# SISTEMAS OPERATIVOS II

**TEMA 2**
**Conceptos de Sistemas Operativos: El caso Unix**

Área de Arquitectura y Tecnología de Computadores
**Escuela Universitaria Politécnica de Teruel**
http://????.????.es/SOII/



## Conceptos de S. O.: El caso Unix

- **Bibliografía:**
  - Silberschatz & Galvin: Sistemas Operativos
  - Stallings: Sistemas Operativos
  - Robbins: Unix programación práctica
    - Capítulos 1, 2 y 3
  - Ayuda de Unix: “man”



## Conceptos de S. O.: El caso Unix

- **Objetivos**:
  - Recordar y afianzar algunos conceptos clave de los S. O. relacionados con procesos, planificación de la CPU, ficheros y gestión de memoria



## Conceptos de S. O.: El caso Unix

- Parte 1: **Procesos**
- Parte 2: **Planificación de la CPU**
- Parte 3: **Ficheros**
- Parte 4: **Gestión de memoria**

# Conceptos de S. O.: El caso Unix

- Parte 4: **Gestión de memoria**
  - Conceptos básicos
  - Paginación
  - Segmentación
  - Memoria virtual
  - Llamadas al sistema y funciones de C relacionadas



# Gestión de Memoria: Generalidades

- **¿Cuál es el propósito de un sistema de computación?**
  - Ejecutar Programas
- **Requisito fundamental**:
  - Los programas (instrucciones y datos) deben estar en memoria principal durante su ejecución
- **Idea**:
  - Cuantos más programas haya en memoria principal mayor es el aprovechamiento de la CPU



# Gestión de Memoria: Generalidades

- **¿En qué consiste la gestión de memoria?**
  - Subdividir la memoria en partes para acomodar el máximo número de procesos posible
  - La memoria debe ser asignada dinámicamente y lo más eficientemente posible. Por tanto, depende en gran medida de la planificación de la CPU
- **Problema**:
  - La memoria es pequeña
  - La memoria es cara
- **Solución**:
  - Utilizar el almacenamiento secundario (disco)



# Gestión de Memoria: Generalidades

- **Objetivos de la gestión de memoria:**
  - Mayor utilización del procesador
  - Mayor eficiencia en las E/S
  - Permitir la comunicación y sincronización entre procesos
  - Facilitar la carga y el movimiento de los procesos en memoria
  - Protección de datos y código (*)
  - Compartir datos y código
  - Liberar al programador de los detalles físicos de la memoria
  - Soportar conceptos de programación de alto nivel:
    - programación modular, montaje dinámico, carga de diferentes módulos por separado, etc.

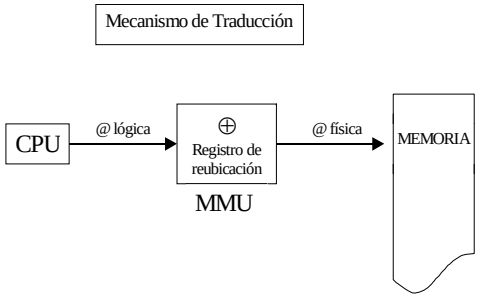
# Objetivos de la gestión de memoria

- * Conseguimos:
  - + seguridad
  - + privacidad de los usuarios
  - + integridad de los procesos



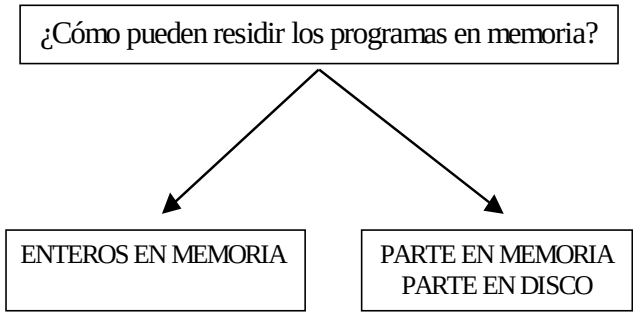
# Gestión de Memoria: Generalidades

- **Espacio de direcciones físico y lógico:**
  - Dirección lógica, es la dirección que genera la CPU
  - Dirección física, es la dirección que se carga en el registro de dirección (AR)



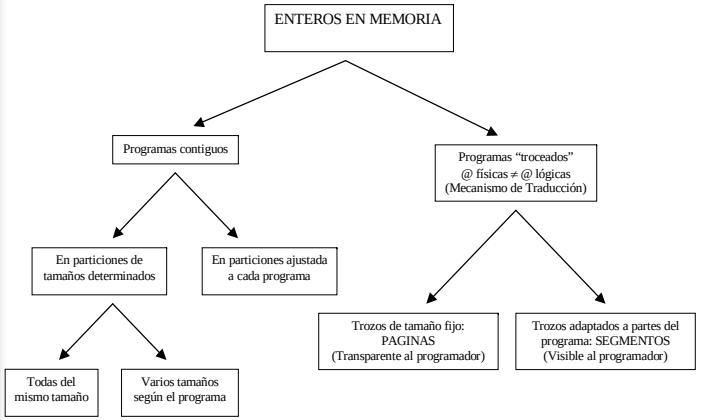



# Gestión de Memoria: Generalidades

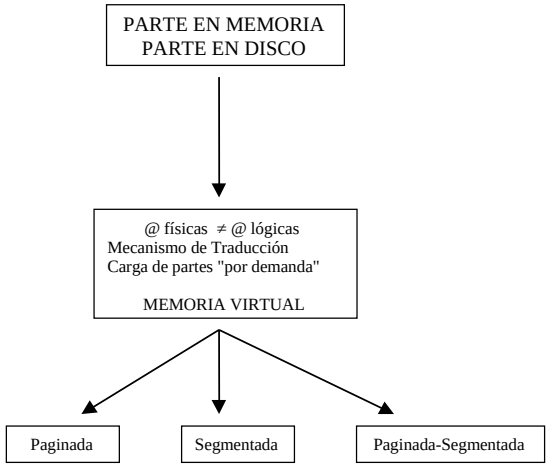





# Gestión de Memoria: Generalidades



## Gestión de Memoria: Generalidades






## Gestión de Memoria: Generalidades

- **Dos conceptos:**
  - **Fragmentación interna**
    - Se produce fragmentación interna cuando no se aprovecha todo el espacio de la memoria asignado a un proceso
  - **Fragmentación externa**
    - Se produce fragmentación externa cuando existe espacio suficiente en la memoria para satisfacer una solicitud de carga de un proceso pero el espacio no es contiguo
    - "La memoria está fragmentada en un gran número de huecos"
    - **Soluciones**:
      - Compactación
      - Paginación



## Gestión de Memoria: Paginación

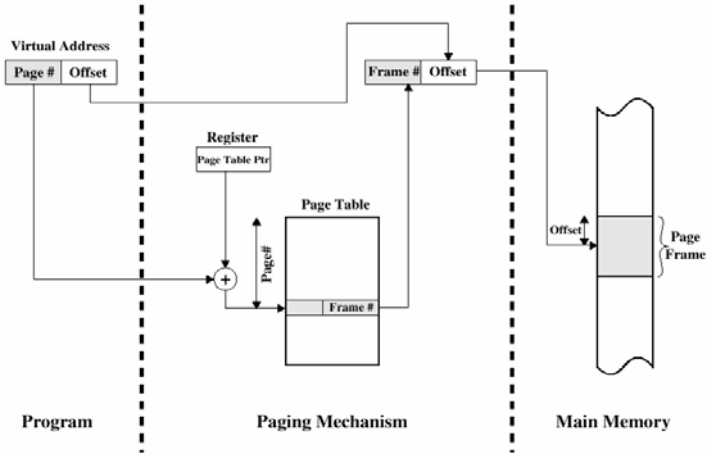
- La memoria física se divide en bloques de tamaño fijo denominados "marcos" (frames)
- La memoria lógica se divide en bloques de tamaño fijo llamados "páginas"
- El S. O. mantiene una tabla de páginas para cada proceso
  - La tabla de páginas contiene los números de "frames" que contienen cada una de las páginas del proceso
  - Permite que el espacio de direcciones físico de un proceso no sea contiguo
  - El PCB guarda un puntero a la tabla de páginas



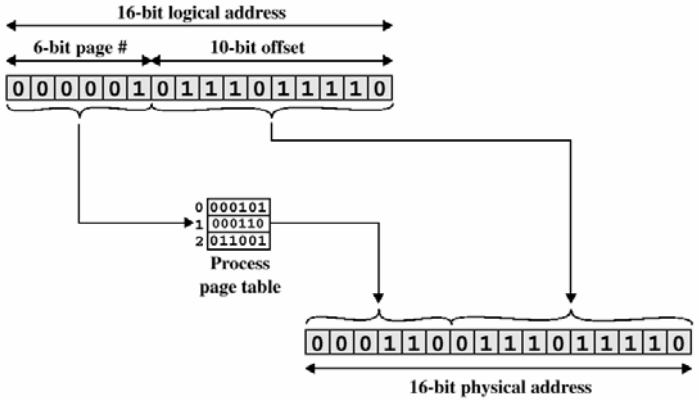
## Gestión de Memoria: Paginación

- Cada dirección generada por la CPU se divide en dos partes:
  - Número de página (P)
  - Desplazamiento en la página (D)
- No hay fragmentación externa:
  - Es una posible solución a la fragmentación externa
- La fragmentación interna se reduce a la última página del proceso

## Hardware de paginación




## Paginación: Traducción de @ lógica a @ física






## Gestión de Memoria: Segmentación

- **¿Cómo imagina un programador la memoria cuando escribe un programa?**
  - Programa principal
  - Sub-rutinas, funciones o módulos
  - Estructuras de datos
- Esquema que apoya la visión del programador
- Cada parte es un segmento de longitud variable y definida por el propósito del segmento



## Gestión de Memoria: Segmentación

- El espacio de direcciones lógico lo forman la colección de segmentos.

  @ lógica ≅ < num_segmento, desplazamiento >

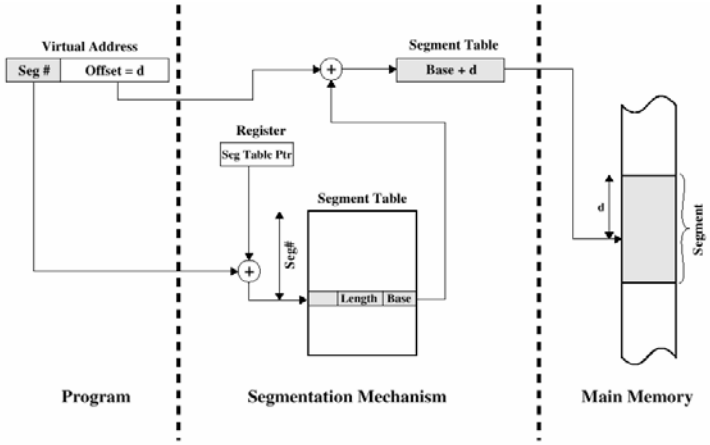
- Cada proceso tiene su tabla de segmentos:
  - El PCB guarda un puntero a la tabla de segmentos
  - Cada entrada contiene:
    - Límite del segmento, indica la longitud del segmento
    - Base del segmento, indica la @ física inicial del segmento

# Gestión de Memoria: Segmentación

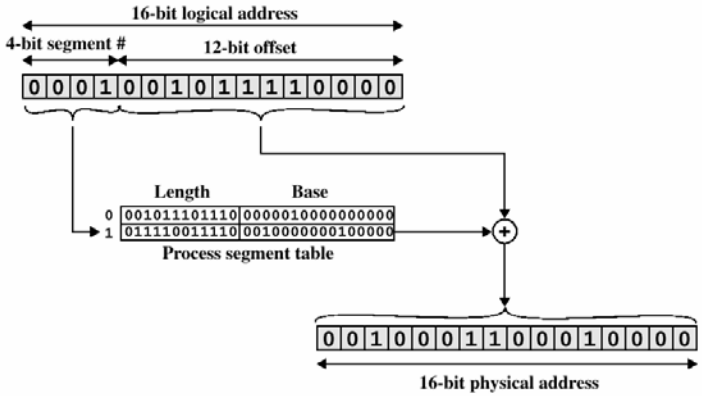
- **Ventajas:**
  - Simplifica la gestión de estructuras de datos crecientes
  - Permite modificar y recompilar los programas sin que sea necesario compilar y volver a montar el programa completo
  - Se pueden compartir segmentos entre procesos
  - Es un buen mecanismo de protección
  - No hay fragmentación interna
- **Desventaja:**
  - Presenta fragmentación externa



# Hardware de segmentación




# Segmentación: Traducción de @ lógica a @ física






# Gestión de Memoria: Memoria Virtual

- **Memoria virtual**
  - Es una técnica de gestión de memoria que permite ejecutar procesos que no están enteros en memoria
- **Funcionamiento**:
  - El S. O. carga en memoria algunos trozos del programa: “conjunto residente”
  - Cuando una dirección es necesaria y no esta en memoria el trozo de programa que contiene dicha dirección debe llevarse a memoria
    - ¿Cómo se realiza esta operación?
  - ¿El funcionamiento de la memoria virtual es eficiente?

# Execution of a Program

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state



# Execution of a Program

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state



# Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently



# Gestión de Memoria: Memoria Virtual

- **Ventajas**:
  - Se pueden mantener más procesos en memoria:
    - Los procesos de usuario ocuparán menos memoria física al no tener que estar completos en memoria
    - Aumenta el aprovechamiento de la CPU
  - La dimensión de los procesos ya no está limitada a la memoria física
    - La tarea de programación se simplifica
  - En casos de intercambio de procesos completos a disco, se requieren menos E/S
    - Aumenta la velocidad de ejecución

# Gestión de Memoria: Memoria Virtual

- **Paginación:**
  - **Hardware que apoya la paginación:**
    - Tabla de páginas
      - Bit de presencia
      - Bit de modificación
      - Número de marco
    - Memoria secundaria
  - **Software necesario para la paginación:**
    - Se necesita poder volver a ejecutar la instrucción abortada después de un fallo de página
    - Algoritmos de reemplazo de páginas

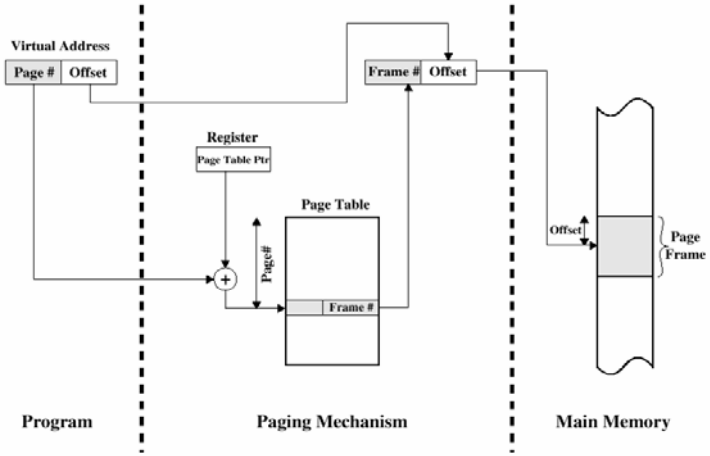


# Gestión de Memoria: Memoria Virtual

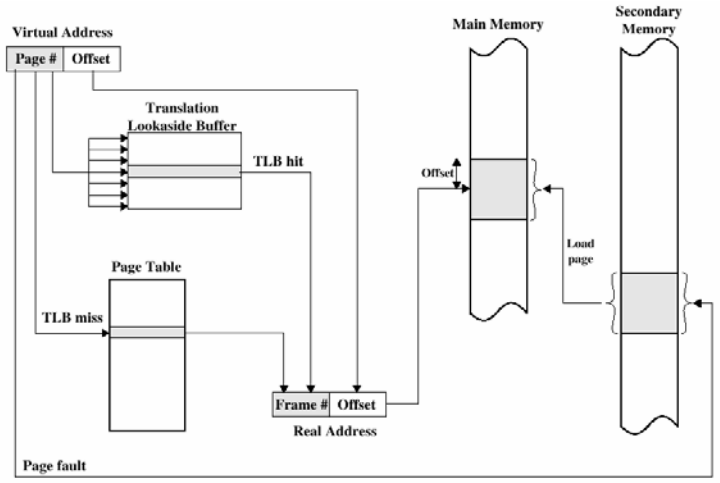
- **Paginación:**
  - **¿Qué sucede si un proceso trata de usar una página que no está en memoria?**
    - Se produce un fallo de página
  - **Pasos para tratar un fallo de página:**
    - Verificar que la página es válida
    - Encontrar un marco libre
    - Planificar una operación de disco para leer la página y cargarla en el marco asignado
    - Modificar la tabla de páginas indicando que la página es válida
    - Volver a ejecutar la instrucción que fue interrumpida



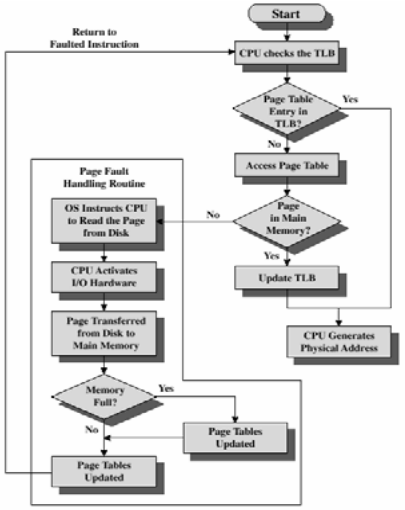
# Hardware de paginación




# Hardware de paginación: Uso del TLB



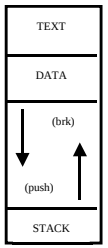
# Funcionamiento del TLB






# Gestión de Memoria en Unix

- **Regiones:**
  - Área contigua del espacio virtual de un proceso
  - Se trata como un único objeto
  - Es uniforme en cuanto a permisos, “shared”, etc



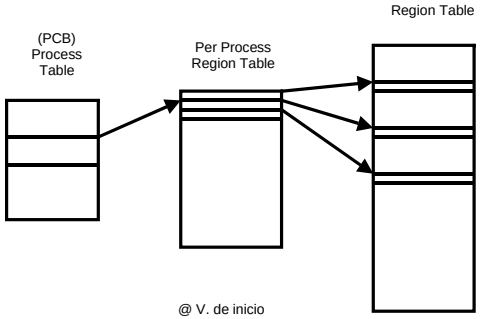



# Gestión de Memoria en Unix

- **Tablas:**
  - Tabla de Procesos: “Process table”
  - Tabla de Regiones por Proceso: “Per process region table”
  - Tabla de Regiones: “Region Table”



# Gestión de Memoria en Unix



## Gestión de Memoria en Unix

- **Sistemas basados en SWAP:**
  - Espacio en disco(swap device):
    - Una partición de disco, almacena procesos expulsados
    - La gestión de espacio se realiza mediante particiones variables
  - Sólo se copia a memoria la parte del espacio virtual usada
  - Razones de expulsión de un proceso:
    - fork()
    - brk()
    - crecimiento de la pila
    - se requiere espacio para recuperar otro proceso



## Gestión de Memoria en Unix

- **Sistemas basados en SWAP:**
  - Proceso “swapper”:
    - Es un proceso mas
    - Devuelve procesos de disco a memoria
    - Entra en ejecución cada cierto tiempo
    - Mira si hay procesos “preparados en disco”
    - Si no hay suficiente espacio en memoria busca un proceso víctima y lo expulsa



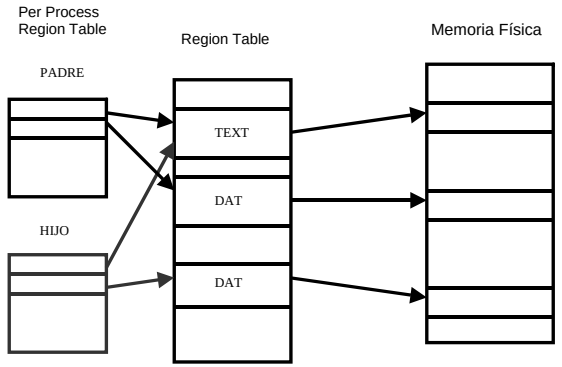
## Gestión de Memoria en Unix

- **Funcionamiento de fork( ) con SWAP:**
  - El sistema hace una copia del padre en memoria:
    - La región de “text” no se duplica
    - Si no hay espacio suficiente en memoria la copia se realiza en disco



## Gestión de Memoria en Unix

- **Funcionamiento de fork( ) con SWAP:**



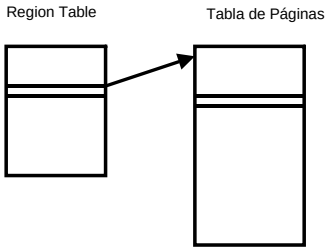
# Gestión de Memoria en Unix

- **Sistemas basados en Demand Paging:**
  - Estructuras de control:
    - Region Table
    - Tabla de Páginas
    - Tabla de paginas del sistema
    - Tabla de páginas en disco
  - Proceso “pager stealer”:
    - Proceso encargado de liberar marcos llevando las páginas a disco
    - Se pone en funcionamiento dependiendo del nivel de alarma



# Gestión de Memoria en Unix

- **Sistemas basados en Demand Paging:**

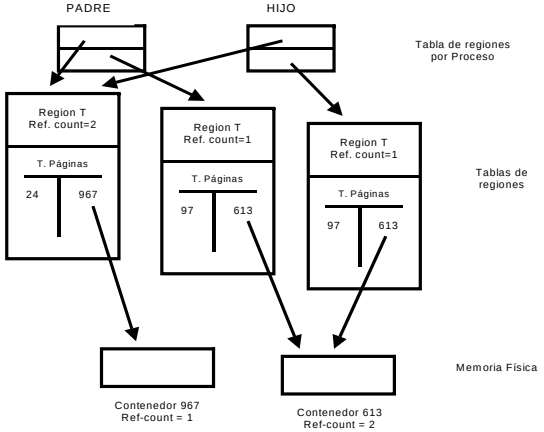



Contenedor
Acceso: read, write …
Estado: MV copy-on-write
Disk Block Descriptor



# Gestión de Memoria en Unix

- **Funcionamiento de fork( ) con Demand Paging: System V**






# Gestión de Memoria en Unix

- **Funcionamiento de fork( ) con Demand Paging: BSD**
  - Crea una copia de todas las páginas privadas: Data, Stack
  - Ofrece una llamada alternativa a fork( )
    - vfork( ): no copia ni las tablas de páginas
    - Trata las regiones de datos como la de texto
    - Padre e hijo comparten memoria física
    - Sólo debe usarse para llamar a exec( ) inmediatamente

## Gestión de Memoria:Llamadas al sistema

- **Llamadas al sistema brk y sbrk**
  - La llamada al sistema **brk** cambia el límite superior del segmento de datos al valor end_data_segment
    - Devuelve el valor antiguo de end_data_segment
  - La llamada al sistema **sbrk** añade increment bytes al límite superior del segmento de datos
    - increment puede ser negativo
  - brk, sbrk - change data segment size
    - #include <unistd.h>
    - int brk(void *end_data_segment);
    - void *sbrk(ptrdiff_t increment);



## Gestión de Memoria:Llamadas al sistema

- **Llamadas al sistema brk y sbrk**
  - brk( ) puede fallar por varias razones:
    - Al ampliar la región se colisiona con otra
    - Al ampliar la región se sobrepasa el espacio virtual máximo del proceso
    - Faltan recursos:
      - No hay entradas en la tabla de regiones del proceso
      - No hay entradas en la tabla de regiones general
      - No hay espacio físico (dependiendo de la implementación)



## Gestión de Memoria:Funciones de librería

- **Funciones de librería calloc, malloc, free, realloc**
  - La función de C **calloc** reserva espacio para nmemb elementos cada uno de size bytes, además inicializa el espacio reservado con ceros
  - La función de C **malloc** reserva espacio para al menos size bytes, pero no inicializa el espacio reservado
  - La función de C **free** libera el espacio apuntado por ptr. Este espacio no queda liberado para el kernel
  - La función de C **realloc** cambia el tamaño del bloque apuntado por ptr al valor size. Devuelve un puntero al inicio del bloque que puede ser distinto al original
  - **Nota.-** No se deben mezclar llamadas a brk( ) con funciones de librería



## Gestión de Memoria:Funciones de librería

- **Funciones de librería calloc, malloc, free, realloc**
  - calloc,  malloc, free, realloc - Allocate and free dynamic memory
    - #include <stdlib.h>
    - void *calloc(size_t nmemb, size_t size);
    - void *malloc(size_t size);
    - void free(void *ptr);
    - void *realloc(void *ptr, size_t size);