**Técnico Superior en** Análisis, Desarrollo y Programación de Aplicaciones

Plan 2012

| PROGRAMA DE LA MATERIA: | **Sistemas Operativos (11215)** |
|---|---|

| | |
|---|---|
| Área Sistemas | Equipo Docente<br>Lic. En Sistemas de Computación Juan Carlos Romero |
| Módulos semanales 2 dos<br>Días de dictado:<br>L M M J V S<br>X | |

| | |
|---|---|
| Materias Correlativas | Sistemas de Computación (11107) |

## 1.1. Ítems del perfil que se desarrollarán

Respecto del perfil profesional, esta asignatura contribuye al desarrollo de las aptitudes básicas del profesional en sistemas, como la comprensión del funcionamiento del software de base, y sus aplicaciones. Aportará los conocimientos necesarios que le permita al profesional adecuarse a las necesidades del mercado laboral, destacando la funcionalidad de su tarea como de Programador, brindando la capacidad de adaptarse a los cambios tecnológicos que se dan tan vertiginosamente, tanto en lo físico como en lo lógico (hardware y software).

Formando un profesional con capacidad de adaptación en lo conceptual, práctico operativo y generador de nuevo software.

## 1.2. Objetivos

- Brindar al alumno las herramientas conceptuales y prácticas aplicadas a un sistema operativo.
- Fortalecer al alumno en su capacidad de programar en lenguaje C, abordando la programación concurrente.
- Formar al alumno para que entienda el diseño del Sistema Operativo y pueda participar en el desarrollo de software de base.
- Preparar al alumno para que pueda decidir y asesorar en la elección de un SO.
- Que el alumno adquiera los conocimientos necesarios para realizar operaciones básicas de un Sistema Operativo.

## 2.0 Programa

Unidad 1 – Introducción y Estructura del Sistema Operativo

1. Definición de Sistema Operativo y necesidad de su utilización.
2. Desde los Sistemas por lotes hasta los sistemas distribuidos.
3. Componentes del Sistema.
4. Servicios, llamadas al Sistema y programas del Sistema.
5. Estructura del Sistema.
6. Máquinas virtuales.
7. Diseño, implementación y generación de Sistemas.

Unidad 2 – Procesos

1. Proceso, multitarea, estado e información del proceso.
2. Planificación de procesos o planificación del procesador.
3. Señales, excepciones y temporizadores.
4. Servicios que brinda el SO para el manejo de procesos.
5. Procesos livianos o hilos.

Parte Práctica

Unidad 3 – Comunicación y Sincronización entre Procesos

1. Procesos Concurrentes.
2. Sección crítica.
3. Mecanismos de comunicación y sincronización.
4. Servicios POSIX para la comunicación y sincronización.
5. Problemas clásicos de sincronización.

La práctica de esta unidad se basa en dos aspectos, primero la identificación de los elementos estudiados en los temas de comunicación y sincronización entre procesos y segundo en la implementación real de un problema, para poder verlo en funcionamiento.

En esta práctica el alumno podrá afianzar la teoría estudiada y aplicarla en un sistema operativo real, haciendo uso de herramientas indispensables para un profesional programador, estas herramientas son: editores de texto, compiladores de lenguaje C, y el uso del intérprete de comandos.

Este trabajo intenta favorecer el acceso a las siguientes metas de aprendizaje:

a) Entender los conceptos que se tienen que aplicar en la difícil problemática de la comunicación y sincronización entre procesos.

b) Comprender el nuevo y maravilloso mundo de la programación concurrente específicamente en la comunicación y sincronización entre procesos.

c) Comprender el problema de la comunicación y la sincronización y la sección crítica de los procesos.

Unidad 4 – Gestión de Memoria y Memoria Virtual

1. Espacios de direcciones, lógico y físico.
2. Intercambio.
3. Asignación Contigua.
4. Paginación.
5. Segmentación.
6. Paginación por demanda.
7. Reemplazo de páginas.
8. Asignación de marcos.
9. Hiperpaginación.

Unidad 5 – Sistemas de entrada y salida y estructura del almacenamiento secundario

1. Hardware de entrada y salida.
2. Interfaz de entrada y salida de las aplicaciones.
3. Subsistema de entrada y salida del núcleo.
4. Planificación y administración de discos.
5. Administración del espacio de intercambio.

Unidad 6 – Gestión de Archivos y Directorios

1. Archivos, accesos y directorios.
2. Protección y semántica de consistencia.
3. Estructura del sistema de archivos y métodos de asignación.
4. Administración de espacio libre.
5. Implementación de directorios.

Unidad 7 – Protección

1. Objetivos y dominio de protección.
2. Matriz de acceso e Implementación.
3. Sistemas basados en capacidades y protección basada en el lenguaje.
4. El problema de la seguridad.
5. Validación y contraseñas de un solo uso.
6. Amenazas por programas al sistema y vigilancia de amenazas.

## 3.0 Bibliografía

Obligatoria

Silberschatz A. Galvin P. Gagne G.; Fundamentos de Sistemas Operativos; 7ma Edición; Mc Graw Hill 2007.

Ampliatoria

Stallings W.; Sistemas Operativos –Aspectos Internos y principios de diseño-; 5ta Edición; Prentice Hall 2007.

Carretero Pérez J. De Miguel Anasagasti P. García Carballeira F. Pérez Costoya F.; Sistemas Operativos –Una visión aplicada-; Mc Graw Hill.

Tanenbaum A. Woodhull A.; Sistemas Operativos –Diseño e implementación-; 2da Edición; Prentice Hall; 1997.

## 4.0 Condiciones de aprobación

**Evaluación formativa:**

Entrega de los trabajos prácticos de las distintas unidades. Predisposición para el trabajo grupal. Grillas de seguimiento y planillas de auto evaluación.

- Dos exámenes parciales individuales
- Trabajos Prácticos requeridos
- Trabajos prácticos sugeridos

Aprobación de los exámenes parciales:

- Habrá dos exámenes parciales en las fechas estipuladas en el cronograma de la asignatura
- Se tomarán exámenes recuperatorios de ambos parciales

La aprobación requiere de una nota mayor o igual a 4 (cuatro), esto significa el 60% de los puntos requeridos. En el parcial se evalúan los ejercicios y/o preguntas y el conjunto. En la aprobación se tendrá en cuenta la calidad de la presentación.

Aprobación de los trabajos prácticos requeridos: se realizarán a lo largo del año, se registrarán en una carpeta en la que figurarán las distintas versiones de cada trabajo, con las distintas correcciones.

**Evaluación de resultados**

Los alumnos con promedio entre 3.99 y 1 rendirán examen recuperatorio de la asignatura, además de cumplimentar todos los requisitos de la cursada, y de aprobarlo con 4 puntos o más accederán al examen final.

La acreditación de la asignatura estará supeditada a la aprobación del examen final de carácter presencial y obligatorio. Se exigirá, además, el porcentaje de cumplimiento de las actividades previstas para la aprobación de la cursada.

# Práctica

# Programación de Procesos

## Ejercicios

**1.- ¿**Qué interpretación tiene la salida generada por la ejecución de los siguientes procesos?

```
#include <stdio.h>
#include <unistd.h>
main()  {
        fork();
        printf("Quien soy, el padre o el hijo\n");
        exit(0);
  }
```

---

```
#include <stdio.h>
#include <unistd.h>
main()  {
        fork();
        fork();
        printf("Soy  el proceso %d\n", getpid());
        exit(0);
  }
```

---

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main()  {
      pid_t pid;
      if ((pid = fork()) < 0)
            {
              printf("Ha ocurrido un error\n");
             exit(pid);
          }
        printf("Quien soy, el padre o el hijo\n");
        exit(0);
  }
```

**2.-** Modifique el programa anterior para identificar el hijo y el padre. Lograr que el padre muestre su identificador y el identificador de su padre, y que el hijo muestre su identificador y el identificador de su padre.

**3.-** Agregar al programa anterior la siguiente línea **system("ps")** que permite la ejecución del comando ("**ps**"). El ps se tiene que ejecutar antes que el **fork.**

**4.-** Agregar al programa anterior la ejecución de ("**ps**") con el parámetro que permita visualizar el **PPID** de los procesos. El comando lo debe ejecutar el hijo. Imprimir una corrida de ejecución y relacionar los identificadores de procesos.

**5.-** Convertir el proceso hijo en huérfano; utilizar la llamada al sistema **sleep**. Presentar una corrida del proceso que lo demuestre.

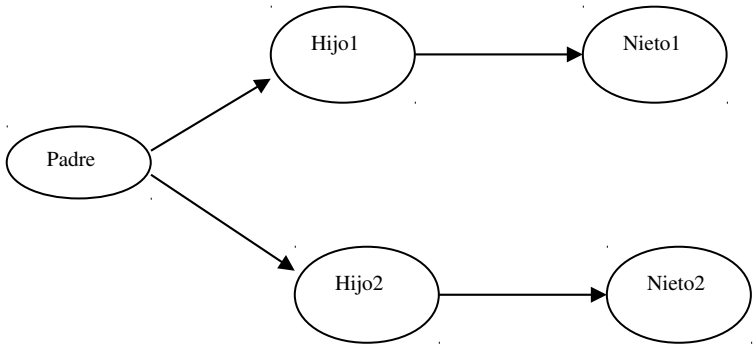
**6.-** Convertir el proceso hijo en zombie; utilizar la llamada al sistema **sleep**. Presentar una corrida del proceso que lo demuestre.

**7.-** Utilizar la llamada al sistema **wait** para que un proceso padre espere la finalización de un proceso hijo.

**8.-** Cuál es la estructura jerárquica de parentesco de procesos que genera el siguiente programa?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main()  {
        pid_t pid;
        pid = fork();
        if (pid == 0) pid = fork();
        if (pid > 0) pid = fork();
        printf("%d hijo de %d\n",getpid(),getppid());
}
```

**9.-** Generar utilizando la llamada al sistema **fork**, la siguiente estructura de procesos.




En el proceso PADRE definir un dato local de **main()** antes del **fork** que será heredado por su descendencia, lograr que los HIJOS y los NIETOS modifiquen el dato utilizando alguna operación aritmética, antes de terminar cada proceso debe mostrar por pantalla el valor del dato. Imprimir una corrida de los procesos y analizar los resultados obtenidos.

**10.-** Realizar un programa utilizando la llamada al sistema **execl,** el programa antes de terminar debe pedir que se presione la tecla ENTER. Probar el programa pasándole como parámetro al **execl** un comando válido y luego pasándole un comando no válido. Cuál es la diferencia en la ejecución?

**11.-** Realizar un programa utilizando la llamada al sistema fork, el proceso hijo debe ejecutar el comando(“ps”) utilizando system y en la línea siguiente utilizando execl. Analizar la diferencia entre la salida del comando (“ps”) lanzado por el system y la salida del comando (“ps”) utilizando la llamada al sistema execl.

**12.-** Crear una cadena de N procesos, informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la

salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos de la cadena.

**13.-** Crear un abanico de N procesos, informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos del abanico.

**14.-** Verificar mediante la creación de un proceso padre y su correspondiente hijo que en el instante de la creación del hijo, este hereda el contenido de las variables del padre, luego cualquier modificación de las mismas ya no son vistas por el otro proceso. Explique porqué? Cuál es la única variable que no se hereda su contenido?

**15.-** Realizar un programa que le permita verificar, que cuando un proceso hijo termina antes que el padre, y el padre está esperando su terminación con la llamada al sistema wait, si el hijo queda en estado zombie o no.

**16.-** Crear una cadena de N procesos, y sincronizar la terminación de los mismos en el orden inverso al que fueron creados. Informar por cada proceso los siguientes datos: ID de proceso padre, ID de proceso actual, ID de proceso hijo. Generar un archivo con la salida y verificar la existencia de procesos zombies y huérfanos. N es un valor que se ingresa desde la línea de comandos e indica la cantidad de procesos de la cadena.

**17.-** Realizar un programa llamado mishell que permita ejecutar comandos. Cuando se ejecuta mishell aparece el prompt msh$ luego escribir el comando a ejecutar, cuando se termina de ejecutar vuelve a aparecer el prompt msh$ . (NO USAR system( )).

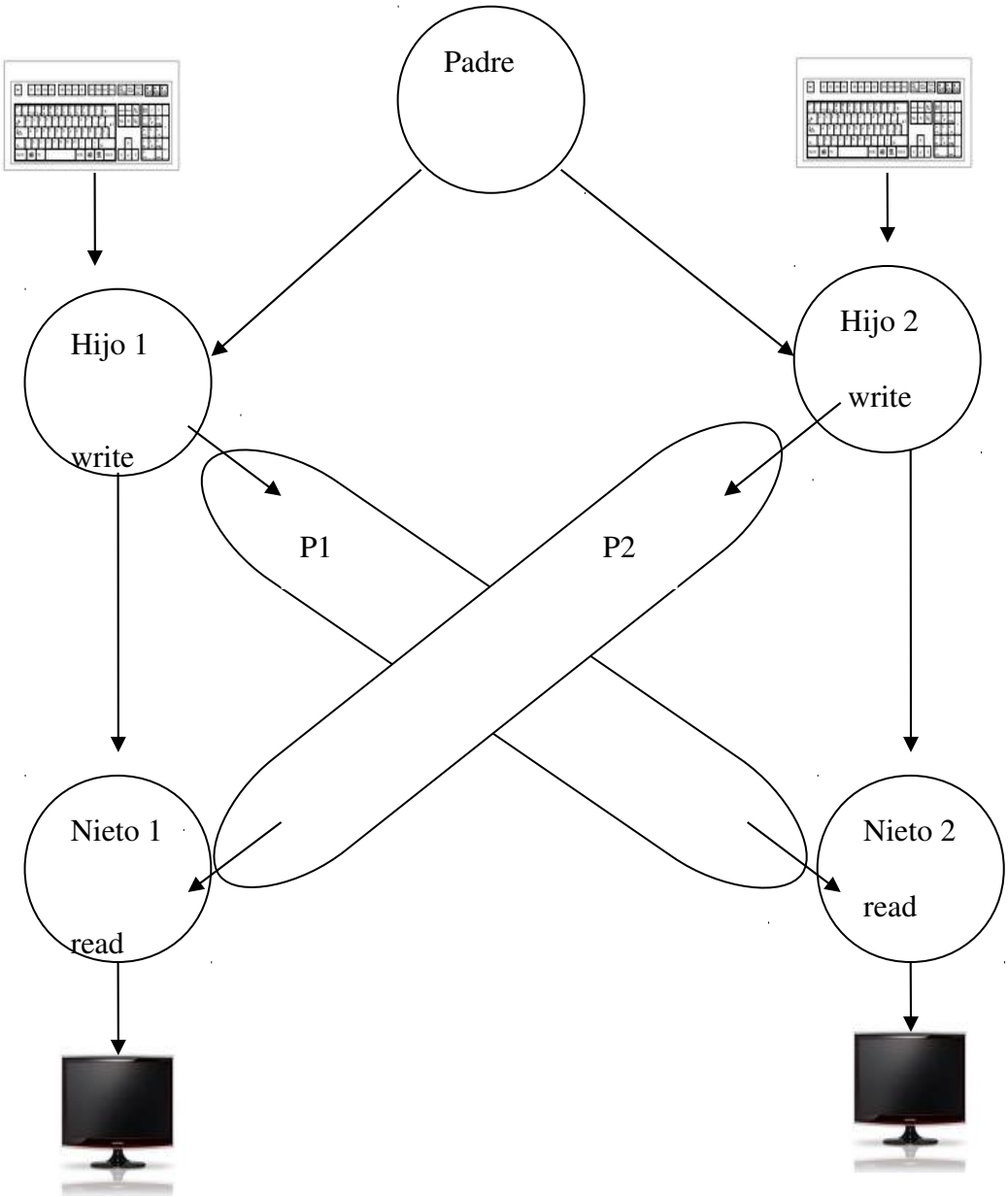
## Ejercicio con pipes

- Realizar un CHAT entre procesos emparentados usando Pipes.

## Objetivos

- Aprender características básicas de la programación concurrente.
- Aprender los mecanismos que usa el Sistema Operativo para la creación de procesos.
- Aprender el uso del recurso Pipe para la comunicación entre procesos emparentados.

## Diagrama del Planteo




## Descripción

**Padre:** Crea los dos recursos pipe, el pipe P1 y el pipe P2, comienza la creación de la estructura de procesos, especifica que no será escritor ni lector de ninguno de los dos pipes y espera la terminación del hijo 1 y del hijo 2.

**Hijo 1:** Será únicamente escritor del pipe P1, el mensaje que escribe en el pipe P1, lo obtiene de la entrada estándar. La lectura de la entrada estándar termina cuando obtiene el string "chau" y se queda esperando la terminación del Nieto 1.

**Hijo 2:** Será únicamente escritor del pipe P2, el mensaje que escribe en el pipe P2, lo obtiene de la entrada estándar. La lectura de la entrada estándar termina cuando obtiene el string "chau" y se queda esperando la terminación del Nieto 2.

**Nieto 1:** Será únicamente lector del pipe P2, la lectura del pipe la escribe en la salida estándar, y termina cuando lee del pipe "chau".

**Nieto 2:** Será únicamente lector del pipe P1, la lectura del pipe la escribe en la salida estándar, y termina cuando lee del pipe "chau".

## Implementación

Se pide la implementación del ejercicio en un entorno operativo Linux, utilizando el compilador del lenguaje C.

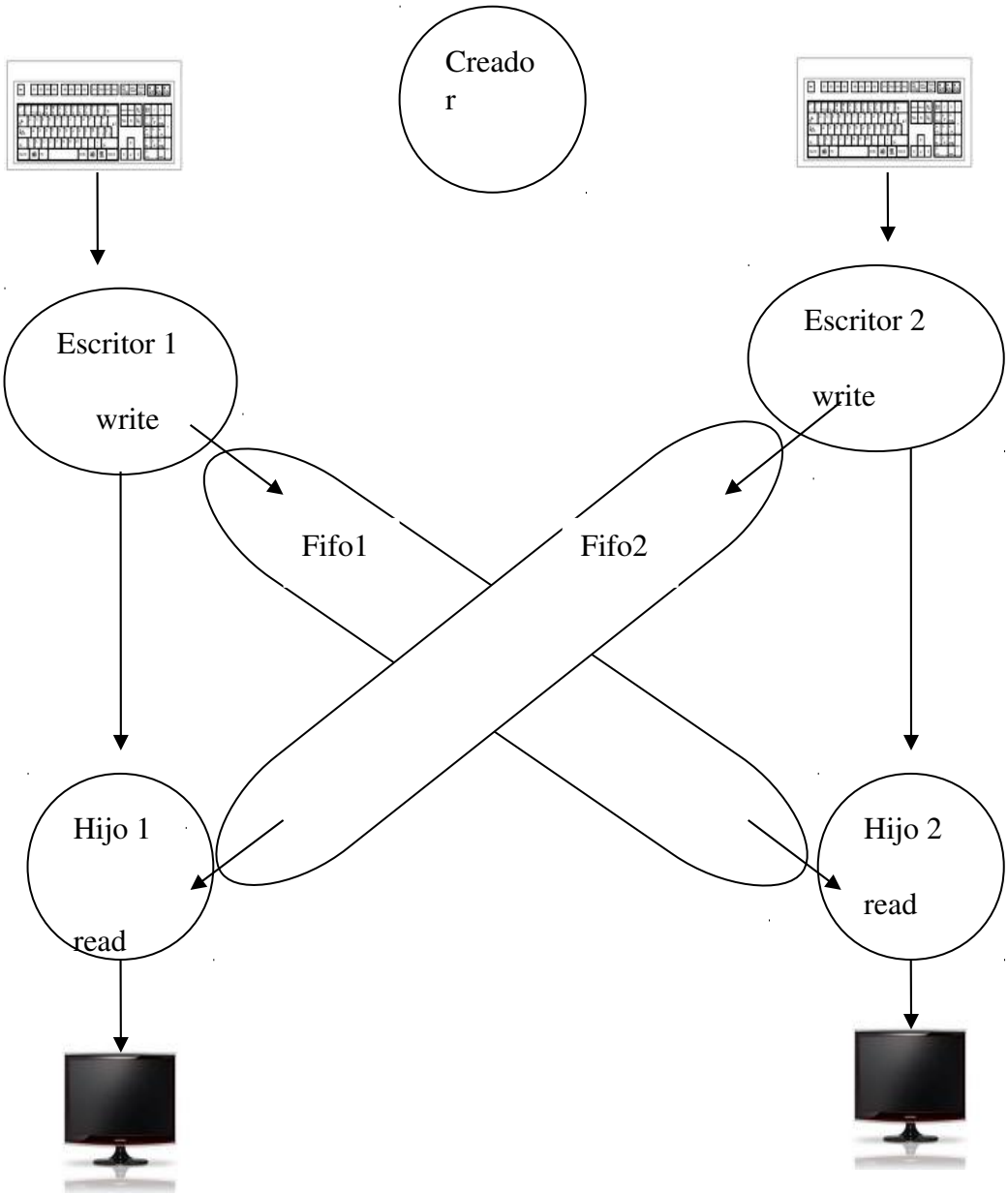
## Ejercicio con fifos

- Realizar un CHAT entre procesos independientes y emparentados usando Fifos.

## Objetivos

- Aprender características básicas de la programación concurrente.
- Aprender los mecanismos que usa el Sistema Operativo para la creación de procesos.
- Aprender el uso del recurso Fifo para la comunicación entre procesos independientes y emparentados.

## Diagrama del Planteo




## Descripción

**Creador Proceso Independiente:** Crea los dos recursos fifo, el fifo 1 y el fifo 2 y termina.

**Escritor 1 Proceso Independiente:** Será únicamente escritor del fifo 1, el mensaje que escribe en el fifo 1, lo obtiene de la entrada estándar. La lectura de la entrada estándar termina cuando obtiene el string "chau" y se queda esperando la terminación del Hijo 1.

**Escritor 2 Proceso Independiente:** Será únicamente escritor del fifo 2, el mensaje que escribe en el fifo 2, lo obtiene de la entrada estándar. La lectura de la entrada estándar termina cuando obtiene el string "chau" y se queda esperando la terminación del Hijo 2.

**Hijo 1 Proceso Emparentado:** Será únicamente lector del fifo 2, la lectura del fifo 2 la escribe en la salida estándar, y termina cuando lee del fifo 2 "chau".

**Hijo 2 Proceso Emparentado:** Será únicamente lector del fifo 1, la lectura del fifo 1 la escribe en la salida estándar, y termina cuando lee del fifo 1 "chau".

### Implementación

Se pide la implementación del ejercicio en un entorno operativo Linux, utilizando el compilador del lenguaje C.

## Ejercicio 1 Memoria Contigua Simple

**Las soluciones a estos ejercicios pueden realizarse en pseudo código o en lenguaje C**

**Definir la metodología, los procesos y las estructuras de datos necesarias para implementar las siguientes simulaciones**

**1.1-** Simular la administración de la asignación de memoria contigua simple. Pedir memoria contigua que simule la memoria RAM de la máquina. El 10 % de dicha memoria será destinada al núcleo del Sistema Operativo. Guardar en una estructura de datos los punteros al comienzo de memoria del Sistema Operativo, al final del Sistema Operativo, al comienzo de memoria de usuario, al final de memoria de usuario. Mostrar los punteros indicando a que apunta cada uno de ellos.

**1.2.-** Modifique el programa del ejercicio 1.- para poder cargar un proceso en la memoria contigua. Modifique la estructura de datos para que se pueda registrar el proceso cargado en memoria. Ingrese el identificador y tamaño del proceso. Si la memoria de usuario es mayor o igual al tamaño del proceso actualizar los datos de la estructura y calcular la fragmentación externa en caso contrario rechazar el proceso.

## Ejercicio 2 Memoria Particionada Fija con particiones de igual tamaño

**2.1**.- Simular la administración de la asignación de memoria particionada fija con particiones de igual tamaño. Pedir memoria contigua que simule la memoria RAM de la máquina. El 10 % de dicha memoria será destina al núcleo del Sistema Operativo. Ingresar la cantidad de particiones fijas en la cuál se dividirá la memoria de Usuario. Generar el Id de la partición

automáticamente. Actualizar las la tabla de particiones. La tabla de particiones contendrá la siguiente información (id partición, dirección de comienzo de la partición, estado de la partición). Mostrar la información contenida en la tabla de particiones.

**2.2**.- Agregar a la simulación anterior la posibilidad de cargar procesos en las particiones libres. El programa debe permitir ingresar procesos mientras haya memoria libre para asignar, por cada proceso ingresar (Id de proceso y tamaño del proceso). La tabla de particiones deberá contener (Id de partición, dirección de comienzo de partición, tamaño de la partición, estado de la partición, id de proceso asignado a la partición, fragmentación interna). Mostrar por cada proceso ingresado la tabla de particiones.

# Ejercicio 3 Memoria Particionada Fija con particiones de tamaño variable

**3.1**.- Simular la administración de la asignación de memoria particionada fija con particiones de tamaño variable. Pedir memoria contigua que simule la memoria RAM de la máquina. El 10 % de dicha memoria será destina al núcleo del Sistema Operativo. Ingresar la cantidad de particiones fijas en la cuál se dividirá la memoria de Usuario. Ingresar el Id y el tamaño de cada partición. Actualizar las la tabla de particiones. La tabla de particiones contendrá la siguiente información (id partición, dirección de comienzo de la partición, tamaño de la partición, estado de la partición). Mostrar la información contenida en la tabla de particiones.

**3.2**.- Agregar a la simulación anterior la posibilidad de cargar procesos en las particiones libres. El programa debe permitir ingresar procesos mientras haya memoria libre para asignar, por cada proceso ingresar (Id de proceso y tamaño del proceso). La tabla de particiones deberá contener (Id de partición, dirección de comienzo de partición, tamaño de la partición, estado de la partición, id de proceso asignado a la partición, fragmentación interna). Mostrar por cada proceso ingresado la tabla de particiones. Utilizando la técnica del mejor ajuste.

# Ejercicio 4 Memoria Particionada Reubicable

**4.1.-** Simular la administración de la asignación de memoria particionada variable reubicable. Pedir memoria contigua que simule la memoria RAM de la máquina. El 10 % de dicha memoria será destina al núcleo del Sistema Operativo. Ingresar procesos indicando el identificador y el tamaño del proceso, a medida que se ingresan los procesos actualizar la tabla de particiones libres y la tabla de particiones ocupadas. La tabla de particiones libres contendrá la siguiente información (id partición, dirección de comienzo de la partición, tamaño de la partición). La tabla de particiones ocupadas contendrá la siguiente información (id partición, dirección de comienzo de la partición, tamaño de la partición, id del proceso).Mostrar la información contenida en la tabla de particiones y la tabla de particiones ocupadas.

# Ejercicio 5 Memoria Paginada Simple (Un Proceso)

**5.1.-** Simular la administración de la asignación de memoria paginada simple para un proceso.

Algunos pasos a tener en cuenta para la construcción de la simulación.
**a)** Definir una tabla de páginas para un proceso utilizando una estructura que contenga (Número de página y Número de Marco).
**b)** Definir un vector de marcos donde la posición del elemento coincide con un número de marco y el elemento indica si ese marco está asignado o no (1 o 0).
**c)** Ingresar cantidad de memoria RAM disponible.
**d)** Pedir memoria utilizando malloc.
**e)** Ingresar el tamaño del proceso.
**f)** Definir páginas y marcos de 32 bytes.
**g)** Definir un vector de punteros al comienzo de cada marco de memoria
**h)** Asignar cada página del proceso a marcos de memoria y actualizar la tabla de páginas y el vector de información de asignación de marcos.
**i)** Mostrar el contenido de todas las estructuras de información.

# Ejercicio 6 Memoria Paginada Simple (N Procesos)

**6.1.-** Simular la administración de la asignación de memoria paginada simple para N procesos.

Algunos pasos a tener en cuenta para la construcción del programa.

**a)** Definir una tabla de páginas para cada uno de los procesos utilizando estructuras, donde cada una de ellas contiene (Número de página y Número de marco).
**b)** Definir un vector de marcos donde la posición del elemento coincide con un número de marco y el elemento indica si ese marco está asignado o no (1 o 0).
**c)** Ingresar una cantidad de memoria RAM disponible.
**d)** Pedir memoria utilizando malloc para simular la memoria RAM.
**e)** Ingresar N que es la cantidad de procesos. Por cada proceso ingresar el identificador de proceso y el tamaño del proceso.
**f)** Definir páginas y marcos de 32 bytes.
**g)** Definir un vector de punteros al comienzo de cada marco de memoria
**h)** Asignar cada página de cada proceso a marcos de memoria y actualizar las tablas de páginas y el vector de marcos asignados.
**i)** Mostrar el contenido de todas las estructuras de información.