Proyecto 3 de Sistemas Operativos.
Manejo de Memoria

Cecilia Hernández

May 30, 2017

Fecha inicio: Martes, 30 de Mayo, 2017.
Fecha entrega: Martes, 13 de Junio, 2017 (a mediodia).

1 Introducción

En este proyecto implementará un manejador de memoria muy simple tipo malloc de C. Para ello deberá solicitar memoria al sistema operativo usando la llamada a sistema **sbrk()**.

2 Objetivos

- Entender los mecanismos de asignación de memoria usando llamadas a sistema en linux.
- Fomentar en los estudiantes el desarrollo de habilidades en programación en C.
- Potenciar la presentación de ideas en forma escrita y oral.

3 Descripción

Cada proceso tiene su propio espacio de direccionamiento el cual se traduce a memoria física en forma dinámica por el HW (MMU en CPU) y Sistema Operativo. La figura 1 muestra el espacio de direccionamiento de un proceso en memoria virtual, donde se muestra los distintos segmentos lógicos que contiene. Un segmento importante es el Heap, donde se maneja la memoria dinámica. En particular se muestra un punto con una flecha denominado **brk**. Este punto se conoce como el inicio del heap. El heap es un espacio virtual continuo de memoria con tres puntos importantes. El primero es el punto de partida (base), un máximo (límite) (manejado con las llamadas a sistema getrlimit y setrlimit) y un punto adicional llamado **break**. El break marca el final de la memoria mapeada con memoria física. Para poder implementar un manejador de memoria se debe obtener el punto de partida, para ello usaremos las llamada a sistemas **brk** y **sbrk**.

```
\\ para moverse en el heap un cierto espacio
void *p;
p = sbrk (0); // para obtener punto de partida
if ( sbrk (size) == (void *) -1)
      return NULL;
return p;

\\para regresar al punto de partida de nuevo.
brk(p)
```

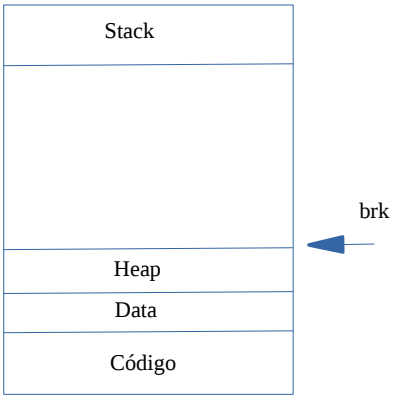

Figure 1: Espacio de direccionamiento de un proceso.

Su labor consiste en implementar la asignación de memoria considerando como base lo que vimos como manejo de espacio libre. Sin embargo acá debe mantener en la lista los bloques con espacio libre y bloques con espacio ocupado. Las interfaces que debe implementar son las siguientes.

- void *malloc(int);
- void free(void *);

Su implemetación debe considerar administrar la memoria usando las ideas de spliting y merge discutidas en clases para manejo de espacio libre. Además su implementación debe considerar una lista doblemente enlazada para simplificar las operaciones y la estructura del bloque dada. Para el magic number solo considere dos valores, uno para los bloques asignados y otro distinto para los libres, osea para los ocupados $0x12345678$ y para los ocupados $0x77777777$.

```
typedef struct mblock memblock;

struct mblock {
  int size;
  memblock *next;
  memblock *prev;
  void *mip;
  int free;
  int magic;
};
```

Ademas debe implementar los criterios de asignación como “First fit” y el “Next fit” según visto en clases.

Su aplicación debe permitir analizar el heap para requerimeintos de memoria de malloc y free por línea de comando y despliege de la lista de administración de memoria.

4 Metodología

El proyecto debe ser desarrollado en C o C++. Trabajo en grupo de 2 o 3 alumnos.

5 Evaluación

La evaluación final consiste en las siguientes partes:

- Demo. Completitud, funcionamiento, calidad código e interrogación. (70 %)
- Informe. Presentación, redacción, análisis. (30 %)