# Libreria Traceback

Angelo Calvo Alfaro - Estudiante Escuela de Informatica y Telecomunicaciones

***Resumen*—El uso del stack del procesador, es una importante herramienta que es útil a la hora de fabricar herramientas que tracen caminos o efectúen métodos de debug sobre sistemas informáticos. La comprensión adecuada acerca del funcionamiento de los movimientos a través de este y la manipulación que se puede efectuar en el usando el lenguaje assembly, permite la fabricar librerías que hacen posible entre otras cosas el acceso de un lenguaje de alto nivel a leer el stack, característica que por supuesto, ellos carecen de forma nativa. El presente informe tiene por objeto describir la construcción de una librería, denominada traceback, la cual permite generar un mapa que imprime en pantalla de la traza de funciones y sus parámetros desde la función que llama a traceback, hasta el wrapper. En este contexto, se presentaran las bases teóricas, decisiones de diseño y construcción necesarias para la fabricación de la librería.**

## I. Introducción

EL procesador de una maquina de propósito general ,es el lugar donde son llevadas a cabo todas las instrucciones que debe ejecutar la maquina en cuestión. Para funcionar, el procesador hace uso de una estructura de datos llamada stack, que viene a representar una estructura del tipo LIFO, Last In First Out, que en otras palabras ejecuta los procedimientos de forma que el primero en ser procesado, es precisamente el ultimo en ser ingresado. Este mencionado Stack, es cargado dentro de la memoria de un computador para seguir un flujo y cada fragmento de procedimiento que debe ejecutar tiene una estructura definida llamada Stack Frame, el cual corresponde a un bloque con una serie de punteros hacia direcciones de memoria en donde se encuentran cada uno de los elementos necesarios para ejecutar un procedimiento que ha sido previamente programado, dentro de los cuales podemos destacar el EBP (Stack Base Pointer) que representa el espacio de memoria base en donde comienza y esta alojado un Stack Frame, el cual esta compuesto de una serie de punteros que entre otras operaciones permiten acceder a otros Stack Frame, conocer la instrucción de retorno y obtener los parámetros y variables locales a utilizar por la instrucción almacenada dentro del presente Stack Frame. Las posibilidades que entrega la compresión y adecuado uso del Stack por parte de un programador son enormes, aunque destaca con gran fuerza la capacidad de implementar trazas o recorridos de procedimientos, que son ampliamente utilizadas en el desarrollo de debugers y compiladores. A si mismo, el uso inadecuado y malicioso podría ocasionar serios problemas de seguridad que permitirán obtener datos de un programa en ejecución si se sabe la dirección de memoria EBP de uno de los Stack Frame de dicho programa. Razones como están argumentan que en los lenguajes de medio y alto nivel, la característica de acceder a dichas direcciones de memoria se encuentra restringida de forma nativa, por lo que es tarea del programador el desarrollar métodos alternativos para hacer uso de las características del Stack, haciendo uso de herramientas de bajo nivel, como lo es el lenguaje ensamblador. Debido a estas limitaciones de lenguaje, el programador debe ser capaz de generar interfaces que permitan acceder a estas direcciones de memoria para posteriormente manipularlas y utilizarlas en un lenguaje adecuado como C. En este contexto, el presente trabajo tiene por objeto el presentar el diseño y desarrollo de una librería que sea capaz de acceder al Stack del procesador, generando una traza de las funciones y parámetros que invoca un programa durante su flujo desde que es llamado por el sistema operativo. La librería que tiene por nombre traceback, debe ser capaz de entregar información en pantalla acerca del flujo de funciones que utilizo el programa que llamo a traceback, indicando los parámetros y valores de llamada. Para desarrollar esta librería, se hará uso del lenguaje C por la facilidad que tiene a la hora de utilizar punteros y efectuar casting de variables y como interfaz de conexión con el Stack se usara el Lenguaje ensamblador x86, ya que permite acceder de forma directa al Stack de funciones a través de su registro EBP y efectuar operaciones que hacen posible navegar entre Stack Frames.

Escuela de Informática y Telecomunicaciones
e-mail: angelo.calvo@mail.udp.cl.

## II. Diseño e Implementación

Para diseñar la librería traceback se deben considerar dos factores importantes. La implementación en C por si sola no es capaz de capas de generar una solución para resolver el desarrollo, debido a que C no puede acceder a los Frames de Stack de forma directa. La capacidades de C solo hacen posible moverse entre Stack teniendo como requisito mínimo un puntero a un EBP (Stack Base Pointer), el cual debe ser necesariamente proporcionado por una interfaz escrita en lenguaje ensamblador x86 de Intel. Assembly x86 provee mecanismos directos para acceder al EBP de la ultima función en ejecución a través de su puntero ESP (Stack Pointer).
La lógica de funcionamiento de un Stack según el manual de intel Assembly x86, dice que los stack frames se colocan unos encimas de otros a medida que el programa avanza en su flujo, quedando el frame mas reciente en la parte inferior de la memoria, tal y como se muestra en la figura 1. ESP corresponde al frame que llamo a la función que se esta ejecutando actualmente y se debe acceder a sus parámetros de acuerdo a sus tamaños y offset respecto al EBP. Como se muestra en la figura 2 notamos que los punteros que nos interesan para identificar las funciones de la tabla generada, están ubicados en 0( %EBP) y 4( %EBP) y ambos tienen un tamaño de 4 bytes. La operación 0( %EBP) nos permitirá recorrer los Stack Frames y por cada uno de esos Stack Frames deberemos obtener la dirección de retorno de la función que

nos indicara la siguiente instrucción a ejecutar después de la llamada a la función. Según el manual de Intel Assembly x86 los punteros a las direcciones de memoria que indican las instrucciones a ejecutar tienen como tamaño 1 byte. Al obtener el valor de la dirección de retorno se deberán ejecutar saltos de -1 byte para obtener las direcciones de memoria correspondientes a las instrucciones previas, dentro de las cuales esta la llamada a la función que deseamos identificar en la tabla.

La tabla que provee a la librería traceback posee estructuras que identifican la función a partir de la dirección de llamada de la función en cuestión, por lo que la solución deberá ser implementada en dos etapas. La primera, sera programar en lenguaje ensamblador las funciones que nos proporcionaran los Stack Frames a partir de un puntero al EBP de cada uno de ellos y los punteros a las direcciones de retorno de los Stacks Frames que luego serán iterados en busca de identificar la función en la tabla. Ademas se debe programar en Assembly x86 funciones para obtener los parámetros de cada Stack Frame a partir del offset que provee la estructura de la función identificada en la tabla de funciones y sus tipos ya que el manejo de estos resulta mas cómodo en Assembly. La segunda etapa consiste en escribir en lenguaje C rutinas recursivas que a partir de punteros EBP y punteros EIP identificaran la función a la que corresponde cada Stack Frame presente en la tabla. Una vez identificado la función a la que corresponde el Stack Frame, se procederá a imprimir sus parámetros obtenidos de las funciones escritas en Assembly y de acuerdo al formato pedido por los requerimientos.

La figura 3 muestra el flujo de funciones que utilizara la librería para determinar la traza de funciones. El diseño planteado propone que las funciones que deben ser escritas en Assembly son demarcadas con azul y las que deben ser escritas en C deben ser demarcadas con amarillo. La librería partirá por traceback que recibirá un puntero a un stack desde la función get_ebp, luego esta función entrara en un bucle el cual se mantendrá hasta que no se reconozcan mas funciones desde la tabla después de haber encontrado la función main, obteniendo el siguiente Stack Frame a partir de la funcion get_next_ebp. En este bucle se buscara a través de la función search_function, la cual obtiene el valor de retorno del Stack Frame a analizar y comienza a partir de las instrucciones previas, operación que se consigue restando 1 al puntero char que es asignado, a la dirección de retorno a buscar en un bucle a que función de la tabla corresponde. La labor de búsqueda se efectúa en la tabla con una función recursiva que recorre la tabla hasta encontrar coincidencia, en caso contrario se retornara y se pasara a analizar la siguiente instrucción.

Debido a que puede darse el caso en que jamas se encuentre una función en la tabla y por consecuente se genere un loop infinito, las búsquedas no podrán ser mas grandes que el tamaño máximo de una función. En caso de no encontrar ninguna coincidencia se retornara con un numero identificativo, que puede ser -1. Existiendo un valor de posición valido de la tabla de funciones, se procede a analizar a través de la función print_func, la cual imprimirá las funciones de acuerdo al formato solicitado.

Luego la función print_func llamara a la función print_args

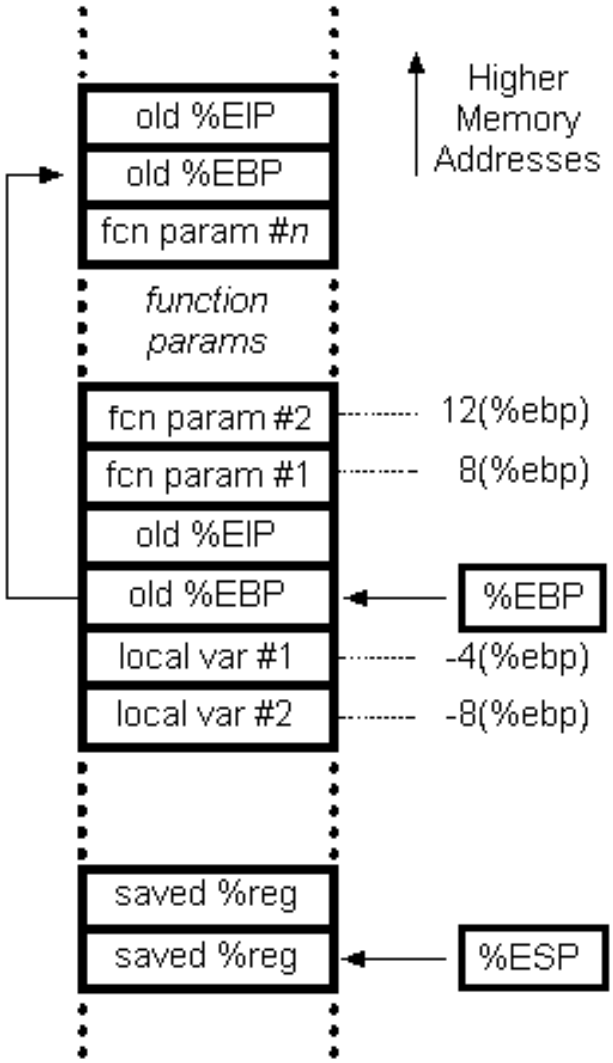


**Figura 1.** Esquema que muestra de forma grafica la representación de un stack en el hilo de ejecución del procesador.

| | |
|---|---|
| **16(%ebp)** | - third function parameter |
| **12(%ebp)** | - second function parameter |
| **8(%ebp)** | - first function parameter |
| **4(%ebp)** | - old %EIP (the function's "return address") |
| **0(%ebp)** | - old %EBP (previous function's base pointer) |
| **-4(%ebp)** | - first local variable |
| **-8(%ebp)** | - second local variable |
| **-12(%ebp)** | - third local variable |

**Figura 2.** Esquema que muestra la forma en la que esta compuesta un Stack Frame y sus punteros.

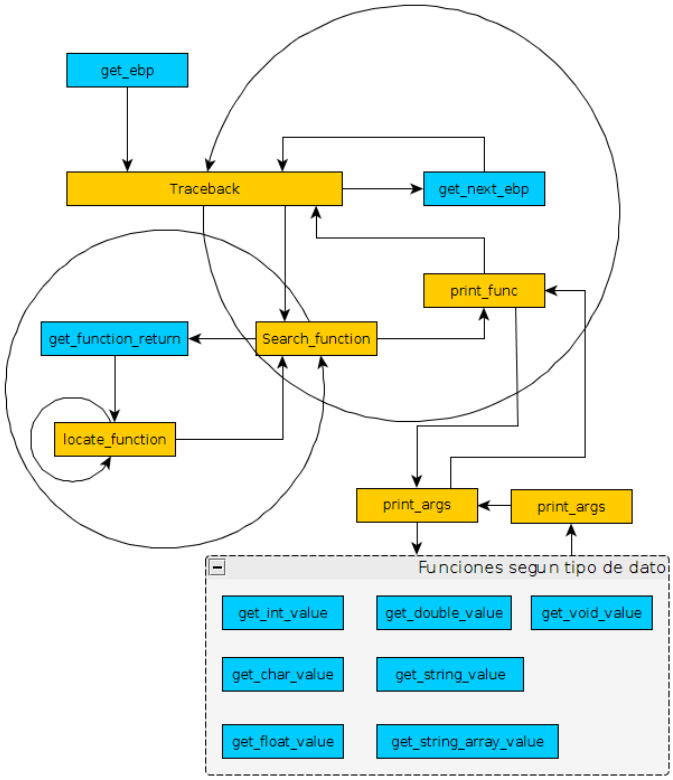


**Figura 3.** Esquema que muestra a modo explicativo el flujo de funciones que usa la librería traceback.

que hará uso de funciones programas en assembly para obtener los valores de los parámetros mencionados por la tabla de los cuales se verificara en caso de corresponder si es posible su impresión a través de la función isprint y isprint_string.
Siguiendo este diagrama de desarrollo debería llegarse a una conclusión exitosa que responda a los requerimientos. Los detalles de construcción de las funciones están documentados por doxygen y el código escrito.

## III. Desarrollo de la solución

Siguiendo el diagrama de construcción planteado previamente propuesto, el desarrollo se concluyo con algunas incidencias que fueron corregidas e iteradas durante la codificación. Por ejemplo el compilador gcc no acepto el casting explicito de punteros, arrojando errores que fueron buscados en foros especializados con el objeto de buscar una solución sin éxito, luego de esto se decidió crear una función por cada tipo de argumento en Assembly.
Otra incidencia ocurría a la hora de cortar los String con tamaño superior a 25, al efectuar el corte en los 25 caracteres y agregar los puntos suspensivos ocurría que la variable char* agregaba 2 caracteres no imprimirles. los cuales podían ser reemplazados por caracteres si se quería. Como el tamaño máximo del string debiera ser 28 que corresponde a los 25 caracteres y los 3 puntos, se busco métodos los locación de memoria para el char* mediante el método malloc y alloc, con los mismo resultados, por lo que se descubrió que el problema ocurría al agregar el ultimo punto suspensivo. Finalmente se opto por utilizar un método de concatenación de la librería string.h para concatenar los 3 puntos luego de aplicar un substring a la cadena.
Otra de las incidencias ocurridas fue al extraer numero flotantes a través de Assembly, debido a que para esta clase de numero existen operaciones especiales y deben ser tratados de una forma distinta a los números convencionales.

## IV. Pruebas propuestas

Para probar que la librería efectúa el fin deseado se diseño un archivo modificado de los test estándar que venían con el paquete. Se efectuaron modificaciones a verify_test.c con el objeto de extender sus pruebas tratando de probar todos los puntos.Es necesario mencionar que el único aspecto que no se probo porque no se puedo reproducir en testing, pero que si se encuentra implementado, es la capacidad de imprimir funciones que no se encuentran en la tabla de funciones proporcionada. Las demás funcionalidades fueron puestas a pruebas con el objeto de cumplir con lo pedido. El archivo verify_test modificado se encuentra junto con la librería entregada.

## V. Resultados esperados de la prueba

Por un análisis del código de prueba y según las especificaciones del documento de requerimientos se debería obtener en pantalla una serie de Strings que indiquen que la librería genera los resultados esperados. La coincidencia indica que efectivamente se cumple con lo esperado de la librería. Para la el archivo modificado verify_test.c se debería obtener el siguiente resultado:

function f8(int i=5, char** f="foo", "bar", Dirección de memoria, ...), in
function f7(int p=6, int g=7, int j=8, int l=8), in
function f6(char* gato=.ªbcdefghijklmnopqrstuvwxy..."), in
function f5(double array=4529999999999999...)in
function f4(int i=5, float f=35.000000), in
function f3(int p=6, char g=f, char l='\6'), in
function f2(void), in
function f1(char** array="foo", "bar", "baz", ...), in
function main(void), in
.
.
.
function (wrapper)

Luego de esto el programa no encontrara mas Stack Frame por lo que termina su ejecución con un FATAL, indicando que no existen mas funciones en el stack de llamada de la función traceback.

## VI. Resultados Obtenidos

Después de depurar el código eliminando todos las alertas de warning, lo cual genero un código limpio en C. la ejecución de verify_test arrojo los siguientes resultados que se indican en la figura 1. La ejecución de la prueba se realizo en un equipo con linux, distribución Ubuntu 14.04.

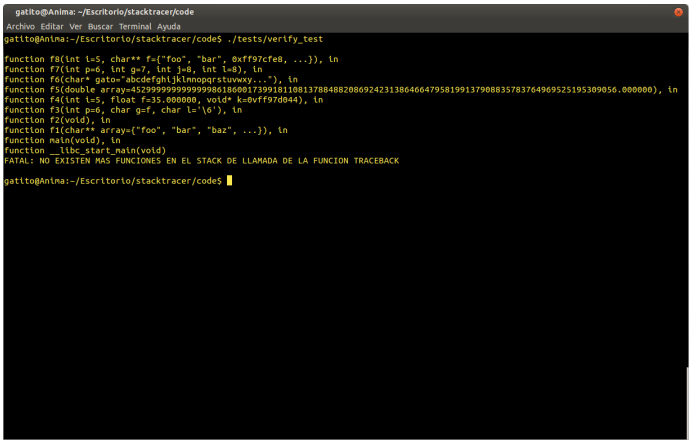


**Figura 4.** Imagen muestra los resultados obtenidos para el archivo de pruebas verify_test, se observa que los resultados obtenidos concuerdan con los resultados esperados.

La figura 4 muestra que los resultados obtenidos concuerdan con lo esperado por la ejecución de la librería. Se observa que la primera función mostrada es la que llamo a la librería traceback que corresponde a f8 y los parámetros indicados son los que se utilizaron en el código, flujo que posteriormente va mostrando uno a uno los resultados.
Los parámetros son impresos de forma correcta de acuerdo a las convenciones indicadas por los requerimientos y se sigue el flujo natural del programa hasta la función wrapper que es la encargada de llamar a main, que viene a ser la primera después de main. Luego de eso nos encontramos con que no existen mas funciones reconocibles en el Stack y la ejecución del programa termina con un FATAL. Este es el comportamiento esperado para la librería, por lo que por relaciona lógica, la librería cumple con lo solicitado y esta lista para ser integrada por otro proyecto.

## VII. Conclusiones

Del presente trabajo practico y el desarrollo de la librería traceback se pueden concluir los siguientes puntos:

1. El lenguaje de programación C es altamente eficiente a la hora de construir librerías que actúen a bajo nivel, pero en ciertos casos es necesario interfaces en Assembly x86 para efectuar operaciones que C de forma nativa no es capaz de ejecutar.
2. EL lenguaje de programación Assembly Intel x86 provee poderosas herramientas para efectuar operaciones a nivel de hardware, pero por sus limitados registros de uso general se hace necesario el uso de un lenguaje de alto nivel como C para generar de una forma mas rápida y amigable operaciones complejas. Si bien es posible realizar todo con lenguaje Assembly por un tema de comodidad es mejor efectuar una mezcla entre código Assembly x86 y C, aunque se corre el riesgo que debido a las operaciones de optimización del copilador de C se efectúen simplificaciones que el programador no tiene contempladas.
3. Los registros %EBP, %ESP y %EIP son punteros importantes a la hora de manipular Stack Frames ya que nos permiten acceder a los espacios de memoria asignados a las funciones los cuales operan a partir del puntero %EBP. Las operaciones de desplazamiento de bytes que se pueden efectuar en estos punteros nos permiten, obtener los parámetros de cada función, así como los puntos en que fueron llamadas. Esto gracias a la tabla suministrada nos permite detectar la traza de funciones que se genera.
4. Los registros de coma flotante como float y double y long double, poseen operaciones especiales en Assembly x86, por lo que al utilizar esta clase de valores, es necesario utilizar los comandos para coma flotante siguiendo la sintaxis de sufijos para las asignaciones de memoria.
5. La librería traceback tiene interesantes aplicaciones en el campo de los debuggers, ya que podríamos hacer la traza de funciones que ocurren cuando se arroja una excepción de error. Características similares ocurren en el núcleo de JAVA al ocurrir errores y se le llama stacktrace.

## VIII. Bibliografia

1. Friedl's,Steve; Intel x86 Function-call Conventions - Assembly View; http://unixwiz.net/techtips/win32-callconv-asm.html
2. University of Virginia Computer Science; x86 Assembly Guide; http://www.cs.virginia.edu/ evans/cs216/guides/x86.html
3. Using Assembly Language in Linux; http://asm.sourceforge.net/articles/linasm.html
4. Intel Corporation; Intel® 64 and IA-32 Architectures Software Developer's Manual; http://home.ifi.uio.no/griff/INF5063/IA32Doc/Intel-IA32-Arch-SW-Dev-Man-Vol2A.pdf