.UBA fiuba

FACULTAD DE INGENIERÍA

Memoria Dinámica en C

Parte 1 - Conceptos

Algoritmos y Programación I - Fundamentos de Programación

rev1.0

Diego Serra - Gustavo Bianchi

Asignación de memoria

La asignación de memoria es un proceso mediante el cual a los programas informáticos se les asigna espacio de memoria física o virtual.

La **asignación** de memoria se realiza **antes o en el momento de la ejecución del programa**.

Hay **dos tipos de asignaciones** de memoria:

➔ Asignación de **memoria estática** o en **tiempo de compilación**

➔ Asignación de **memoria dinámica** o en **tiempo de ejecución**

Memoria Estática vs Dinámica

Principales Diferencias

Memoria Estática	Memoria Dinámica
➔ Una vez asignada la memoria, **el tamaño no puede cambiar**.	➔ Luego de asignada la memoria, **el tamaño se puede cambiar**.
➔ En este esquema de asignación de memoria, **no podemos reutilizar** la memoria no utilizada.	➔ Esto **permite reutiliza**r la memoria. El usuario **puede asignar más memoria** cuando sea necesario. Además, el usuario puede **liberar la memoria** cuando la necesite.
➔ **La ejecución es más rápida** que la asignación de memoria dinámica.	➔ **La ejecución es más lenta** que la asignación de memoria estática.

Diagrama de memoria de un proceso

La cantidad máxima de memoria que puede ser gestionada, depende de la arquitectura del procesador.

En un **procesador de 32 bits** se destinarán un máximo de **4 GB** de memoria, ya que es posible direccionar hasta 2^{32} espacios de memoria.

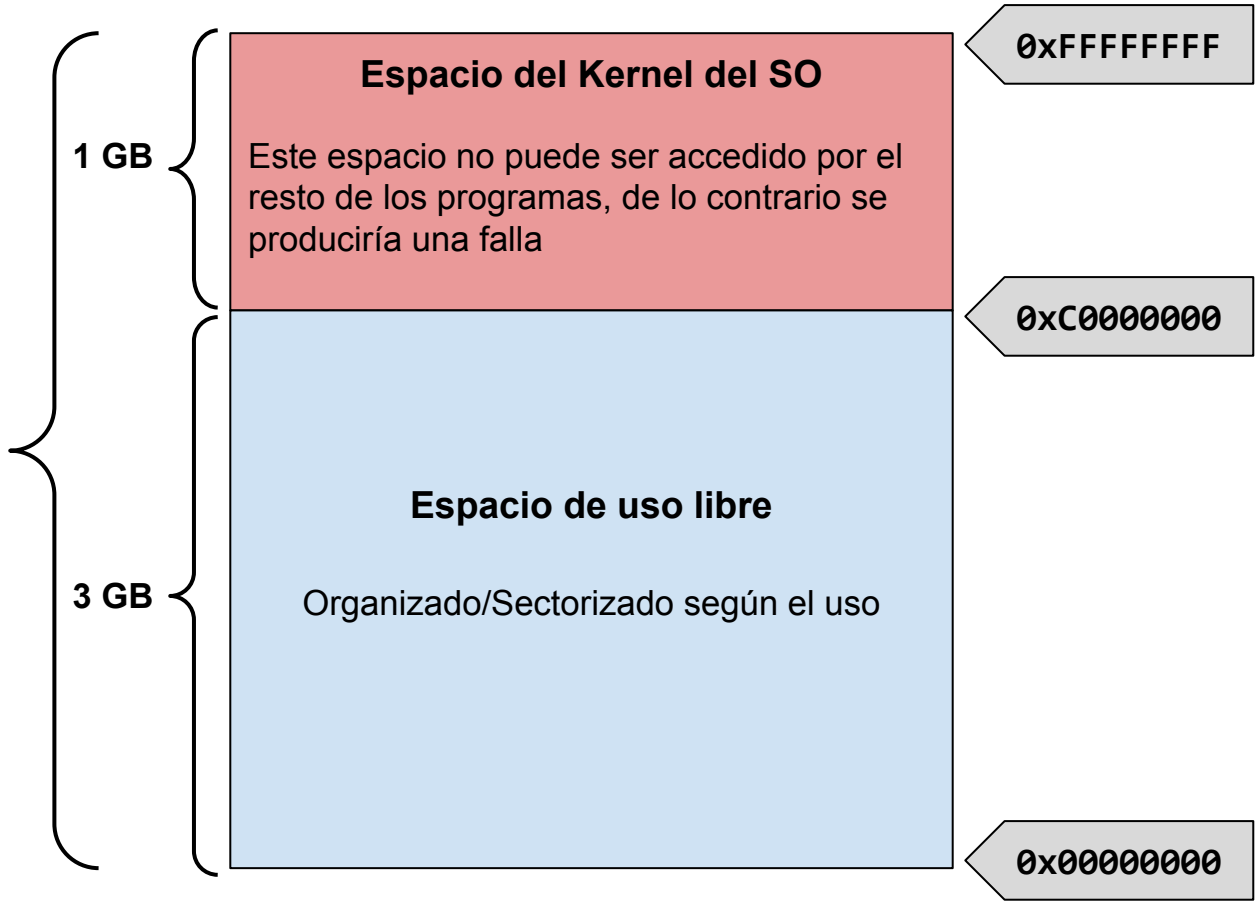

Diagrama de memoria de un proceso

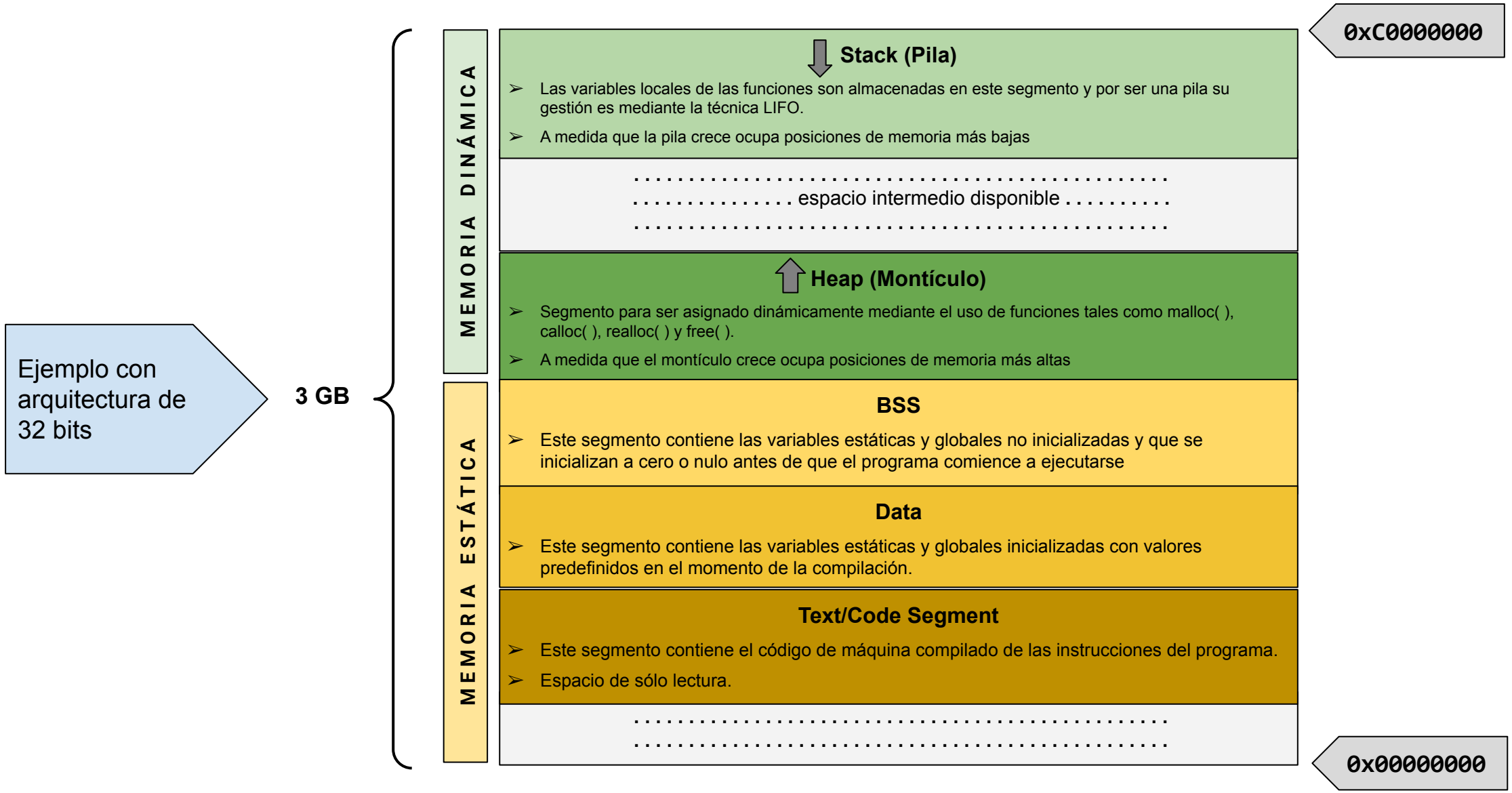

Diagrama de memoria de un proceso

OS Kernel Space: Espacio reservado por el sistema, direcciones no accesibles por el usuario.

Stack: La pila es un sector de memoria que se utiliza para administrar llamadas a funciones y variables locales. Funciona como una estructura de datos de último en entrar, primero en salir (LIFO), lo que significa que el elemento agregado más recientemente es el primero en eliminarse. Cuando se llama a una función, sus variables locales y parámetros de función se asignan en la pila. A medida que las funciones salen, el espacio de pila asignado se desasigna, lo que hace que la pila sea eficiente para administrar datos vinculados al alcance, es decir, crece y se reduce automáticamente durante las llamadas y retornos de funciones.

La memoria de la pila es limitada y la asignación excesiva o las cadenas de llamadas de funciones profundas pueden provocar un desbordamiento de la pila y provocar que el programa falle.

Heap: A diferencia de la pila, el montón es una región de memoria dinámica que se utiliza para la asignación de memoria dinámica. El programador debe administrar explícitamente la memoria asignada en el montón utilizando funciones como malloc(), calloc(), realloc() y free(). La memoria del montón no se desasigna automáticamente y puede persistir más allá del alcance de una única función.

El montón es esencial para manejar estructuras de datos con tamaños dinámicos, como matrices cuyos tamaños se determinan en tiempo de ejecución. Una gestión inadecuada de la memoria del montón puede provocar pérdidas o fragmentación de la memoria, donde la memoria se utiliza de manera ineficiente debido a pequeños fragmentos de memoria asignada pero no utilizada.

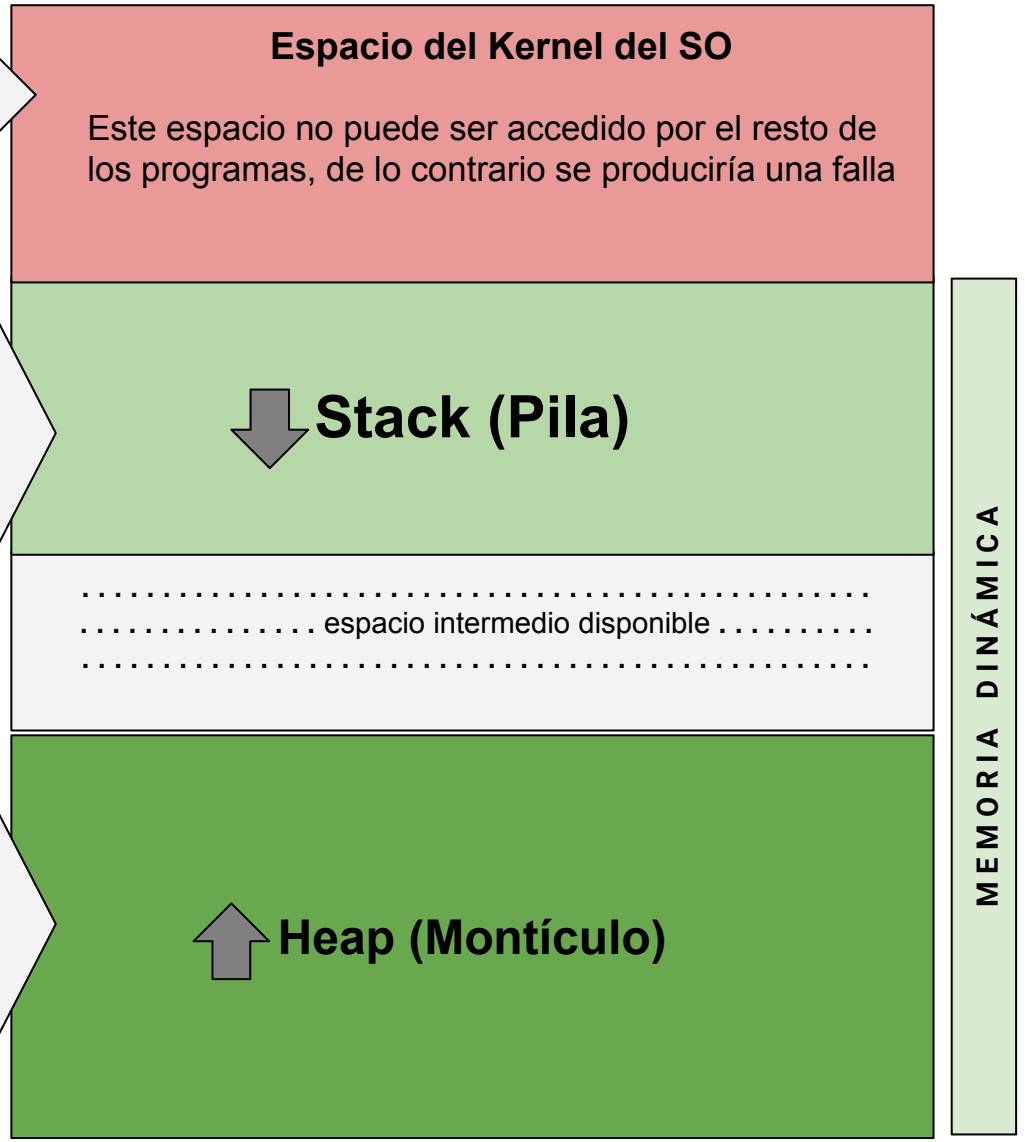

Diagrama de memoria de un proceso

BSS: El segmento BSS contiene variables estáticas y globales no inicializadas. Estas variables se inicializan a valores cero o nulos antes de que el programa comience a ejecutarse.

Data: El segmento de datos contiene variables estáticas y globales inicializadas. Estas variables tienen valores predefinidos establecidos durante la compilación del programa.

Text: El segmento de texto, también conocido como segmento de código, contiene el código de máquina compilado de las instrucciones del programa. Este segmento es de solo lectura y contiene las instrucciones ejecutables que la CPU recupera y ejecuta durante el tiempo de ejecución del programa.

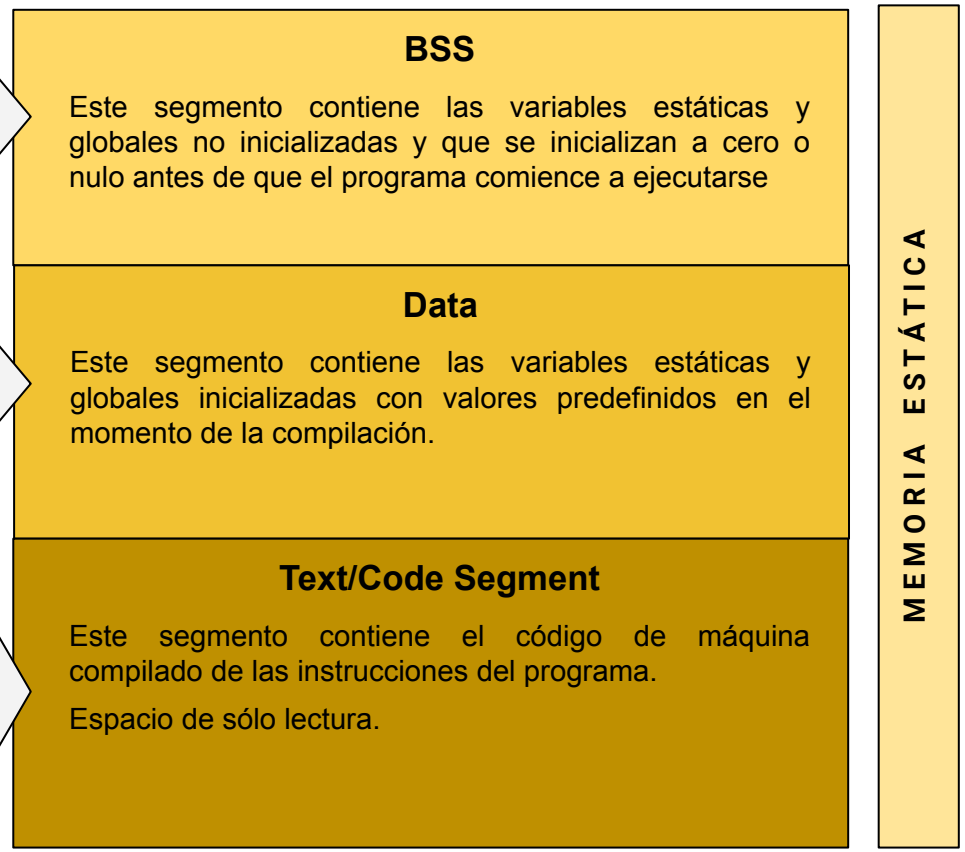

Diagrama de memoria de un proceso

```c
#include <stdlib.h>

int global_var_init = 10;    // (Data)
int global_var_uninit;       // (BSS)

static int static_global_var_init = 21; // (Data)
static int static_global_var_uninit;    // (BSS)

int func() {
    int local_func_var_init = 10;    // (Stack)
    int local_func_var_uninit;       // (Stack)

    static int static_local_func_var_init = 21; // (Data)
    static int static_local_func_var_uninit;    // (BSS)

    int *pDynamicVariable = malloc(sizeof(int));// (Heap)
    free(pDynamicVariable);
}

int main() {
    int local_main_var_init = 10;    // (Stack)
    int local_main_var_uninit;       // (Stack)

    static int static_local_main_var_init = 21; // (Data)
    static int static_local_main_var_uninit;    // (BSS)

    int *pDynamicVariable = malloc(sizeof(int)); // (Heap)
    int *pDynamicArray = malloc(2*sizeof(int));  // (Heap)

    free(pDynamicVariable);
    free(pDynamicArray);
    func();
    return 0;
}
```

Stack (Pila)

Se almacenan variables locales de las funciones y por ser una pila su gestión es mediante la técnica LIFO. Autogestionado por el SO.

. . . . . espacio intermedio disponible . . . . . .

Heap (Montículo)

Segmento para ser asignado dinámicamente por el programador, mediante el uso de funciones tales como malloc( ), calloc( ), realloc( ) y free( ).

MEMORIA DINÁMICA

BSS

Variables estáticas y globales NO inicializadas y que se inicializan a cero o nulo antes de que el programa comience a ejecutarse.

Data

Contiene las variables estáticas y globales inicializadas con valores predefinidos en el momento de la compilación.

Text/Code Segment

Contiene el código de máquina compilado de las instrucciones del programa. Sólo de lectura.

MEMORIA ESTÁTICA

Instrucciones

Bibliografía

1) *Programación en C - Metodología, algoritmos y estructura de datos [Joyanes Aguilar][Mcgraw-Hill] - Capítulo 11 - Asignación dinámica de memoria.*
2) *Computer Systems, A Programmer's Perspective - Capitulo 9.9 - Dynamic Memory Allocation [Bryant & O'Hallaron][3ed][Pearson]*
3) *Pointers in C Programming - Chapter 9 - Dynamic Memory Management [Thomas Mailund][Apress]*
4) *The C Programming Language [Kernighan & Ritchie][2ed][Prentice Hall]*

Links

1) *Linux Manual Page - malloc(3)*
   https://man7.org/linux/man-pages/man3/malloc.3.html
2) *Multiple-byte units*
   https://en.wikipedia.org/wiki/Byte#Multiple-byte_units