XV JENUI. Barcelona, 8-10 de julio de 2009
ISBN: 978-84-692-2758-9
http://jenui2009.fib.upc.edu/

Visualización del comportamiento de llamadas al sistema POSIX sobre procesos

Miguel Riesco Albizu, M. Ángeles Díaz Fondón
Departamento de Informática
Universidad de Oviedo
c/ Calvo Sotelo, s/n. 33007 Oviedo
{albizu, fondon}@uniovi.es

Resumen

El estudio de las llamadas al sistema POSIX es una materia habitual en el programa de muchas asignaturas relacionadas con los Sistemas Operativos. Dentro de estas, las llamadas relacionadas con la creación y gestión de procesos (*fork, exec, wait, exit*) suelen ser, por su propia naturaleza, difíciles de entender para el estudiante.

Teniendo en cuenta este problema se ha se ha desarrollado un sistema que visualiza gráficamente el comportamiento de estas llamadas a partir de la ejecución de un programa que las utilice. Dicho sistema facilita tanto al profesor la explicación de esta materia y como al alumno su comprensión. En este trabajo se describe dicho sistema.

1. Introducción

En la totalidad de los planes de estudios de las titulaciones informáticas se estudia la materia de Sistemas Operativos. Esta materia puede aparecer estructurada en distintas asignaturas (S*istemas Operativos, Ampliación de Sistemas Operativos*, *Laboratorio de Sistemas Operativos,* etc.).

Sea cual sea la forma en que se cubra la materia en cada titulación, un contenido que suele aparecer frecuentemente es la programación utilizando la API POSIX. El objetivo buscado en este caso es capacitar al alumno para desarrollar programas (habitualmente en C) que utilicen los servicios que proporciona un sistema operativo con API POSIX.

En esta API hay multitud de funciones para manejar los distintos recursos del sistema operativo. En el caso particular de los procesos, el conjunto de servicios disponibles es relativamente pequeño (fundamentalmente las funciones *fork, exec, wait, exit*), pero habitualmente es difícil de entender por los alumnos.

Para facilitar la comprensión del funcionamiento de esas llamadas es habitual recurrir a la utilización de gráficos explicativos, donde cada proceso se muestra como nodo de un árbol, mientras que las aristas representan las relaciones padre-hijo que presentan los procesos en un sistema Unix. En [1] o en [5] pueden encontrarse ejemplos similares a los mostrados en la Figura 1, donde se visualizan los procesos creados durante la ejecución de un código como el que aparece en el Programa 1, además de la relación existente entre ellos.

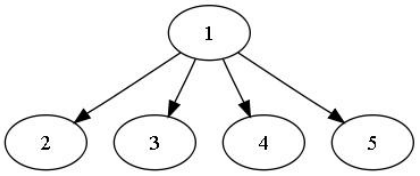

Figura 1. Ejecución de varias llamadas *fork*

Este tipo de gráficos son muy útiles para el profesor al ilustrar el proceso de creación y destrucción de procesos, con lo que el material de la asignatura (transparencias, apuntes, etc.) de este tema está repleto de ellos.

```
#include <sys/types.h>

main() {
 pid_t pid;
 int i, n=4;
   for (i=0; i<n ; i++ ) {
     pid=fork();
     if (p==0) break;
  }
}
```

Programa 1. Ejemplo de uso de *fork*

Sin embargo, los profesores de la asignatura, tras muchos años explicando esta materia, repa-

raron en varios problemas que el uso de ese tipo de gráficos comporta:

1. Los gráficos deben ser creados a mano, cosa que si bien es sencilla para programas simples puede complicarse para casos como el mostrado en el Programa 2 (puede verse su representación gráfica en la Figura 6).
2. Este tipo de gráficos son *estáticos*, mostrando los procesos creados al finalizar los procesos implicados. Sería mucho más ilustrativa la utilización de *animaciones*, donde se vea cómo se van creando y destruyendo los procesos a medida que se ejecutan los programas.
3. Sólo se muestra gráficamente el comportamiento de las llamadas *fork*, mientras que otras como *wait*, *exit* o *exec* no se consideran.

Para intentar paliar la dificultad 2 y en parte la 3, durante la explicación el profesor ha ido depurando, a lo largo de los años, una técnica consistente en la "animación" por medio de la tiza y el borrador: los procesos se dibujan en la pizarra cuando se crean y se borran cuando terminan.

Esta "técnica", además de requerir de mucho tiempo de clase y ser susceptible de errores, no ayuda en mucho a los alumnos: puede aclarar el funcionamiento en ese momento, pero dada la dificultad que conlleva plasmar esa "creación y borrado" en sus apuntes, cuando van a estudiar la materia ya no recuerdan cómo funcionan.

```
#include <sys/types.h>

main() {
 pid_t pid;
 int i, n=4;
   for (i=0; i<n ; i++ ) {
     pid=fork();
   }
}
```

Programa 2. Creación recursiva de procesos

Con el fin de solventar los problemas señalados se creó un sistema que permite, a partir de un programa en C que utilice llamadas al sistema como las descritas, generar una serie de gráficos representativos de la ejecución de los procesos a los que ese programa dé lugar.

2. Descripción de la herramienta

A continuación se describe la funcionalidad de la herramienta que se ha desarrollado, y que puede descargarse desde [3], para intentar paliar los problemas antes citados.

2.1. Servicios POSIX visualizados

El desarrollo inicial de la herramienta [4] se realizó para resolver el primer problema de los expuestos anteriormente (se intentaba automatizar la creación los gráficos explicativos), con lo que sólo se consideraba la llamada al sistema *fork*. Sin embargo, inmediatamente reparamos en que se podía ampliar fácilmente a otras llamadas. Actualmente, la herramienta visualiza el comportamiento de los siguientes servicios POSIX sobre procesos:

- *fork*: clona el proceso que realiza la llamada.
- *exec*: cambia el programa que está ejecutando el proceso (no crea un proceso nuevo, sólo cambia el código que está ejecutando el proceso).
- *wait*: espera la finalización de un proceso hijo.
- *exit*: termina la ejecución de un proceso.

El servicio *exec* aparece realmente en el sistema en forma de un conjunto de funciones (*execl, execlp, execle, execv, execve* y *execvp*), pero son funcionalmente equivalentes y sólo difieren en la forma de especificar los parámetros, por lo que nos referiremos a todas ellas durante este trabajo con el nombre genérico de *exec*.

2.2. Representación gráfica generada

Para cada uno de los servicios citados, el sistema genera un gráfico representativo de su comportamiento.

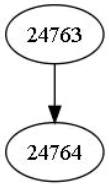

Figura 2. Representación de fork

Para el caso del *fork*, la representación gráfica generada (Figura 2) muestra dos nodos con-

teniendo el PID de los procesos relacionados mediante una arista dirigida; el origen de la misma es el proceso que ha ejecutado la función (proceso padre), mientras que el destino es el proceso creado (proceso hijo).

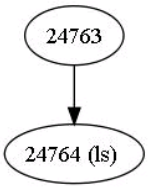

Figura 3. Representación de exec

La llamada *exec* (en cualquiera de sus formas) se representa añadiendo dentro del nodo, y entre paréntesis, el nombre del programa ejecutado, tal y como se ve en la Figura 3.

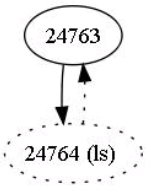

Figura 4. Representación de wait

En el caso del servicio *wait* (Figura 4) lo que se muestra es el envío de la señal SIGCHLD al finalizar el proceso hijo. Dicho envío se representa mediante una flecha discontinua desde el origen (proceso hijo) hasta el destino (proceso padre).

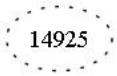

Figura 5. Representación de exit

Finalmente, la Figura 5 muestra la representación de la llamada *exit*. En este caso, en lugar de hacer desaparecer el nodo representativo del proceso del gráfico, lo mostramos mediante una línea discontinua para indicar que ya no existe.

2.3. Animación

La primera versión de la herramienta se limitaba a representar el resultado final de la ejecución de los procesos. Así, tras la ejecución del código mostrado en Programa 2, se generaba el gráfico que puede verse en Figura 6.

Esto resultó útil para el profesorado para generar los gráficos a incluir en la documentación para el estudiante, pero no ayudaba mucho para su utilización en clase ni para que los alumnos estudiaran la materia posteriormente, dado que no se reflejaba en ese tipo de gráfico a qué llamada se correspondía cada elemento ni cuál era el orden en el que se habían ido realizado cada una de ellas.

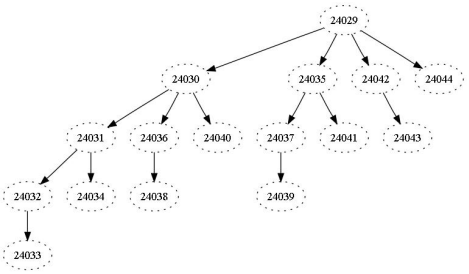

Figura 6. Ejecución del Programa 2

La actual versión del visualizador solventa ese problema mostrando cómo evoluciona la representación de la ejecución del programa a medida que se van realizando las llamadas objeto de estudio. Así, en lugar de un único gráfico final se generan tantos como llamadas al sistema de las estudiadas se ejecuten en el programa.

Veamos un ejemplo. En la Tabla 1 se reúnen los 5 pasos a los que da lugar el programa que aparece en la misma tabla (se ha simplificado la llamada *exec* en el código por legibilidad del mismo). La herramienta desarrollada mostraría en primer lugar el gráfico (a), resaltando en el código la línea que se ha ejecutado para dar lugar a esa situación (la creación del segundo proceso).

En cada uno de los pasos siguientes, se puede apreciar cómo se representa la ejecución de la llamada *exec* (b), la llamada *wait* (d) o la terminación de los procesos al ejecutar la llamada *exit* (c) y (e).

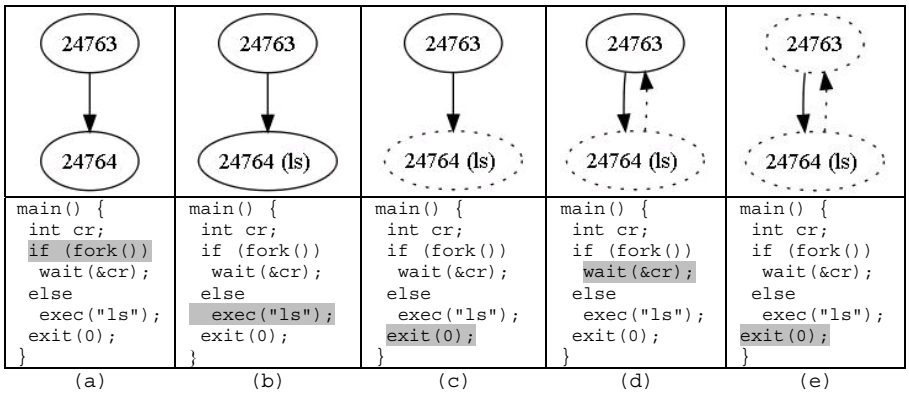

Tabla 1. Evolución de la ejecución de un programa

2.4. Utilización

La herramienta está pensada para que sea utilizada tanto por el profesor, para ilustrar sus explicaciones, como por los alumnos, para que desarrollen sus programas de prueba y puedan visualizar cómo se comportan. Teniendo esto en cuenta, la forma de trabajo actual con la herramienta es reflejo de la infraestructura existente en la que realizan las prácticas los alumnos:

1. El primer paso para visualizar el comportamiento de un programa (llamémosle *p1.c*) es la escritura del mismo. En nuestro caso utilizamos un servidor Linux donde cada alumno tiene su propia cuenta de usuario y donde creará sus programas de prueba.
2. A continuación se ejecutará el *script* "compilar", que realizará varias tareas (descritas más adelante), y dará como resultado, si no hay errores, un código ejecutable (*p1*).
3. Después el usuario realizará la ejecución del programa generado, redirigiendo la salida de errores a un fichero (pongámos *sp1*), lo que hará que, además de llevar a cabo las acciones previstas en su código, se almacene en el fichero *sp1* la información gráfica representativa de la ejecución del programa.
4. Una vez concluida la ejecución del programa, se deberá ejecutar otro script denominado "*generar_graficos*", que, a partir del fichero generado durante la ejecución del programa (*sf1*), creará los ficheros gráficos que representan cada uno de los pasos de ejecución del proceso. Esta representación podrá visualizarse posteriormente, utilizando un navegador web, accediendo a la dirección indicada al finalizar este script.

La aplicación web que visualiza la página accede a los gráficos generados y al código fuente del programa para llevar a cabo la representación gráfica de cada uno de los pasos del programa. En cada página habrá un enlace al siguiente paso de ejecución del programa (y al anterior), con lo que el usuario podrá estudiar en detalle la misma.

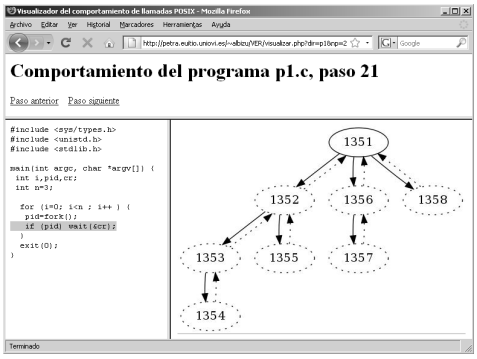

Figura 7. Página web de ejemplo

La Figura 7 muestra un ejemplo del aspecto que tiene la página, para un usuario y un programa determinado. Pueden observarse los enlaces al siguiente y al anterior paso de ejecución, por medio de los cuales se podrá ir analizando la situación de los distintos procesos tras la ejecución de alguna llamada al sistema de las contempladas en este trabajo.

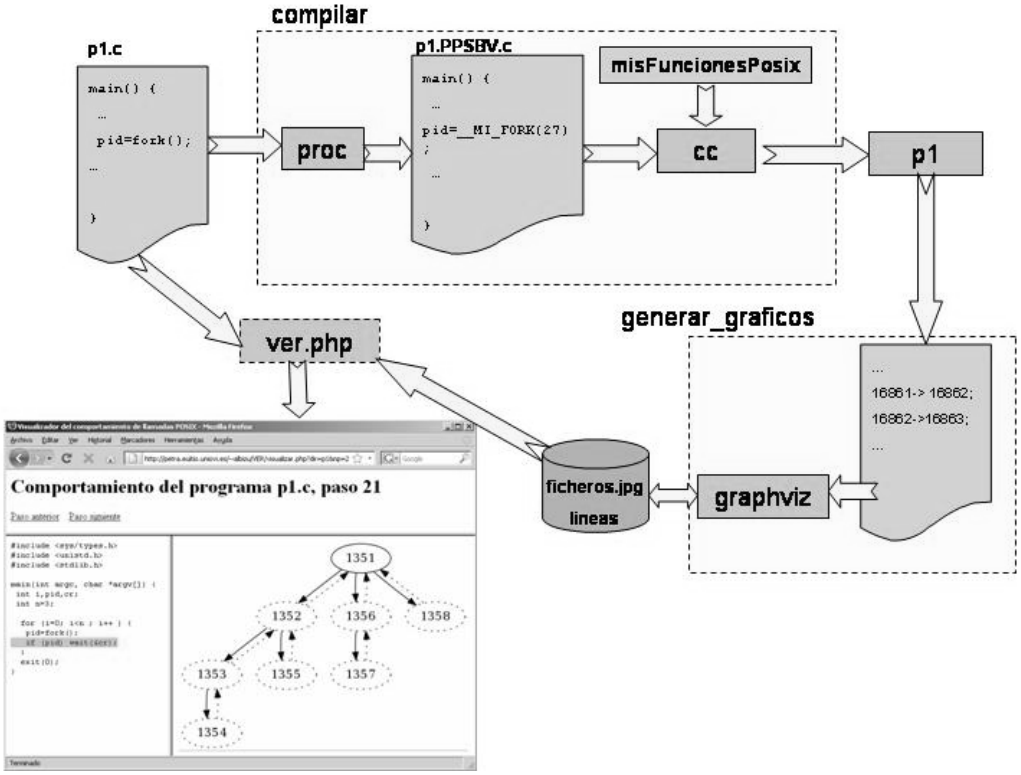

Figura 8. Implementación de la aplicación

3. Implementación de la herramienta

La implementación de la herramienta es relativamente sencilla (Figura 8). Consta básicamente de cuatro partes:

1. Una biblioteca de funciones donde se ha implementado una por cada función POSIX a estudiar. Nuestras funciones mantienen funcionalidad de la original, pero además se añade lo necesario para poder generar posteriormente la representación gráfica.
2. El script "*compilar*".
3. El script "*generar_graficos*".
4. Una pequeña aplicación web para generar y visualizar los gráficos representativos del comportamiento de las funciones estudiadas.

Seguidamente se explican con más detalle cada uno de estos módulos.

3.1. Biblioteca *misFuncionesPosix*

La idea fundamental de la herramienta radica en sustituir, dentro del programa a estudiar, cada llamada a una función POSIX sobre procesos por nuestra propia versión.

Se ha desarrollado así una biblioteca de funciones (*misFuncionesPosix*) que contiene una para cada función POSIX cuyo comportamiento queremos visualizar. Así, la biblioteca cuenta con un total de 9 funciones (*__MI_FORK, __MI_WAIT, __MI_EXIT,* etc.).

Cada una de estas funciones lo que hace es, además de realizar una llamada a la función original, generar la información necesaria para crear posteriormente el gráfico representativo de su ejecución, tal y como se explica más adelante.

Nuestras funciones añaden a los originales un parámetro adicional, que indica el número de

línea del código original donde aparecía la llamada (para poder señalar posteriormente la línea que ejecuta la llamada al sistema).

3.2. Script *compilar*

Como hemos visto anteriormente, una vez creado el programa en C que queremos estudiar se lo pasamos como parámetro a un shell script (*compilar*). La función de este script es transformar el programa original sustituyendo las llamadas a las funciones POSIX que estamos estudiando por llamadas a las funciones equivalentes de la biblioteca *misFuncionesPosix*. Así, si en el programa aparece la instrucción "`pid=fork()`" lo sustituirá por "`pid=__MI_FORK(27)`" (suponiendo que esa línea es la 27 en el código original).

El programa resultante de dichas transformaciones se compila y enlaza junto con nuestra biblioteca, obteniendo el programa ejecutable final.

3.3. Script *generar_graficos*

Durante la ejecución del programa obtenido en el paso anterior, las llamadas a las funciones de *misFuncionesPosix* habrán almacenado la información necesaria para generar los gráficos

A partir de esa información *generar_graficos* genera un conjunto de ficheros en formato jpg (según se explica más adelante) y los almacena en un directorio accesible por el servidor web (`$HOME/PPSBV/nombre`, donde `nombre` es el nombre original del programa).

Además, copia el código fuente del programa al mismo directorio para que, posteriormente, el visor web pueda mostrarlo a la vez que los gráficos representativos de su ejecución.

Una última función que realiza este programa es crear un fichero denominado "lineas" conteniendo el número de línea donde se ejecuta cada llamada al sistema.

3.4. Visor web

En el paso anterior se han almacenado en el directorio señalado los gráficos representativos de cada paso de computación y los números de línea del código fuente responsable de estado del proceso representado en cada gráfico.

El visor web accederá a dicho directorio y compondrá la página web donde se verá el código y el gráfico correspondiente a ese paso de computación.

Esta aplicación tiene como parámetros el nombre de usuario y el del programa, para poder localizar los ficheros señalados, además del número del paso de computación, que sirve para determinar qué gráfico hay que representar (que se corresponderá con la ejecución de todas las llamadas que se hayan realizado hasta ese paso de computación), pudiendo llevarse a cabo de esta manera la animación de la ejecución.

3.5. Creación de los gráficos

Para la creación de los gráficos se ha recurrido a la herramienta *graphviz* [2], disponible de manera gratuita bajo "The Common Public License".

Así, la información que generan las funciones de nuestra biblioteca son instrucciones del lenguaje *dot*. Mediante este lenguaje se define un grafo que el programa del mismo nombre (dot) se encargará de crear en forma gráfica almacenándolo en cualquiera de los muchos formatos gráficos disponibles.

Por ejemplo, para crear el grafo de la Tabla 1 (a), que representa el comportamiento de la llamada *fork*, se genera una única línea en formato dot:

```
24763 -> 24764 ;
```

Para cada llamada se genera una única línea en este formato, con lo para generar el gráfico representativo de la situación del programa en un paso de computación determinado lo único que se tiene que hacer es componer el fichero .dot correspondiente a ese paso y realizar una llamada a la utilidad dot de graphviz.

4. Utilización en clase

Las primeras versiones del programa se realizaron durante el curso pasado, precisamente durante la docencia de la práctica dedicada a la utilización de llamadas al sistema POSIX.

Entonces la herramienta estaba en un estado embrionario, por lo que sólo se pudo utilizar para generar los gráficos *estáticos* de la documentación que se entregó a los alumnos (transparencias y apuntes), así como para componer un conjunto de 8 páginas web con ejemplos *dinámicos* de ejecución de programas que utilizan las llamadas en cuestión.

Algunas de estas páginas web se utilizaron también como base para la explicación en clase, mientras que otras se dejaron para que el alumno las estudiara por su cuenta.

Durante el curso 2008-2009 además se ha dejado a disposición de los alumnos la herramienta para que puedan utilizarla para realizar sus propias pruebas que les ayuden a comprender la materia.

5. Resultados observados

No se dispone de un análisis exhaustivo del impacto que supone su utilización en el aprendizaje de la materia en cuestión. Sin embargo, sí se pueden constatar dos hechos:

1. Los profesores que han utilizado este método se han mostrado muy satisfechos, puesto que simplifica notablemente el trabajo de construir ejemplos. Asimismo, estiman que la visualización del comportamiento dinámico de los procesos facilita notablemente tanto la explicación como su comprensión por parte de los alumnos.
2. Los alumnos han accedido de forma importante a las páginas web donde se muestran los ejemplos creados por los profesores. En la primera semana hubo un total de 874 accesos a las páginas de ejemplos, cifra que consideramos importante considerando que había 72 alumnos y 8 ejemplos distintos.

En relación con el uso de la herramienta por parte de los alumnos para estudiar el comportamiento de sus propios programas, a la hora de escribir este trabajo apenas lleva una semana a disposición de los alumnos. En este plazo, sólo un 15 % de los alumnos matriculados han hecho uso de la herramienta.

En cualquier caso, consideramos que esta experiencia ha sido positiva y tenemos la intención de profundizar en la misma y aplicar estas ideas a otras partes de la asignatura.

6. Ampliaciones y mejoras

Ante la experiencia acumulada en el desarrollo y utilización de la herramienta aquí descrita, se han podido detectar aspectos mejorables de la misma, así como posibles ampliaciones.

El principal aspecto mejorable es la interfaz de uso de la aplicación, que si bien no es complicado sí se podría simplificar mediante una única interfaz gráfica que integre todo el trabajo (edición, compilación, ejecución y visualización).

Otro punto a mejorar es hacer que, en cada paso de computación, se resalte no sólo la llamada que ha producido el gráfico mostrado, sino también cuál de los procesos incluidos en el mismo ha sido el responsable de la ejecución de la llamada.

Una ampliación que sería interesante realizar es incorporar la posibilidad de trabajar con más de un programa. Actualmente sólo se visualiza el comportamiento de un único programa, aun cuando este lance otros programas por medio de la llamada *exec.* Sería deseable que, si se dispone del código fuente del programa lanzado, pudiéramos realizar también la representación de su comportamiento, cosa que ahora no se hace.

Otro apartado en el que estamos trabajando es en permitir la visualización de la evolución del programa en tiempo real, permitiendo la interacción con el programa original, desarrollando así una especie de depurador gráfico simple.

Finalmente, tenemos intención de aplicar ideas similares a otras partes de la asignatura muy susceptibles de ser ejemplificadas de esta manera. La gestión de ficheros, donde se pueda apreciar la evolución de las estructuras de datos implicadas en cada servicio o los mecanismos IPC son los campos donde más trabajo tenemos realizado.

7. Conclusiones

En este trabajo se ha presentado una herramienta para visualizar de manera gráfica el comportamiento de las llamadas al sistema POSIX sobre procesos.

Este sistema ha facilitado a los profesores la confección de ejemplos para su explicación, y se espera que permita a los alumnos analizar el comportamiento de sus propios programas de una manera más cómoda.

La experiencia llevada a cabo ha sido positiva, y se continúa trabajando en la herramienta para aumentar su funcionalidad, para hacer más

sencillo el proceso de creación de los gráficos por un lado, y para aplicar las mismas ideas en otros temas de la asignatura de Sistemas Operativos.

Referencias

[1] Félix García, Félix, Carretero, Jesús, de Miguel, Pedro y Pérez, Fernando. *Sistemas Operativos, 2/e*. McGraw-Hill Interamericana, Inc., Madrid, 2007.

[2] Página web de graphviz (última consulta: 10-2-09), `http://www.graphviz.org`

[3] Página web de la herramienta descrita: `http://156.35.81.1/PPSBV`

[4] Riesco, M., Fondón, M. D., y Álvarez, D. 2009. *Using Graphviz as a Low-cost Option to Facilitate the Understanding of Unix Process System Calls*. Electronic. Notes in Theorical. Computer. Science. 224, Enero. 2009, pag. 89-95

[5] Robbins, Kay y Robbins, Steve. *UNIX Systems Programming: Communication, Concurrency and Threads (2nd Edition)*. Pearson Education, Inc., Upper Saddle River, New Jersey, USA,2003.