udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Stacktracer

Proyecto 1

Sistemas Operativos

1. Introducción

En este proyecto usted escribirá una *librería* que contiene una función llamada `traceback()`. La función `traceback` imprime la traza del stack del programa donde fue llamada. La traza del stack muestra las llamadas a funciones que fueron hechas para llegar a la posición actual del programa. Usted tendrá la información acerca de las funciones del programa y sus argumentos.

Un ejemplo para el uso de este tipo de funciones puede ser el uso en un manejador de errores de segmentación para ayudar a depurar un programa.

2. Detalles

El prototipo de la función `traceback`, está definido en `traceback.h`, y es

```
void traceback(FILE *);
```

El argumento a la función es un archivo de flujo (stream) al que la función escribirá. Para la mayoría de programas este archivo será `stderr`, pero el tomar un argumento permite una mayor flexibilidad en el uso de la función.

Está definida en `traceback_globals.c`, una tabla que contiene una entrada por cada función en el programa (en algunos casos extremos, pueden existir ejecutables que no tienen una tabla de entradas). Cada entrada en la tabla de funciones tiene un tipo `functsym_t` (definido en `traceback_internal.h`), que contiene el nombre de la función y la dirección en la que la función empieza junto con una lista de argumentos. Cada argumento está definido como un `argsym_t` y contiene el tipo del argumento y su nombre. El tipo es almacenado como un entero y puede ser emparejado con las definiciones en `traceback_internal.h`. Por simplicidad, se solicita reconocer solamente `char`, `int`, `float`, `double`, `char*`, `char**`, `void*`. Cualquier referencia subsecuente a 'string' en este documento se refiere a strings de estilo `C`.

Si la lista de funciones contiene menos entradas que `FUNCTS_MAX_NUM` ésta terminará con una función con nombre de longitud cero. Similarmente, si la lista de argumentos de una función contiene menos argumentos que `ARGS_MAX_NUM`, ésta terminará con un argumento con nombre de longitud cero. Las funciones en la lista están ordenadas por dirección.

Para cada invocación a una función, usted deberá imprimir el nombre de la función y todos sus argumentos. Cuando imprima cada argumento, usted deberá mostrar el nombre y el valor actual (cuando el tipo sea conocido). Esto significa que usted deberá imprimir el string en el caso de `char*` y algunos (ver Sección 2.1) de los strings en el caso `char**`. Tenga la precaución de que la función `traceback()` no debe de causar al programa que la llama, una terminación temprana o inesperada debido a un error de segmentación. Si el tipo del argumento es desconocido, no se espera que se imprima el valor del mismo (ver Sección 2.1).

Si se pregunta como puede obtener los valores de las funciones (nombres y argumentos) en una tabla global, normalmente no es posible dentro del framework de `C`. Cada programa de prueba que sea enlazado (link) con la librería `trackback` obtendrá el código para la función `traceback()` y una tabla de símbolos en blanco. Después de que el programa sea construido, un script de python decodificará el archivo de objetos y lo modificará para agregar en la tabla la información correcta. Nótese que ésta no es la forma correcta de obtener esta información; uno debería de obtenerla dinámicamente a través de una conversación larga y complicada con una librería (larga y confusa) que entiende como analizar los archivos ejecutables. La forma correcta, sin embargo, es más trabajo del que se pretende para este proyecto y no esta dentro del alcance del proyecto. Adicionalmente, la experiencia de aprendizaje es similar y el utilizar la librería correcta solo agrega el ejercicio de saltar trampas (trampas que están en fuego, que se mueven, y cambian de tamaño de manera aleatoria).

2.1. Salida

La función `traceback()` deberá de imprimir las llamadas a las funciones en el orden en que ocurrieron (de la más reciente a la más antigua —como un stack). La salida deberá de contener los nombres y valores de todos los argumentos (y `void` en el caso de que no existan argumentos). La salida de `traceback()` deberá ser, por ejemplo,

```
Function foo(int i=5, float f=35.000000), in
Function foobar(char c='k', char *str="test",
     char *unprintable=0xffff0000), in
Function bar(void), in
```

Esto indica que alguna función (no mostrada) llamó a `bar` sin argumentos. Luego la función `bar` llamó a

`foobar` con el carácter `k`, un string `test`, un string que no se puede imprimir (por `traceback`) localizado en la dirección de memoria `0xffff0000`. Luego, `foobar` llamó a la función `foo` con argumentos `5` y `35`. Y `foo` llamó a nuestra función `traceback`.

Si usted determina que una llamada a una función no se adecua a la convención (por ejemplo, el valor para su puntero en el stack no es un stack frame), entonces `traceback` deberá terminar. Si así lo desea, podrá emitir una linea, iniciando con `FATAL:`, la explicación al error que califique como tal. Note que esta característica no cubre el caso en el que punteros o valores ilegales aparezcan como argumentos de una función (perfectamente una función puede enviar punteros a cualquier dirección de memoria sin violar las convenciones de llamada).

Si un stack frame está bien formado pero la dirección de retorno (por ejemplo, `0x20002ab0`) indica una llamada a una función que no está en la tabla de funciones, usted deberá imprimir una linea de la forma

```
Function 0x20002ab0(...), in
```

En este caso, usted deberá mantener una traza de los stack frames después de la función (de ser posible).

Todos los argumentos deben ser impresos de la manera `tipo nombre=valor`, pero las siguientes reglas especiales deben ser aplicadas:

- Para este proyecto, caracteres imprimibles son aquellos para los que la función `isprint()` de la librería estándar retorne verdadero (ver `ctype.h`). Un string que contiene un carácter que no es imprimible es considerado como no imprimible. (Código o comentarios que contienen caracteres no imprimibles es considerado inmaduro, pero a veces entendible, pero eso no debe de afectar la salida de su función `traceback`.)

- Caracteres deben de imprimirse entre comillas simples, si son imprimibles. Si no, los caracteres deben de imprimirse (entre comillas simples) escapados como caracteres octales. Por ejemplo, si un argumento `c` contiene el carácter ASCII 'ACK', el argumento debe de imprimirse como `char c='\6'`. Esto aplica solamente para caracteres no imprimibles —ver a continuación como tratar strings no imprimibles.

- Enteros y números de punto flotante deben de imprimirse en base 10. El comportamiento por defecto de `printf()` es aceptable para `float`s y `double`s, ambos en terminos de el número de dígitos a imprimir y lo que es impreso para valores inusuales (`NaN`, y más menos infinito).

- El tipo `void*` debe de imprimirse en hexadecimal, excepto que en lugar de `0x` se debe de imprimir con el prefijo `0v`.

- Los strings se deben de imprimir entre comillas dobles.

- Los arreglos de strings se deben de mostrar en el siguiente formato {`"string1", "string2","string3"`}. Las comillas deben de agregarse alrededor de cada string a través de la función `traceback` —no son generalmente parte del string.

  La impresión de arreglos `char**` es basado en heurística. Ya que no hay forma de determinar el número de elementos en un arreglo dado el puntero base al arreglo (o aún peor, un puntero al medio del arreglo). No hay manera de imprimir en cualquier situación todos los strings válidos en un arreglo de strings sin imprimir todos los strings no válidos —a menos que algún contrato especial exista, como por ejemplo, el parámetro `int argc` en `main()`. Entonces, nosotros proveeremos una convención para este proyecto.

  Si un string en el arreglo no es imprimible, la dirección de ese string deberá ser impresa en su lugar. Sin embargo, porque es de uso común, usted deberá tratar un puntero nulo (cero) en un arreglo de strings no como un "string no imprimible", sino como una indicación de que el arreglo a llegado a su fin antes de llamar a un puntero nulo. Por otra parte, si aparece que un arreglo de strings contiene cuatro o más strings, solo los primeros tres deben de imprimirse, seguido por '`...`'. Por ejemplo, {`"string1", "string2","string3", "string4"`} debe de imprimirse como {`"string1", "string2","string3", ...`}. Los strings no imprimibles cuentan para el número total de elementos en el arreglo (i.e., usted debe de mirar los primeros cuatro elementos del arreglo sin importar que).

- Si un string imprimible tiene más de 25 caracteres, solo los primeros 25 deben de imprimirse seguidos por un '`...`' (e.g., "`esta cadena tiene más de 25 caracteres`" deberá de imprimirse como "`esta cadena tiene más de ...`", —note que el espacio en blanco es el vigésimo quinto carácter).

- Como una regla general, algo que no pueda tener sus valores impresos (por cualquier razón) debe de tener su dirección impresa en hexadecimal, excepto si la situación está cubierta por alguna regla anterior (e.g., un carácter válido que tiene un valor que es no imprimible). Si parte de un string `char*` es imprimible y otra parte no lo es, entonces el string completo no es imprimible. Un arreglo de strings con uno o más strings no

imprimibles dentro de él es considerado imprimible mientras el arreglo de string sea considerado como válido.

- Cualquier cosa de un tipo desconocido debe de mostrarse como un tipo `UNKNOWN` y tratado como si fuera una constante no imprimible, es decir, con la dirección en hexadecimal.

- Note que estas descripciones no son una sugerencia. La salida de su programa se analizará automáticamente por un algoritmo, si se confunde en alguna regla su programa fallará en esta parte de la evaluación.

3. Objetivos

- Escribir código limpio en `C`. A muchas personas les gusta el lenguaje `C` ya que da libertad al programador (punteros, casting, etc.). Sin embargo, es muy sencillo el tropezar con nuestros pies. Desarrollar (y mantener) un sistema consistente de definición de variables, comentarios, y separación de la funcionalidad es esencial.

  Note que debe de apegarse a `C`, y no `C++`. El escribir un kernel en `C++` es probablemente más difícil de lo que parece, porque se tiene que implementar versiones seguras a hilos (o al menos consciente de interrupciones) de `new` y `delete`. Adicionalmente, usted deberá implementar otras piezas del código de ejecución de `C++`. De tal manera que el iniciar a familiarizarse con `C` es una buena forma de iniciar los proyectos.

- Escribir pseudocódigo (o diseñar antes de programar). Para la programación de sistemas es muy importante el pensar estructuras de datos cruciales y algoritmos antes de tiempo, ya que se convierten en primitivas para el resto del programa.

- Comentar. Es importante el incluir comentarios para que alguien más pueda ver y mantener el código tan rápidamente como quien lo escribió sin tener que ver los detalles. Debe de comentarse el código utilizando doxygen, que es similar a javadoc pero para `C`.

- Utilizar herramientas comunes de desarrollo, e.g., `gcc` o `ld`.

4. Instrucciones

4.1. Trabajo a realizar

1. Este proyecto se desarrollará en parejas.

2. Lean detalladamente el código proporcionado con este enunciado, incluyendo el `makefile`. Muchas de las respuestas a las preguntas comunes están en el código fuente.

3. Probablemente, se necesitará alguna información que no está disponible en el framework de `C`. Entonces, usted necesitará el utilizar una pieza de código o dos de assembler `x86`. Se recomienda encarecidamente el hacer esto a través de una función de `C` que se llame en un archivo `.S` (note la `S` en mayúscula) en lugar de utilizar las facilidades en linea `asm()`. Cualquiera de las dos formas puede funcionar, pero en la práctica es mucho más sencillo el escribir código con `asm()` que funcione con una versión del programa o una versión en particular del compilador, pero que dejará de funcionar misteriosamente. Adicionalmente, el poblar el código de `C` con llamadas `asm()` lo hace extremadamente difícil de portar de una plataforma a otra.

   El código de soporte incluye un archivo de muestra `.S` (`add_one.S`), y usted puede encontrar el detalle de la función `asm()` en la sección Assembler Instructions with C expression Operands de la documentación de `gcc`. Si aún desea utilizar `asm()`, note que debe utilizar la versión "complicada" para que el programa sea correcto. Así su programa le comunicará su intención al compilador.

   En términos de hacer que `make` construya los archivos `.S`, note que éstos (`.S`) son isomorfos a los archivos `.c` en el sentido que `make` contiene las reglas por defecto para poder construirlos en `.o`.

4. Es importante que lea cada programa de prueba que se le provee y que entienda que es lo que hace, y que es lo que la implementación de su función `traceback()` debe de imprimir cuando se ejecute. **Note que su programa debe de pasar las pruebas que se realizarán en otro computador** —no donde usted lo desarrolló.

   Es importante que su función `traceback` pueda manejar cualquier tipo de programa en que alguien quiera utilizarla. Usted debe asegurarse de que trabaje adecuadamente sin importar donde sea llamada dentro de cualquier programa, y asegurarse que la función `traceback` no dañe la correcta operación del programa después que retorne. Note que la función `traceback` está diseñada como una función para depurar y no todo el tiempo será capaz de imprimir un stack bien formado con 100 % argumentos válidos, de tal forma, nunca deberá de destruir la ejecución del programa o entrar en un ciclo infinito.

   Recuerde que algunas funciones de `C` son inse-

guras en algunos casos, por ejemplo, `sprintf()`, `fprintf()`, y `printf()`. Tome un momento para considerar que puede utilizar y en que momento, sus implicaciones y considere alternativas.

Se provee una script de verificación sencillo que asegura que la salida sea igual a lo que se solicita —ver `verify` en el `makefile`.

5. Comentar es una parte importante de escribir código. Para esto, utilice doxygen para documentar su código, y se recomienda el seguir la guía de doxygen en el sitio del curso sobre como documentar.

   Revise los comentarios existentes en los archivos que se le entregan, por ejemplo `traceback_internal.h`, para ver que tipo de documentación se espera. Adicionalmente, el `makefile` contiene una regla para la generación de código `make html_doc` que generará el código en html.

4.2. Entregables

El código fuente que generen deberá de tener la siguiente estructura:

- `codigo`: carpeta que contiene todos los archivos fuente que definen su proyecto.
- `informe`: el código fuente para generar el informe que presenta.

Estas carpetas deberán de ser comprimidas en un solo archivo, y subido al sitio del curso junto con un `PDF` del informe, y un `PDF` de la documentación del código generada en Doxygen (ver Sección 4.5 sobre el uso de Doxygen en su código).

4.3. Fecha de entrega

La fecha de entrega es el día 07 de abril. En horario de clase deberán de entregar el informe impreso. Y deberán subir los archivos al sitio del curso antes de la entrega impresa. Noten que la entrega digital de los documentos cierra al inicio de la hora de la clase, por lo que deberán subir sus archivos antes de la clase.

Se les recomienda el subir los archivos antes de la hora límite para evitar problemas con el servidor. No se considerarán correos que se envíen horas antes de la fecha límite de la entrega con problemas subiendo los documentos.

Se les recuerda nuevamente que el curso tiene una política de *tolerancia cero* para las entregas tardías.

4.4. Sobre el informe

- Deben preparar un informe de **máximo 4 páginas** de contenido (éstas no incluyen las figuras o referencias) utilizando `IEEEtran.cls` y la guía entregada en clase.
- El informe debe de contener al menos:
  - Resumen
  - Introducción (explicando la idea general de `traceback`, e introduciendo las tecnologías que utilizó para su solución)
  - Explicación del diseño e implementación de la función `traceback`. Explique decisiones de diseño, y compromisos al realizar la función. No es un copy-paste del código, sino una explicación de la función y su diseño y desarrollo. Para revisar el código está la documentación que entrega generada por Doxygen.
  - Discusión de los experimentos realizados. Explique los resultados de las distintas pruebas que realizó y como se aseguró de que el código hace lo que se supone que hace.
  - Conclusión
- **Si se detecta plagio ustedes obtendrán un uno en el proyecto**, y se dará aviso a las autoridades correspondientes para que se tomen las sanciones del caso.

4.5. Notas de interés y restricciones

- Su informe debe de estar realizado siguiendo la guía de documentos técnicos discutida en clase. Recuerde que se penalizará cada falta de ortografía y de redacción con 1 % de la nota final hasta un 20 %.
- No se aceptarán entregas fuera de la hora de entrega. Esto incluye copia digital fuera del plazo, o copia impresa fuera del horario de clase. Además, no se considerará una entrega completa sino existe la entrega digital y la física.
- Para evitar problemas al subir sus archivos (por ejemplo, problemas con el tamaño de la entrega, o disponibilidad del servidor) se les recomienda el subir los archivos antes de la hora de entrega. Noten que la hora de entrega no es la hora para empezar a subir los archivos, sino un límite para tener un punto de corte. Programen su entrega y háganla antes de la fecha límite.
- En caso se sobrepase el límite de páginas estipulado, se evaluará lo presentado hasta el límite

solicitado. Para evitar problemas, sigan las instrucciones y sean concisos.

- El código que generen debe de estar documentado utilizando Doxygen. Se recomienda el leer la guía sobre puntos a tener en cuenta al documentar código en el sitio del curso.
- Al documentar en Doxygen deben de utilizar la opción `JAVADOC`, y las opciones afines para la documentación del código.
- Noten que Doxygen permite exportar la documentación a varios formatos, si le es más fácil el evaluar con otro formato mientras desarrolla no hay ningún problema, siempre y cuando el documento final esté generado en LaTeX y se entregue en `PDF`.
- El código fuente que entreguen debe de contener solo archivos que no son generables. Es decir, incluyan archivos fuente de código (`.cpp`, `.h`, `.py`, `.tex`, etc.), pero no incluyan librerías que se generan (`.o`, `.so`, `.dll`), ejecutables, archivos auxiliares, o similares.
- Se recomienda que utilicen un manejador de versiones para mantener su código (por ejemplo, git).
- Usted debe de hacerse responsable de que su código compile, enlace (link), y se ejecute correctamente en cualquier ambiente (se calificará en un ambiente Unix).
- No cambie ningún archivo, excepto por `traceback.c` y `config.mk`. El cambiar `config.mk` le permitirá el hacer cualquier cambio necesario para la compilación de la librería `traceback` y cualquier caso de prueba.
- Usted puede considerar el caso en el que `traceback` se ejecute en un programa multi hilo. Esto es muy difícil, sino imposible, de resolver y presenta muchos problemas. Así que no se preocupe de ello. Los programas en los que se evaluará su librería son casos restringidos en los que se llamará a la función `traceback` en un hilo a la vez.

5. Tips

- Note que un error de segmentación no necesariamente matará su programa. ¿Qué causa un error de segmentación? ¿Cómo reacciona el kernel de Unix? ¿Qué control sobre esa secuencia de eventos se tiene? Estudie la librería de `C` y su documentación para entender las opciones que tiene. Además, encuentre que funciones son interrumpibles y que funciones son llamadas después de que la función ha sido interrumpida. Alguna información sobre este tipo de funciones está disponible en Open Group documentation of `sigaction()`.
- Si estudia cuidadosamente la documentación de las llamadas del sistema (relacionadas con VM), por ejemplo, `mmap`, `mprotect`, y `msync`, usted puede encontrar una forma de (ab)usar de una o más de ellas para su beneficio. Algunas de estas funciones tienen comportamientos no documentados, así que se le sugiere el evaluar cuidadosamente su uso para estar seguro que hacen lo que se supone que hacen.
- También es posible que utilice la llamada de sistema `write`, y otras llamadas relacionadas (nuevamente, no todo funciona de la manera en que lo espera –así que haga pruebas extensivas).
- La documentación del pseudo-sistema de archivos `proc` puede ser de ayuda.

Tabla 1: Ponderación del proyecto.

	Informe	**Práctica**	**Subtotal**
Resumen	2		2
Introducción	4		4
Diseño `traceback`	15	5	20
Explicación de los compromisos de la función	15	5	20
Funcionamiento de casos de prueba		40	40
Conclusiones	4		4
Doxygen		10	10
Total	40	60	100

6. Ponderación

El proyecto está ponderado 60 % de práctica y 40 % del informe (esto incluye escritura, estructuración de ideas, redacción, etc.) que será evaluado a través del informe escrito, de la documentación entregada en doxygen, y del código entregado. La evaluación será general y se evaluarán los aspectos teóricos, prácticos, y la presentación (a través del informe) en conjunto. El detalle de la ponderación está en la Tabla 1.