Desarrollo de un prototipo para un visualizador de estructuras de un sistema operativo en ejecución a través de la comunicación serial.

Graciela De Luca, Martín Cortina, Nicanor Casas
Esteban Carnuccio, Sebastián Barillaro, Sergio Martín

Departamento de Ingeniería e investigaciones Tecnológicas
Universidad Nacional de La Matanza
Florencio Varela 1903 - San Justo, Buenos Aires, Argentina
Te. (54-11) 4480-8900
{gdeluca; mcortina; ncasas;ecarnuccio; sbarillaro}@ing.unlam.edu.ar

Resumen

En este artículo se exponen los avances relativos conseguidos durante la investigación en curso acerca del desarrollo de una interfaz de depuración remota. Si bien la construcción del visualizador será genérica, primeramente está desarrollada para el sistema operativo SODIUM[1]. Este documento se centra principalmente en las metodologías y herramientas utilizadas para desarrollar un prototipo básico sobre la interfaz de la aplicación cliente, la cual permite comunicarnos en forma remota con el S.O mencionado.

Primeramente se presenta el análisis efectuado sobre distintos lenguajes de programación y herramientas de creación de ventanas GUI con el fin de seleccionar el más adecuado para el desarrollo del prototipo visualizador. A continuación se detallan su diseño, funcionamiento e interacción con el sistema operativo SODIUM a través de dispositivos de puerto serie. Finalmente se describe como se ha implementado un protocolo de comunicación del lado del Servidor, para poder aplicarlo en la aplicación Cliente gracias al prototipo desarrollado.

Palabras clave: Interfaces GUI, Pyqt, Pyserial, Estructuras de datos, RSP, GDB, UART

Contexto

Esta Línea de Investigación es parte del proyecto “Visualización de estructuras in-ternas de un sistema operativo en ejecución como herramienta didáctica”, dependiente de la Unidad Académica del Departamento de Ingeniería e Investiga-ciones Tecnológicas perteneciente al programa de Investigaciones CyTMA2 de la Universidad Nacional de La Matanza, el cual es continuación de otros proyectos, entre ellos del proyecto SODIUM.

Introducción

1 Elección del lenguaje a utilizar

Inicialmente se evaluaron las herramientas de programación para desarrollar una aplicación del lado cliente. Los lenguajes considerados inicialmente fueron C, C++, Java, Perl y Python. Se excluyeron los lenguajes no orientados al desarrollo de aplicaciones de escritorio y sin capacidad de generación de interfaces gráficas con el usuario y se determinó que el uso de Python para desarrollar la interfaz gráfica del visualizador sería el más conveniente. Debido a que ofrece la posibilidad de crear aplicaciones multi-plataforma y posee una funcionalidad encargada de gestionar la memoria

[1]SODIUM.-Sistema Operativo del Departamento de Ingeniería de la Universidad de La Matanza

automáticamente sin necesidad de ser peticionada en forma explícita. Se descartó Java debido a que no permite variar de tipo durante la ejecución del programa sin una conversión.

Python [1] puede ser utilizado tanto para scripting como para programación orientada a objetos, posee tipos dinámicos, tipos primitivos de alto nivel como listas, diccionarios y tuplas, y una abundante biblioteca de módulos, que brindan valiosas funcionalidades como cálculo científico, manejo de ventanas, multiprocesamiento, manejo de puerto serie, funciones de red, etc.

2 Elección del framework.

Entre las bibliotecas que implementan interfaces gráficas de usuario en Python, las principales son Tkinter, WxPython y PyQT. [7]

Se descartaron las bibliotecas Tkinter y WxPython debido a la cantidad de elementos gráficos, el comportamiento de la interfaz o a la falta de un editor gráfico de ventanas en aplicaciones, multiplataforma.

Se resolvió emplear el binding de Python que posee el Framework QT, Pyqt, para la construcción de la interfaz gráfica. Esta biblioteca [3] brinda facilidades para el desarrollo de interfaces GUI, un completo conjunto de elementos gráficos y un flexi-ble y potente control del comportamiento de la interface, posee un mecanismo de conexión de señales y eventos, sencillo, rápido, de apariencia nativa, y se puede separar el diseño de la interface.

3 Elección del modelo de ciclo vida del desarrollo

El desarrollo presenta desafíos técnicos y funcionales, con especificaciones al momento volátiles, que irán siendo forjadas conforme a los resultados de diversos estudios de factibilidad, estaremos siguiendo un modelo de desarrollo incremental con prototipado desechable para el visualizador. El módulo de comunicación del extremo servidor seguirá un ciclo de vida incremental, acompañando las necesidades del visualizador.

4 Diseño y construcción del Prototipo #1

Para el uso de la interfaz serie, se encontró un módulo de bajo nivel denominado Pyserial [4], ya desarrollado con el que se consiguió efectuar la conexión y transferencia de datos entre SODIUM, funcionando como Servidor, y un programa escrito en Python, actuando como Cliente. Posteriormente se empleó el módulo Miniterm, que utiliza interna-mente Pyserial con funcionalidades de más alto nivel.

5 Funcionamiento del sistema Cliente/ Servidor Inicial

En principio se confeccionó un prototipo muy elemental de una aplicación que actúa como cliente. Actualmente la interfaz GUI desarrollada permite la visualización de cadenas de caracteres recibidas del servidor, como así también el envío de datos a través de ella. El prototipo brinda la posibilidad al usuario de detener y reanudar la ejecución de SODIUM en cualquier momento a través de botones, lo que permite observar en forma gráfica la ocurrencia de ciertos sucesos en SODIUM y empezar a controlar su ejecución. En la Figura 1 se muestra el prototipo desarrollado.

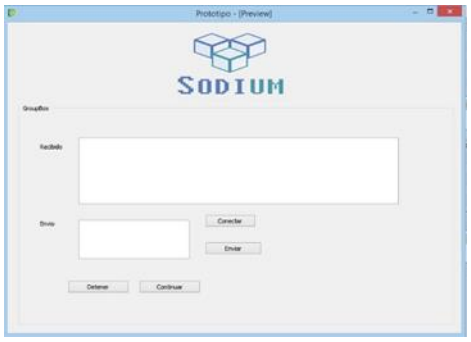

Figura 1- Interfaz gráfica del prototipo 1

En la figura 2 se pueden ver los componentes principales de la aplicación. El prototipo está conformado por dos componentes, el módulo Miniterm y la interfaz GUI. Miniterm es el encargado de efectuar la administración del puerto serie en el terminal cliente, comunicándose por medio de señales con la interfaz gráfica desarrollada en Pyqt.

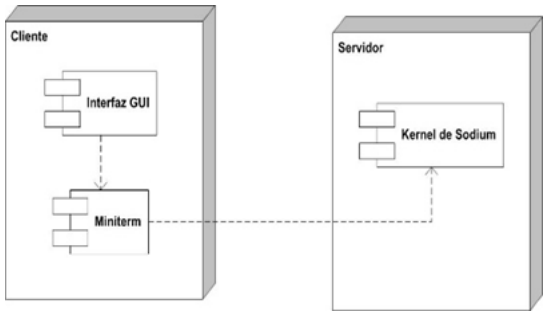

Figura2-Diagrama de Componentes del prototipo

6 Detalle de la interacción entre el cliente y el servidor

El S.O, una vez iniciado, buscará una petición de conexión del cliente. A partir de ese momento el usuario podrá interactuar con la interfaz presionando el botón "Conectar" para solicitar que se establezca la conexión.

En el instante en que SODIUM detecte una solicitud de conexión, establecerá el enlace y continuará con su ejecución normal a la espera de peticiones de datos por medio del puerto serie.

El visualizador, después que confirma el vínculo, finaliza la configuración de su UART [5] y queda a la espera de peticiones de datos de parte de la Interfaz GUI o del servidor. Para evitar que la interfaz gráfica se bloquee durante el uso del puerto serie, en el código de la aplicación cliente se genera un hilo que será responsable del manejo del mismo.

Como se puede observar en la figura 2, en la interfaz existen dos cuadros de texto. El primero con la etiqueta "Recibido", muestra las cadenas de caracteres que el S.O envía al prototipo. De esta manera le informa al usuario la ocurrencia de eventos que suceden en tiempo real. El otro cuadro de texto, con la etiqueta "envío", es el sitio donde el usuario puede ingresar caracteres ASCII para enviarlos al servidor.

7 Implementación del Protocolo RSP del lado del Servidor

En un principio se evaluó la posibilidad de incorporar en SODIUM el módulo GDB-Stub que ofrece GDB y realizar las adaptaciones necesarias para que el mismo funcione en nuestro sistema, pero resultó más simple aislar las rutinas básicas de GDB-Stub y en base a ello generar un módulo propio muy elemental, el cual denominamos "*Stub-Sodium"*. Este componente creció paulatinamente hasta implementar gran parte de la funcionalidad de GDB-Stub. Por lo tanto, tenemos una implementación parcial pero funcional del

protocolo RSP [6] en el lado Servidor, con la que podemos leer y escribir remotamente en posiciones arbitrarias de memoria, detener y reanudar la ejecución del SO, y también consultar el estado estructuras internas del mismo una vez detenido. También se implementó el envío de mensajes iniciados por el servidor que indicarán la ocurrencia de eventos al cliente, funcionalidad necesaria para implementar puntos de instrumentación.

Actualmente se está implementado el protocolo del lado del cliente, utilizando para ello el prototipo desarrollado.

8 Traducción del contenido de estructuras del sistema operativo estudiado para su representación en el visualizador

En primer lugar, cabe destacar que no siempre es de interés toda la información contenida en una estructura si el objetivo de su extracción es lograr una abstracción del detalle de la misma a través de la visualización de sus interrelaciones con otras. Llamaremos a esta abstracción del detalle, la representación canónica.

9 Representación canónica o normalizada de una estructura

Entendemos como representación canónica de una estructura de un sistema operativo a una suficientemente genérica como para describirla adecuadamente en su calidad funcional, sin tener en cuenta los detalles de implementación del sistema operativo estudiado. Esto permitirá su análisis y vuelco en los distintos diagramas que proveerá la herramienta de visualización.

A modo de ejemplo, podemos citar la estructura que representa una entrada de la tabla GDT propia de la arquitectura i386 declarada en lenguaje C que respeta estrictamente su organización en memoria. Seguidamente, vemos su potencial equivalente en lenguaje Python, donde tendremos en cuenta únicamente los datos de interés para su visualización.

C(dependiente de arquitectura i386)

```
typedef struct {
 ushort usLimiteBajo;
 ushort usBaseBajo;
 uchar ucBaseMedio;
 uchar ucAcesso;
 uint bitLimiteAlto:4;
 uint bitGranularidad:4;
 uchar usBaseAlto;
} __attribute__((packed))
stuGDTDescriptor;
```

Python (normalizada):

```
class stuGDTDescriptor:
 def __init__(self):
 self.base = 0
 self.limite = 0
 self.tipo = ''
 self.priv = 0
 self.pres=False
```

10 *Transmisión del contenido de las estructuras*

Con respecto a la necesidad de transmitir el estado de las estructuras administradas por el sistema operativo en estudio hacia la interfaz del visualizador, optamos por transmitir el contenido completo de las mismas, sin alteraciones, a través de la interfaz serie.

Esta alternativa aporta como ventaja principal la simplicidad del código del stub. El mismo debe conocer únicamente información acerca de la ubicación y sólo en algunos casos el tamaño de las estructuras del sistema y de las estructuras propias de la arquitectura del hardware ya que no debe encargarse de extraer o interpretar el contenido de las mismas. La simplicidad de este módulo no sólo aporta robustez a la solución, sino que además permite encapsular la responsabilidad de interpretar el estado interno del sistema operativo en el visualizador.

Prevemos que será necesario tener en cuenta la extremidad (término más conocido como "endianness") de los bytes recibidos por el visualizador, así como tener un conocimiento exacto de los campos de las estructuras que han sido transferidas. La extracción de los campos de las estructuras transferidas no será trivial en los casos en los que las mismas se encuentren empaquetadas, o se encuentren desalineados con respecto al ancho de las palabras del bus de direcciones.

En los casos en los que se requiera navegar por una lista enlazada de estructuras, prevemos también que el visualizador se verá obligado a solicitar un nodo por vez, extrayendo y evaluando el puntero al siguiente elemento entre cada adquisición.

Líneas de Investigación, Desarrollo e Innovación

Actualmente, nos encontramos trabajando sobre los siguientes aspectos:

-Implementación del protocolo RSP del lado de la aplicación Cliente

-Diseño y desarrollo de mecanismos a utilizar para visualizar las estructuras del sistema operativo SODIUM

-Diseño y desarrollo de las interfaces gráficas del visualizador

Formación de Recursos Humanos

El grupo de trabajo consta de dos investigadores de categoría IV, un investigador categoría V, dos investigadores con varios años de experiencia y un investigador inicial.

Resultados y objetivos

Estamos actualmente finalizando la implementación del Sodium-Stub, y ya desarrollamos los primeros prototipos del visualizador, uno de los cuales ya es capaz de conectarse a nuestro sistema operativo SODIUM, detener y reanudar su ejecución, y consultar el estado de una estructura interna.

En etapas posteriores desarrollaremos además una guía de adaptación del visualizador a otros sistemas operativos de código abierto para poder analizar el funcionamiento de los mismos y utilizar esta herramienta como elemento didáctico para la enseñanza de Sistemas Operativos.

Referencias

[1] Guido van Rossum,"*El tutorial de Python*",Editorial: Fred L. Drake Jr. *Septiembre 2009*

[2] Raúl González Duque,*"Python para todos"*,Editorial: Autoedición, Año: 2010

[3] Mark Summerfiled, *"Rapid Gui Programming with Python and QT",* Editorial: Prentice Hall, Año: 2007

[4] Chris Liechti,*"pySerial Documentation", Versión:2.6, Diciembre 2011*

[5]David S.Lawyer: "Serial HowTo" Greg Hankin

[6]Bill Gatliff "*Embedding with GNU: the GDB Remote Serial Protocol*", revista Embedded Systems Programming, Noviembre 1999

[7] "Interfaces gráficas (GUI)", http://python.org.ar/InterfacesGraficas

[8]http://www.linuxchix.org/content/courses/security/raw_sockets