Trabajo práctico

Programación I
1er cuatrimestre 2023

Condiciones

El trabajo práctico se implementará en grupos de 2 o 3 personas.
El entregable del trabajo consistirá de:

- *Documentación describiendo la solución.*
- *Código fuente con la implementación de la resolución al problema.*
- *La entrega será a través del sistema de control de versiones Git de la plataforma GitHub.*

La evaluación para determinar la aprobación del trabajo serán:

- *Correcta presentación de código en el sistema requerido (Git).*
- *Pasar exitosamente el caso de prueba planteado en el enunciado.*
- *Explicación de la solución planteada en un coloquio grupal.*

Descripción del problema

Un *garbage collector* (recolector de basura) es un agente que administra un bloque de memoria para un conjunto de programas que necesitan ejecutar sus algoritmos. Su funcionamiento se describe en los párrafos que siguen:

- Al arrancar el sistema se define la cantidad de memoria que administrará el sistema y se inicializa la estructura con punteros nulos.
- Una vez en ejecución, el sistema recibe pedidos de memoria por parte de los clientes. Este pedido puede venir acompañado de una cantidad de memoria necesaria y un texto con la firma del proceso.
    - En caso de haber disponibilidad de memoria a reservar, se devuelve un identificador del puntero,que usará el programa para operar.
    - Los identificadores se guardan en una variable de tipo `int`.
    - En caso de no haber disponibilidad debe devolver un identificador inválido.
- Durante la ejecución del sistema, un cliente con un bloque reservado puede requerir ampliar la memoria. En este caso se intentará aumentar el espacio reservado y se devolverá OK (cero) cuando sea posible y ERROR (-1) cuando no se pueda.
- Cada vez que el cliente asigna el identificador a otra variable, el compilador se lo informa al *garbage collector* mediante la función `add_reference`, que aumenta un contador de referencias a un bloque.
- Cada vez que una variable termina su tiempo de vida, el compilador se lo informa al *garbage collector* mediante la función `remove_reference` y el *garbage collector* decrementa un contador de referencias.
- El sistema mantiene el estado de las referencias y cuando algún contador de referencias llega a 0 (cero), el *garbage collector* debe liberar la memoria.
- El administrador ofrece información del estado de la memoria a través de las funciones:
    - Cantidad de memoria en uso (vs. cantidad de memoria restante).
    - Cantidad de bloques prestados, tamaños y cantidad de referencias vivas.
    - Listado de procesos (identificados con su firma) y cuanta memoria tienen reservada.
- Al terminar la vida del sistema, el *garbage collector* deberá liberar toda la memoria tomada por los clientes.

Se pide: implementar la interfaz de funciones dada y ejecutar el caso de pruebas propuesto en el enunciado. El sistema no deberá generar pérdidas de memoria del sistema.

Interfaz de funciones (API)

```
int init_gc(int max_mem);
int new_block(int sz,char name[]);
int resize(int block, int sz);
int add_reference(int block);
int remove_reference(int block);
int used_memory();
int available_memory();
void destroy_agent();
```

Caso de prueba

El sistema implementado deberá ejecutar el siguiente caso de prueba.

```
#include <stdio.h>
#include <string.h>
#include “garbage.h”

int main() {
    int max_mem = 1000;
    int block1, block2, block3, block4;

    // Prueba de init_gc
    init_gc(max_mem);

    // Prueba de new_block
    block1 = new_block(200, "Block 1");
    block2 = new_block(300, "Block 2");
    block3 = new_block(150, "Block 3");
    block4 = new_block(400, "Block 4");

    // Prueba de resize
    resize(block2, 400);
    resize(block4, 250);

    // Prueba de add_reference
    add_reference(block1);
    add_reference(block2);
    add_reference(block2); // Agregar dos referencias al mismo bloque
    add_reference(block3);

    // Prueba de remove_reference
    remove_reference(block2);
    remove_reference(block2); // Quitar una referencia más al mismo bloque

    // Prueba de used_memory
    int used_mem = used_memory();
    printf("Used memory: %d\n", used_mem);

    // Prueba de available_memory
    int available_mem = available_memory();
    printf("Available memory: %d\n", available_mem);

    // Prueba de destroy_agent
    destroy_agent();

    return 0;
}
```

```
//
//  Programación 1 - Trabajo pràctico: Implementaciòn de un garbage
collector.
//
//  Declaración de interfaz de funciones del garbage collector.


// Contantes de valores de retorno de funciones
#define OK         0
#define ERROR   (-1)


//Inicializacion del GC. Se indica cantidad de memoria.
int init_gc(int max_mem);


//Creacion de un nuevo bloque de memoria.
int new_block(int sz,char* name);


//Redimensionamiento de memoria.
int resize(int block, int sz);


//Se agrega una referencia a un bloque de memoria existente.
int add_reference(int block);


//Se elimina una referencia a un bloque de memoria.
int remove_reference(int block);


//Calcula cuanta memoria queda
int cur_used_memory(void);


//Calcula cuanta memoria queda disponible.
int cur_available_memory(void);


//Destructor del GC.
int destroy_agent();
```