Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Tabla de contenidos

6.1. Actividades

6.1.1. Gestión de memoria en C y en Java

Recursos

- Para esta actividad se utilizará la siguiente tabla:

Aspecto	C	Java
Llamada para obtener memoria		
La reserva de memoria precisa tamaño		
Más de una función disponible para reserva/creación		
¿Se inicializa la memoria reservada?		
Llamada para destruir/liberar memoria		
Llamada para cambiar el tamaño de un bloque previamente reservado		
¿Es posible acceder fuera de los límites de un bloque reservado?		
¿Qué sucede cuando el puntero a un bloque de memoria reservado se pierde (no está almacenado ya en ninguna variable)?		

Plan de trabajo

En grupos de cuatro personas y durante cinco minutos, completar el cuadro anterior consensuando las respuestas.

Evaluación

Las respuestas de cada grupo se ponen en común con el resto de la clase para rellenar una tabla común.

6.1.2. Las funciones para gestionar memoria en C

Plan de trabajo

1. Resuelve los trece primeros problemas. Si tras resolverlos tienes dudas sobre el funcionamiento

de la gestión de memoria acude a consultas con el profesor. Si ves que los conceptos los tienes claros, resuelve los problemas restantes.

6.1.3. Fugas de memoria en C

Plan de trabajo

1. Resuelve los cuatro problemas que se plantean en el segundo documento. Asegúrate de que asistes a la sesión magistral con ellos bien entendidos.

Evaluación

En la siguiente clase se utilizarán estos conceptos para resolver un problema de gestión de memoria, por lo que necesitarás las soluciones para responder correctamente a las preguntas.

Autoevaluación automática

Comprueba con estas preguntas si has entendido este documento

Indique los errores que encuentra en cada trozo de código.

1.

```
1  int count_element(int id, struct list *inicio)
2  {
3    int cont=0;
4    struct list *aux= (struct list *) malloc (sizeof(struct list));
5    aux=inicio;
6    while (aux!=NULL)
7    {
8     cont++;
9     aux=aux->next;
10   }
11   return cont;
12 }
```

- No hay errores.
- Línea 5: hay una fuga de memoria de 1 elemento de tamaño `sizeof(struct list)`.
- Línea 8: uso de memoria sin inicializar.
- Línea 9: hay una fuga de memoria. Se pierde toda la lista.

2.

```
1  int count_element(int id, struct list **inicio)
2  {
3    int cont=0;
4    struct list *aux;
5    aux=*inicio;
6    while (*inicio!=NULL)
7    {
8     cont++;
```

```
 8      cont++;
 9      *inicio=(*inicio)->next;
10    }
11    return cont;
12  }
```

- No hay errores.
- Línea 5: hay una fuga de memoria de 1 elemento de tamaño `sizeof(struct list)`.
- Línea 8: uso de memoria sin inicializar.
- Línea 9: hay una fuga de memoria. Se pierde toda la lista.

3.

```
 1  int count_element(int id, struct list *inicio)
 2  {
 3    int cont;
 4    struct list *aux;
 5    aux=inicio;
 6    while (aux!=NULL)
 7    {
 8     cont++;
 9     aux=aux->next;
10    }
11    return cont;
12  }
```

- No hay errores.
- Línea 5: hay una fuga de memoria de 1 elemento de tamaño `sizeof(struct list)`.
- Línea 8: uso de memoria sin inicializar.
- Línea 9: hay una fuga de memoria. Se pierde toda la lista.

4.

```
 1  struct list *del_element(int id, struct list *inicio)
 2  {
 3    struct list *aux,*ant;
 4    aux=inicio;
 5    ant=inicio;
 6    while ((aux!=NULL)&&(aux->id!=id))
 7    {
 8     ant=aux;
 9     aux=aux->next;
10    }
11    if (aux == NULL)
12      return inicio;
13    if (aux == ant)
14    {
15      free(aux);
16      inicio=inicio->next;
```

```
17    }
18    else
19    {
20      ant->next=aux->next;
21      free(aux);
22    }
23    return inicio;
24  }
```

- No hay errores
- Línea 16: acceso a memoria con un puntero corrupto.
- Línea 20: fuga de memoria.
- Línea 13: memoria sin inicializar.

5.

```
1   struct list *del_element(int id, struct list *inicio)
2   {
3     struct list *aux,*ant;
4     aux=inicio;
5
6     while ((aux!=NULL)&&(aux->id!=id))
7     {
8      ant=aux;
9      aux=aux->next;
10    }
11    if (aux == NULL)
12      return inicio;
13    if (aux == ant)
14    {
15      inicio=inicio->next;
16      free(aux);
17    }
18    else
19    {
20      ant->next=aux->next;
21      free(aux);
22    }
23    return inicio;
24  }
```

- No hay errores.
- Línea 15: acceso a memoria con un puntero corrupto.
- Línea 20: fuga de memoria.
- Línea 13: memoria sin inicializar.

6.

```
1   struct list *del element(int id, struct list *inicio)
```

```
2  {
3    struct list *aux,*ant;
4    aux=inicio;
5    ant=inicio;
6    while ((aux!=NULL)&&(aux->id!=id))
7    {
8     ant=aux;
9     aux=aux->next;
10   }
11   if (aux == NULL)
12     return inicio;
13   if (aux == ant)
14   {
15     inicio=inicio->next;
16     free(aux);
17   }
18   else
19   {
20     ant->next=aux->next;
21   }
22   return inicio;
23 }
```

- No hay errores.
- Línea 15: acceso a memoria con un puntero corrupto.
- Línea 20: fuga de memoria.
- Línea 13: memoria sin inicializar.

7.

```
1  struct  vector *create_vector(int number)
2  {
3  if (number == 0)
4    return NULL;
5  struct vector *nuevo= (struct vector *) malloc(sizeof(struct vector)*number);
6  if (nuevo == NULL)
7    return NULL;
8  int i=0;
9  while (i≤number)
10 {
11   nuevo[i]=i;
12   i++;
13 }
14 return nuevo;
15 }
```

- No hay errores.
- Línea 9: memoria sin inicializar.
- Línea 11: sobre-escritura de memoria dinámica.

8.

```
1  struct  vector *create_vector(int number)
2  {
3  if (number == 0)
4    return NULL;
5  struct vector *nuevo= (struct vector *) malloc(sizeof(struct vector)*number);
6  if (nuevo == NULL)
7    return NULL;
8  int i;
9  while (i<number)
10 {
11   nuevo[i]=i;
12   i++;
13 }
14 return nuevo;
15 }
```

- No hay errores.
- Línea 9: memoria sin inicializar.
- Línea 11: sobre-escritura de memoria dinámica.

6.1.4. Gestión de conjuntos de palabras

Un programa en C necesita manipular un conjunto de palabras. Se nos encarga la tarea de implementar esta funcionalidad y para ello se nos ofrece la siguiente descripción de requisitos:

- La aplicación puede manipular varios conjuntos separados de palabras.
- El orden y la forma en que se almacenan las palabras en la estructura de datos no es relevante para el resto de la aplicación.
- Las operaciones que se deben soportar son:
  1. Obtener un conjunto vacío.
  2. Dada una palabra, obtener un conjunto con esa única palabra.
  3. Dado un conjunto y una palabra, obtener el nuevo conjunto que contenga la palabra dada.
  4. Dado un conjunto y una palabra, devolver un conjunto que no contenga la palabra dada. Si la palabra no existe, el conjunto no se modifica
  5. Destruir un conjunto de palabras dado.
- Todas las operaciones que reciben una palabra y la almacenan, deben crear un duplicado con la llamada al sistema `strdup`. Esta función recibe una cadena de tipo `char *` terminada por un byte igual a cero y devuelve una cadena (de tipo `char *`) que es un duplicado del parámetro en una porción de memoria reservada con `malloc`.
- La solución propuesta debe ser compacta, es decir, tener un uso de memoria lo más reducido posible.

1. En grupos de cuatro personas, discutir durante cinco minutos la estructura de datos que se propone para manipular estos conjuntos de cadenas de texto. Escribir su definición en C. Comentar la solución propuesta con el resto de la clase. Derivar una propuesta final sobre la que trabajarán todos los grupos.
2. El equipo propone una implementación de la función que devuelve un conjunto vacío y la que devuelve un conjunto con una única palabra. Comprobar la solución propuesta con el profesor.
3. El equipo propone una implementación para la función que inserta una palabra en un conjunto. Prestar especial atención al tipo de datos que debe devolver esta función.
4. Escribir las definiciones de las cabeceras de las dos funciones restantes (la que borra un elemento y la que destruye un conjunto).
5. Supongamos que ya tenemos escritas todas las funciones, ahora el resto de la aplicación las quiere usar, pero tal y como se incluye en la especificación, no se requiere saber cómo están almacenados los datos internamente. ¿Cómo organizarías el código y las definiciones para conseguir esto?
6. Implementar una de las dos funciones restantes.