Universidad Simón Bolívar
Departamento de Computación y tecnología de la Información
Organización del Computador - CI3815
Prof. Yudith Cardinales y Kitty Alvarez

Informe del Proyecto 1
Manejador de Memoria y Lista Enlazada

Autores:
Luis Pino 15-11138
Arturo Yepez 15-11551

Sartenejas, Abril 2019

Resumen

El objetivo de este proyecto es el manejo de memoria de forma dinámica utilizando un manejador, que a su vez se utilizará para la implementación de una lista simplemente enlazada para probar su correctitud y desarrollar el uso de estructuras de datos en MIPS.

Tabla de Mensajes de error

Para la función Perror se definió una tabla de errores para cada una de las funciones del manejador, las cuales se ven reflejadas de la siguiente forma

Número	*Error*	*Descripción*
-1	ErrorInit1	La cantidad de memoria que solicitó el usuario es menor a 1 byte, el cual es la cota inferior del manejador
-2	ErrorInit2	La cantidad de memoria que solicitó el usuario es mayor a 1000 bytes, el cual es la cota superior del manejador
-3	ErrorMalloc1	No está disponible la cantidad de memoria solicitada de forma continua
-4	ErrorFree1	El bloque de memoria a eliminar no tiene contenido

Funciones Implementadas

1-. Funciones del TAD_Manejador

Para esta función se utilizaran 2 arreglos en memoria, uno llamado memory, que representa los espacio de memoria que podrán ser manejados por el usuario y otro arreglo llamado ref_list. Es un arreglo referencial que solo tiene -1 y 1, para representar bytes vacíos y en uso. Esta última tendrá ocupados los espacios que el usuario reserve y libres los que el usuario libere además de los que vienen libres por inicialización.

Obligatorias:

1. *Init:* En esta función utilizando la cantidad de bytes dadas por el usuario, se procede a inicializar cada uno de los arreglos con -1 en cada uno de los bytes solciitados por el usuario para representar que están reservados, pero no en uso. Se realiza en ambos arreglos porque estos deben cambiarse de forma consecutiva porque representan lo mismo. Si el numero de bytes es menor a 1 que es a cota inferiro del manejador, se devuelve el error “ErrorInit1”. Si el numero de bytes que pide el usuario es mayor a 1000 que es la cota superior de nuestro manejador, se devuelve el error “ErrorInit2”.

2. *malloc:* En esta función se verifica primero en la ref_list si hay espacio libre, utilizando un ciclo. Después de conseguirlo, se reserva el espacio en este arreglo y se guarda la primera dirección reservada en el, para luego ir a memory a esa misma dirección y reservar los bytes solicitados de forma contigua. Si no se puede por falta de espacio, se envía el error “ErrorMalloc1”.

3. *free:* Busca la dirección solicitada para eliminar en memory, contando la cantidad de bytes que hay previos, para luego teniendo ese numero, ir a ref list e identificar mediante un ciclo el tamaño del bloque a eliminar y además lo libera en este arreglo. Luego de tenerlo, se devuelve a memory y libera el espacio que se identificó en Ref_list. Si el espacio de memoria que dió el usuario está vacío, se procede al error “ErrorFree1”.

4. *perror:* A está función llegan los codigos de error y se verifica si cumple con alguno de los errores que corresponden a la tabla superior. De no encontrarlos, esta función no hace nada

Auxiliares:

1. *print_string: Impmir strings*
2. *print_integer: Imprimir enteros*
3. *newline: Imprime una nueva linea*
4. *SendToPerror_init1: Envía el error 1 del init a perror*
5. *SendToPerror_init2: Envía el error 2 del init a perror*
6. *malloc_linear_search: Verifica que haya espacio para reservar*
7. *sendtoperror_malloc1: Envía el error 1 de malloc a perror*
8. *sendtoperror_free1: Envía el error 1 de free a perror*
9. *freespacecounter: Calcula el tamaño del bloque a eliminar por free*
10. *free_memory: Libera el espacio ocupado en memoria*
11. *free_reference: Libera el espacio en el arreglod e referencia*
12. *error_init1: Imprime el mensaje de rror del codigo de error -1*
13. *error_init2: Imprime el mensaje de rror del codigo de error -2*
14. *error_malloc1: Imprime el mensaje de rror del codigo de error -3*
15. *error_free1: Imprime el mensaje de rror del codigo de error -4*

2-. Funciones de la lista enlazada

Para la creación de las listas, en principio depende del $sp para guardar las posiciones de todos los nodos y del Head por seguridad, aunque directamente estos luego de creados no dependen de esos valores de $sp. Sobre la creación de nodos al igual que el Head de la lista se crean reservando una cantidad de espacio con syscall 9 de forma que no pueda ser sobreescrito ese espacio: Los nodos tienen 2 palabras de espacio, una para el key del nodo y la otra para la dirección del siguiente nodo; mientras que el Head tiene 3 palabras, una para la dirección de memoria reservada del primer nodo, la segunda palabra para el mismo caso en el último nodo y la última palabra contiene la cantidad de elementos que posee la lista.

En general, el proposito de la lista es poder estructurar las direcciones de espacio libre reservado y el peor de los casos poder prescindir de ellas a la hora de “liberar” esa memoria, para eso la necesidad de la implementación de una lista simplemente enlazada.

Obligatorias:

1. *Cabeza_Lista:* Es la creación de un “nodo” especial que contendra las direcciones del primer y último nodo, aparte de la cantidad de elementos o nodos que contiene la lista; luego reserva la dirección del nodo en una palabra reservada llamada Head. No amerita ningún parametro puesto que funciona como un constructor de la lista, y por lo tanto no amerita guardar ningún valor en $sp para su futura recuperación.

2. *Nodo:* Esta función es la encargada de crear los nodos o elementos que contenga la lista, su creación consiste en el uso de 5 sub-funciones que son las encargadas de poder tomar en cuenta los 3 posibles casos en la creación de un nodo: cuando el nodo es el primero o último de la lista, que corresponde a los casos bordes o el caso de que sea un nodo regular. La diferencia de los casos bordes radica en que debe al final incluir sus direcciones en la respectiva posición del Head.

Creamos el nodo, primero que nada reservando el espacio en memoria para evitar sobreescrituras con 2 palabras, una dedicada para el key y la otra para la dirección del siguiente nodo, se guarda la respuesta en el key del nodo en la dirección reservada, inmediatamente se incrementa el size del Head sumando un elemento. Lo último es que mediante comparaciones elige que tipo de nodo es (primero, último o regular) para enviarlo a su función especial que solo se encargara de decidir si guardar memoria o no, y si lo hace, en donde.

3. *delete:* Se encarga de, si es necesario eliminar un nodo mediante el uso de un número que corresponderá a la posición del nodo que se desea eliminar. El único parametro que se pasa, es como se menciona, el número que corresponde al nodo que se desea eliminar.

4. *print_List:* Es la función encargada de imprimir al usuario los elementos de la lista en pantalla, su implementación consiste en un loop que evalúa su es un caso borde o caso regular y va imprimiendo nodo por nodo con un espacio de por medio hasta llegar al final de lista.