el profesor cuando la integra dentro de su proceso de calificación. El trabajo en curso relacionado con esta plataforma progresa también en la línea de explorar la herramienta propuesta en un curso de doctorado. El plan actual de actuación abordaría problemas de la concurrencia cliente-servidor y nuevos modelos de concurrencia y gestión de memoria (e.g. (Basanta Val, Garcia-Valls 2013) (Cuevas et al. 2013) (Sáez, Crespo 2013)) que están siendo validados mediante núcleos basados en nuevos motores (Jannesari et al. 2009) (Serebryany, Iskhodzhanov 2009).

English Summary

Lightweight Web-Tool for C Concurrent Programming

Abstract

Tools for computer-aided teaching and learning provide multiple benefits from the point of view of teaching because it allows emphasizing or illustrating certain issues that are sometimes difficult to emphasize without such type of support. This is exactly the case for the tools to detect if there is any type of problem in a concurrent-C program. These tools provide interfaces that can complement the information given by a compiler with additional information about different types of race conditions and memory leaks that appear in the code. This article aims to address how to integrate a core validation tools for concurrent-C as a web application, allowing you to be accessible through the Internet. This tool has been evaluated in an existing programming course, which has shown to be able to provide additional information useful to the learner and the teacher. There have also been a number of performance measures to establish operational limits designed tool within a course that teaches concurrent-C programming.

Keywords:

Tools, Industrial Informatics, Concurrent Systems, Education, C.

Agradecimientos

Este trabajo ha sido realizado con el apoyo del proyecto iLAND (ARTEMIS-JU 100026) parcialmente financiado por ARTEMIS JTU y el Ministerio de Industria, Comercio y Turismo español y también de forma parcial por REM4VSS (TIN2011-28339) del Ministerio de Ciencia e Innovación y e-Madrid (S2009/TIC-1650). Contiene también resultados parciales del proyecto de innovación docente “Sistema de Evaluación Genérico Basado En el Mecanismo de Rúbrica Participado Por el Alumno y bajo Supervisión Parcial del Profesor” del Programa 2011-12 de Innovación Docente de la Universidad Carlos III de Madrid y de “Mejora de la Eficacia del Aprendizaje mediante un Mecanismo de Hitos Participado por Alumnos y Profesores” del Programa 2012-2013. Por último, los autores agradecen a todos los profesores de la asignatura utilizada en el experimento su aporte al artículo y a los anónimos revisores del artículo su aporte a la mejora de la calidad del presente artículo.

Referencias

Alonso, D., Pastor, J. & Álvarez, B. 2004, "Real–Time Teaching with Java: JPR 3" in On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, eds. R. Meersman, Z. Tari & A. Corsaro, Springer Berlin Heidelberg, , pp. 246-255.

Basanta Val, P. & Garcia-Valls, M. 2013, "A Distributed Real-Time Java-centric Architecture for Industrial Systems", Industrial Informatics, IEEE Transactions on, vol. PP, no. 99, pp. 1-1.

Basanta-Val .P, García-Valls, M., Estévez-Ayres, I. & Martin-Gutiérrez, M.J. 2012, "Módulo Empresarial para la Validación Formal de Ejercicios aplicado a la Programación Concurrente en Java", Revista Iberoamericana de Automática e Informática Industrial RIAI, vol. 9, no. 3, pp. 209-299.

Bouyssounouse, B. & Sifakis, J. 2005, Embedded systems design: the ARTIST roadmap for research and development, Springer, Verlag, NJ, USA.

Caspi, P., Folher, G., Garcia-Valls, M., Kopetz, H., Lakhnech, Y., Laroussinie, F., Lavagno, L., Lipari, G., Maraninchi, F., Peti, P., Puente, J.d.l., Sangiovanni-Vincentelli, A., Scaife, N., Sifakis, J., de Simone, R., Torngren, M., Veríssimo, P., Wellings, A.J., Wilhelm, R., Willemse, T., Yi, W., Almeida, L., Benveniste, A., Bouyssounouse, B., Buttazzo, G., Crnkovic, I., Damm, W. & Engblom, J. 2005, "Guidelines for a graduate curriculum on embedded software and systems ", ACM Transactions on Embedded Computing Systems, vol. 4, no. 3.

Committee, P.A.S. 2003, POSIX Realtime and Embedded application Support, IEEE Standard for Information Technology.

Crenshaw, T.L. 2013, "Using Robots and Contract Learning to Teach Cyber-Physical Systems to Undergraduates", IEEE Trans.Education, vol. 56, no. 1, pp. 116-120.

Cuevas, C., Barros, L., Martínez, P.L. & Drake, J.M. 2013, "Beneficios que aporta la metodología MDE a los entornos de desarrollo de sistemas de tiempo real", Revista Iberoamericana de Automática e Informática Industrial RIAI, vol. 10, no. 2, pp. 216-227.

Estevez-Avres, I., Basanta-Val P. & García-Valls, M. 2004, "Docencia de programación concurrente. Experiencias de Laboratorio.", VII Jornadas de Tiempo Real.

Garcia-Valls, M. & Basanta-Val, P. 2012, "Usage of DDS Data-Centric Middleware for Remote Monitoring and Control Laboratories", Industrial Informatics, IEEE Transactions on, vol. PP, no. 99, pp. 1.

Hamblen, J.O. & Bekkum, G.M.E.v. 2013, "An Embedded Systems Laboratory to Support Rapid Prototyping of Robotics and the Internet of Things", IEEE Trans.Education, vol. 56, no. 1, pp. 121-128.

Havelund, K. & Pressburger, T. 2000, "Model checking Java programs using Java PathFinder", International Journal on Software Tools for Technology Transfer (STTT), vol. 2, no. 4, pp. 366-381.

Ihantola, P. 2006, "Test data generation for programming exercises with symbolic execution in Java PathFinder", Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006ACM, New York, NY, USA, pp. 87.

Jannesari, A., Kaibin Bao, Pankratius, V. & Tichy, W.F. 2009, "Helgrind+: An efficient dynamic race detector", Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, may, pp. 1.

Kim, S.H. & Jeon, J.W. 2009, "Introduction for Freshmen to Embedded Systems Using LEGO Mindstorms", Education, IEEE Transactions on, vol. 52, no. 1, pp. 99.

Lawrence Livermore National Laboratory , POSIX Threads Programming Exercise [2012, 8].

Lee, J.W., Kester, M.S. & Schulzrinne, H. 2011, "Follow the river and you will find the C", Proceedings of the 42nd ACM technical symposium on Computer science educationACM, New York, NY, USA, pp. 411.

Merino, P.J.M., Molina, M.F., Organero, M.M. & Kloos, C.D. 2012, "An adaptive and innovative question-driven competition-based intelligent tutoring system for learning", Expert Syst.Appl., vol. 39, no. 8, pp. 6932.

Monzón, A., Fernández, J.L. & de la Puente, J.A. 2012, "Application of Deadlock Risk Evaluation of Architectural Models", Software: Practice and Experience, vol. 42, no. 9, pp. 1137-1163.

Nethercote, N. & Seward, J. 2007, "Valgrind: a framework for heavyweight dynamic binary instrumentation", SIGPLAN Not., vol. 42, no. 6, pp. 89-100.

Pardo, A. & Kloos, C.D. 2011, "SubCollaboration: large-scale group management in collaborative learning", Softw.Pract.Exper., vol. 41, no. 4, pp. 449-465.

Pinto, M., Moreira, A.P. & Matos, A. 2012, "Localization of Mobile Robots Using an Extended Kalman Filter in a LEGO NXT", Education, IEEE Transactions on, vol. 55, no. 1, pp. 135.

Rodriguez-Andina, J.J. & Gomes, L. 2013, "Guest Editorial Special Section on Information Technologies Within Engineering Education", Industrial Informatics, IEEE Transactions on, vol. 9, no. 1, pp. 546.

Sáez, S. & Crespo, A. 2013, "Mejora de los Test de Planificabilidad para Asignación Incremental de Tareas en Sistemas Multiprocesadores de Tiempo Real", Revista Iberoamericana de Automática e Informática Industrial RIAI, vol. 10, no. 2, pp. 197-203.

Salido, J. & Lillo, A. Déniz Suárez, O., Bueno, G. 2011, "CTRWeb: Una Herramienta de Programación para Telecontrol de Sistemas Físicos Educativos", Revista iberoamericana de automática e informática industrial, vol. 8, no. 1, pp. 89-99.

Santana, I., Ferre, M., Izaguirre, E., Aracil, R. & Hernandez, L. 2013, "Remote Laboratories for Education and Research Purposes in Automatic Control Systems", Industrial Informatics, IEEE Transactions on, vol. 9, no. 1, pp. 547.

Savage, S., Burrows, M., Nelson, G., Sobalvarro, P. & Anderson, T. 1997, "Eraser: a dynamic data race detector for multithreaded programs", ACM Trans.Comput.Syst., vol. 15, no. 4, pp. 391-411.

Serebryany, K. & Iskhodzhanov, T. 2009, "ThreadSanitizer: data race detection in practice", Proceedings of the Workshop on Binary Instrumentation and ApplicationsACM, , pp. 62.

Sierra, A.J., Ariza, T., Fernandez, F.J. & Madinabeitia, G. 2012, "TVSP: A Tool for Validation Software Projects in programming labs", Global Engineering Education Conference (EDUCON), 2012 IEEE, april, pp. 1.

Sifakis, J. 2011, "A vision for computer science — the system perspective", Central European Journal of Computer Science, vol. 1, no. 1, pp. 108-116.

Sun Microsystems 2005, Online [2005] at http://jcp.org/aboutJava/communityprocess/pr/jsr220/index.html-last update, Enterprise Java Beans [Homepage of SUN], [Online].

Weber, J. & Rehkopf, A. 2009, "A Java-based remote GUI concept for distributed automation systems", Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on, sept., pp. 1.