Práctica 1: Estructura del stack en dos sistemas.

Diseño de Sistemas Operativos

Pablo Castro Valiño (pablo.castro1@udc.es)

Curso 2012-2013

Índice

1. Preguntas.

1. *¿Qué es el stack?*
   Es una estructura de pila organizada en frames que almacena variables, valores argumento y dirección de retorno del proceso al que pertenece.

2. *¿Por qué estás seguro de que estás mostrando el stack?*
   Porque visualizo el stack a partir del acceso a través de una variable local, la cual al pasarme de rango en los campos de la variable me permite observar el resto de objetos que hay en el mismo stack como variables o direcciones de retorno.

3. *¿Qué es la dirección de retorno?*
   Es la posición de memoria a la que una función, terminada su ejecución, debe saltar para continuar el flujo del programa.

4. *¿Por qué estás seguro de la identificación de la dirección de retorno?*
   Porque tiene un número dentro del rango de direcciones del proceso, se mantiene constante con cada llamada recursiva de la función y al ser modificada podemos, mediante un proceso empírico, verificar que se puede retornar a la instrucción siguiente a la que debería, como se muestra en el fichero **salto_espacial.c** (4).

5. *¿Qué es la dirección de la frame anterior?*
   La dirección del frame anterior es la que contiene la dirección en donde se encuentran las variables, argumentos, etc. de la función desde la que se ha llamado a la función actual y a la que pertenece el frame actual.

6. *¿Por qué estás seguro de la identificación de la dirección del frame anterior?*
   Porque dicha dirección apunta a una posición dentro del stack que a su vez apunta a otra y asi sucesivamente y que todos se encuentran colocados en las mismas posiciones en el stack. Al igual que con la dirección de retorno, también podemos comprobar que podemos saltarnos un frame, esta vez mediante un salto temporal con el fichero **salto_temporal.c** (5).

7. *¿Por qué estás seguro de la identificación de las variables argumento?*
   Se ubican después del lugar donde se almacena la dirección de retorno y es sencillo distinguirlas cuando realizamos cambios en ellas, volvemos a ejecutar el programa y observamos dichos cambios reflejados en ellas.

Si ejecutamos el fichero **test1-programa_basico.c** (2) se ve como aparecen reservadas varias posiciones, la primera corresponde a la única variable argumento y las 2 siguientes están guardadas por si introducimos mas variables argumento.

Podemos utilizar el fichero **test2-programa_argumentos.c** (3) para identificar los argumentos, viendo como cambia la 1º variable argumento y como se mantienen las 2 siguientes que no son modificadas.

8. *¿Por qué estás seguro de la identificación de las variables locales?*
   Igual que en anteriores ocasiones, basamos nuestro juicio en la prueba empírica mediante el fichero **test1-programa_basico.c** (2) en el que al realizar cambios en las variables, podemos ver como estas cambian también en el stack.

   También hay que mencionar que la pila deja 2 espacios no utilizados entre las variables locales y la dirección del frame anterior.

9. *¿Por qué estas seguro de la identificación del principio y fin de cada frame del stack?*
   Viendo los valores que se guardan en el stack podemos observar como las variables locales están al principio, por lo tanto, si vemos de abajo hacia arriba, el frame siguiente al ultimo impreso por pantalla termina donde empiezan las variables locales del ultimo.

10. *¿Por qué estás seguro del orden en el que se colocan los componentes de cada frame del stack?*
    Porque conocemos la estructura del stack y por lo tanto podemos reconocer los componentes que lo forman en los diversos frames.

    Incluso a traves de pruebas podemos ver como, modificando variables en el programa, se producen dichos cambios en los frames, por ejemplo la variable n en **test1-programa_basico.c** (2) que se reduce en 1 para cada llamada recursiva.

11. *¿Qué es un salto espacial?*
    Es un salto en el código. Mediante la modificación de la dirección de retorno, se salta a una linea de código distinta a la que debería llevarnos el flujo del programa.

    Un ejemplo de *salto espacial* es el realizado en el fichero **salto_espacial.c** (4).

12. *¿Qué es un salto temporal?*
    Es un salto en el tiempo. Mediante la modificación de la posición del

frame anterior, hacemos que apunte a otro, saltándonos de esta forma llamadas recursivas de la función.

Un ejemplo de *salto temporal* es el realizado en el fichero **salto_temporal.c** (5).

13. *¿Cómo se realiza un salto por acceso al stack?*
Se realiza mediante la modificación de la dirección de retorno.

14. *¿Cómo se obtienen los valores correspondientes para un salto cualquiera?*
Si lo que pretendemos es realizar un *salto temporal*, necesitamos saber la posición del stack en la que se encuentra la dirección del frame anterior para poder modificarlo. También es imprescindible saber el tamaño del propio stack para sumarselo, después de haberlo multiplicado por 4 (tamaño de int), y así saltar un frame.

    En el caso de un *salto espacial* necesitamos saber la posición en el stack de la dirección de retorno de la función para sumarle la distancia que querramos saltar. En nuestro caso, queremos saltar una instrucción y por lo tanto usamos la función **distancia_salto.c** (6) para calcular la distancia que luego aplicamos en el salto.

15. *¿Por qué estás seguro de que se ha saltado correctamente?*
Porque una vez realizado el salto, el programa continua ejecutándose en la posición que nosotros buscábamos. La prueba se ha realizado con el fichero **salto_espacial.c** (4) en el que se consiguió saltar una instrucción al regresar desde la ejecución de una función.

16. *Realiza el salto a otra posicion espacial.*
Véase la función **salto_espacial.c** (4)

17. *Realiza el salto a otra posición temporal.*
Véase la función **salto_temporal.c** (5)

2. Procedimiento para la realización de la práctica.

2.1. Análisis de la estructura y del contenido del stack.

Para la realización de la práctica se ha seguido como guión las preguntas proporcionadas en clase, por lo tanto primeramente se han creado los ficheros **test1-programa_basico.c** (2) y **test2-programa_argumentos.c** (3) con la intención de conocer la estructura y organización de la pila y que nos ha devuelto como resultados los contenidos en los ficheros **stack_linux.txt** (7) y **stack_windows.txt** (8), para Linux y Windows respectivamente.

2.2. Realización de un salto espacial.

A continuación siguiendo el guión entregado, hemos realizado el salto espacial que hemos recogido en el archivo **salto_espacial.c** (4), para el cual también hemos necesitado **distancia_salto.c** (6) con el que calculamos el número de instrucciones en ensamblador que debemos saltar para poder evitar la instrucción *x++;*.

2.3. Realización de un salto temporal.

Finalmente se ha realizado el salto temporal tras haber calculado manualmente lo que ocupaba cada frame del buffer, y manteniendo un variable global que incrementamos en 1 cada vez que se regresa de una recursión. El valor de la variable en una ejecución normal debería ser 4 pero al realizar el salto entre buffers el valor es 3.

3. Índice de archivos adjuntos.

3.1. Parte obligatoria.

1. **Makefile** Se ha introducido un fichero *Makefile* para agilizar la compilación.

   - **make all** Compila todos los ficheros.
   - **make tests** Compila los ficheros de tests para comprobar la estructura del stack.
   - **make espacial** Compila el fichero que realiza el salto espacial.
   - **make temporal** Compila el fichero que realiza el salto temporal.
   - **make clean** Elimina todos los ficheros ejecutables.

2. **test1-programa_basico.c**
   Programa básico para testear la colocación y la evolución de las variables locales que llama recursivamente a una funcion y muestra el stack.

3. **test2-programa_argumentos.c**
   Programa para testear la posición y la evolucion de los argumentos de una función y que llama recursivamente a dicha funcion mostrando el stack.

4. **salto_espacial.c**
   Programa que llama recursivamente a una funcion, muestra el stack y finalmente salta una instruccion al modificar el valor de retorno de dicha función.

   Utilizando el archivo *distancia_salto.c* (6) calculamos la distancia que debemos saltar y la sumamos a la dirección de retorno para que cuando la función regrese, se salte la instrucción 'x++' e imprima por pantalla 'x = 0' en lugar de 'x = 1'

5. **salto_temporal.c**
   Programa que llama recursivamente a una funcion, muestra el stack y realiza un salto temporal entre frames.

   Utilizamos una variable global llamada 'test' inicializada a 0 que suma 1 en cada vuelta de la recursión, al ser 'n = 3', habrá 3 llamadas recursivas mas la ejecución normal de la función por lo tanto si todo fuese correctamente, 'test' debería valer 4 pero como se realiza un salto temporal, valdrá finalmente 3.

6. **distancia_salto.c**
   Programa que calcula la distancia para la que se debe realizar el salto.

   Se etiqueta con 'a' la instrucción 'x++', con 'b' el 'printf...' y con la diferencia de sus direcciones de memoria se calcula la distancia a saltar.

7. **stack_linux.txt**
   Fichero que contiene el stack del programa *test2-programa_ argumentos.c* (3) ejecutado en **linux**.

8. **stack_windows.txt**
   Fichero que contiene el stack del programa *test2-programa_ argumentos.c* (3) ejecutado en **windows**.