Universidad Rey Juan Carlos

Arquitectura de Computadores/Arquitectura de Sistemas Audiovisuales II

Práctica 5: Reservar memoria dinámica en tiempo de ejecución

Katia Leal Algara

INTRODUCCIÓN:

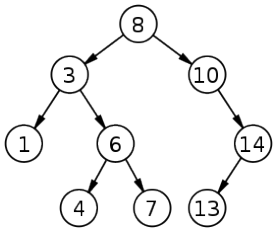
- Representar estructuras
- Reserva dinámica de memoria

A. Treesort: algoritmo de ordenación

El *treesort* es un algoritmo de ordenación que construye un árbol binario de búsqueda. El interés de los árboles binarios de búsqueda radica en que su recorrido *inorder* (en este caso se trata primero el subárbol izquierdo, después el nodo actual y por último el subárbol derecho) proporciona los elementos ordenados de forma ascendente y en que la búsqueda de algún elemento suele ser muy eficiente.

Un árbol binario no vacío, de raíz R, es un árbol binario de búsqueda sí:

- En caso de tener subárbol izquierdo, la raíz R debe ser mayor que el valor máximo almacenado en el subárbol izquierdo, y que el subárbol izquierdo sea un árbol binario de búsqueda.
- En caso de tener subárbol derecho, la raíz R debe ser menor que el valor mínimo almacenado en el subárbol derecho, y que el subárbol derecho sea un árbol binario de búsqueda.



Se pide implementar el programa **treesort.asm** que lee una lista de números introducidos por el usuario para posteriormente imprimirlos en orden ascendente. Se lee un número por línea, utilizando la llamada al sistema encargada de leer un entero, hasta que el usuario introduce el **número 0** para indicar el final de la entrada de datos.

En C, nuestra estructura de datos para representar un árbol sería la siguiente:

```
typedef struct _tree_t {
       int val;              /* the value of this node. */
       struct _tree_t *left; /* pointer to the left child. */
       struct _tree_t *right; /* pointer to the right child. */
} tree_t;
```

Un NULL en los campos *left* o *right* indicará que el nodo no tiene rama izquierda o derecha.

Para representar en la arquitectura MIPS esta estructura nos harán falta 3 palabras (12 bytes):

- una palabra para representar el valor
- una palabra para almacenar el puntero a la rama izquierda
- una palabra para almacenar el puntero a la rama derecha

Por lo tanto, necesitamos un trozo de memoria de tamaño 12 bytes para representar un nodo del árbol.

```
lw $s0, 0($t1) # a = foo->val;
lw $s1, 4($t1) # b = foo->left;
lw $s2, 8($t1) # c = foo->right;

sw $s0, 0($t1) # foo->val = a;
sw $s1, 4($t1) # foo->left = b;
sw $s2, 8($t1) # foo->right = c;
```

Ahora que sabemos cómo representar los nodos para poder formar un árbol, necesitamos la llamada al sistema sbrk para reservar memoria de forma dinámica para cada uno de dichos nodos. El único inconveniente de esta llamada al sistema es que solo nos permite reservar memoria, no liberarla.

Service	Code in $v0	Arguments	Result
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory

Para facilitar la implementación del programa, a continuación de incluye un pseudocódigo.

```
# Sugerencia de utilización de los registros
# $s0 – Nodo raíz del árbol
# $s1 – Siguiente número introducido por el usuario
# $s2 – Valor centinela (número 0)
main:
        …
        # Paso 1: crear el nodo raíz
        # root = tree_node_create ($s2, 0, 0);
        ……
        jal tree_node_create
        …
        # Paso 2: leer números e insertarlos en el árbol hasta leer el 0
in_loop:
        …
        jal tree_insert  # tree_insert (number, root);
        …
end_in:
        # Paso 3: imprimir los subárboles izquierdo y derecho
        …
        jal tree_print # tree_print(left_tree);
        …
        jal tree_print # tree_print(right_tree);
        …
# end main

# tree_node_create (val, left, right): crear un nuevo nodo con el valor indicado y
# con los punteros a los subárboles y izquierdo y derecho indicados
tree_node_create:
        # Crear pila
        # Invocar sbrk syscall
        # Comprobar si queda memoria
        # Liberar pila
        # Retornar
# end tree_node_create

# tree_insert (val, root): crea un nuevo nodo y lo inserta en el árbol de forma
# ordenada
tree_insert:
        # Crear pila
        # Crear un nuevo nodo, tree_node_create (val, 0, 0);
        jal tree_node_create
```

```
        …
        # Insertar el nuevo nodo creado implementando el siguiente algoritmo en C
        # for (;;)
        # {
        #       root_val = root->val;
        #       if (val <= root_val) // Recorrer subárbol izquierdo
        #       {
        #               ptr = root->left;
        #               if (ptr != NULL)
        #               {
        #                       root = ptr;
        #                       continue;
        #               }
        #               else
        #               {
        #                       root->left = new_node;
        #                       break;
        #               }
        #       }
        #       else // Recorrer subárbol derecho
        #       {
                        // Igual que para el subárbol izquierdo
        #       }


        # Liberar pila

        # Retornar

# end tree_insert

# tree_print(tree): recorre el árbol de forma inorder imprimiendo el
# valor de cada nodo. Código C equivalente:
# void tree_print (tree_t *tree)
# {
#       if (tree != NULL)
#       {
#               tree_print (tree->left);
#               printf ("%d\n", tree->val);
#               tree_print (tree->right);
#       }
# }
tree_print:

        # Crear pila

        …

        jal tree_print # Recursivo por la izquierda

        …

        jal tree_print # Recursivo por la derecha

        …

        # Liberar pila

        # Retornar

# end tree_print
```