Dibujo en blanco y negro

Descripción generada automáticamente con confianza baja

**INSTITUTO POLITECNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**Práctica 5**

**Comunicación inter procesos (IPC) en Linuex y en Windows**

**Sistemas Operativos**

**Grupo: 2CM12**

**Integrantes:**

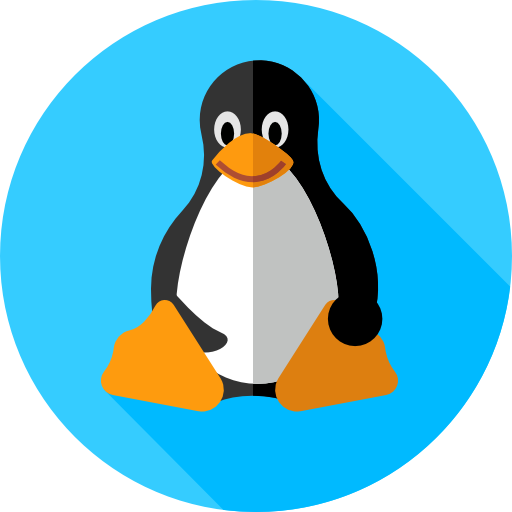
* **Baldovinos Gutiérrez Kevin**
* **Bocanegra Heziquio Yestlanezi**
* **Castañares Torres Jorge David**
* **Hernández Hernández Rut Esther**

**Profesor**

**Jorge Cortes Galicia**

Pantalla de computadora

Descripción generada automáticamente con confianza baja



Contenido

[Introducción 4](#_Toc104734258)

[Comunicación Entre Procesos 4](#_Toc104734259)

[Competencia. 6](#_Toc104734260)

[Desarrollo. 6](#_Toc104734261)

[pipe() 6](#_Toc104734262)

[Sintaxis 6](#_Toc104734263)

[Parámetros 6](#_Toc104734264)

[Retorno 6](#_Toc104734265)

[shmget() 7](#_Toc104734266)

[Sintaxis 7](#_Toc104734267)

[Parámetros 7](#_Toc104734268)

[shmat() 8](#_Toc104734269)

[Sintaxis 8](#_Toc104734270)

[Parámetros 8](#_Toc104734271)

[Compilación 11](#_Toc104734272)

[Sección Windows 18](#_Toc104734273)

[Padre 18](#_Toc104734274)

[Hijo 20](#_Toc104734275)

[Nieto 21](#_Toc104734276)

[Linux 27](#_Toc104734277)

[Código - compilación 27](#_Toc104734278)

[Código hijo 29](#_Toc104734279)

[Código nieto.c 30](#_Toc104734280)

[Ejecución del código 31](#_Toc104734281)

[Archivo de texto creado 31](#_Toc104734282)

[Windows 32](#_Toc104734283)

[Código padre.c 32](#_Toc104734284)

[Código hijo.c 34](#_Toc104734285)

[Código nieto.c 35](#_Toc104734286)

[Ejecución 36](#_Toc104734287)

[Imagen 1 Parámetros 6](#_Toc104734619)

[Imagen 2 Ejecución del código de Linux 9](#_Toc104734620)

[Imagen 3 gcc hola.c -o h 9](#_Toc104734621)

[Imagen 4 ./h 9](#_Toc104734622)

[Imagen 5 Código 10](#_Toc104734623)

[Imagen 6 Padre.c Imagen 7 Padre1.c 18](#_Toc104734624)

[Imagen 8 Padre2.c 19](#_Toc104734625)

[Imagen 9 Hijo.c 20](#_Toc104734626)

[Imagen 10 Nieto.c 21](#_Toc104734627)

[Imagen 11 Captura de código 23](#_Toc104734628)

[Imagen 12 Ejecución de código 23](#_Toc104734629)

[Imagen 13 Ejecución servidor 24](#_Toc104734630)

[Imagen 14 Ejecución cliente 24](#_Toc104734631)

[Imagen 15 Código de windows servidor 25](#_Toc104734632)

[Imagen 16 Código de windows cliente 25](#_Toc104734633)

[Imagen 17 Compilación código windows desde CMD 26](#_Toc104734634)

# Introducción

# Comunicación Entre Procesos

La comunicación entre procesos, en inglés IPC (Inter-process Communication) es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC. El IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.

La comunicación se establece siguiendo una serie de reglas (protocolos de comunicación). Los protocolos desarrollados para internet son los mayormente usados: IP (capa de red), protocolo de control de transmisión (capa de transporte) y protocolo de transferencia de archivos , protocolo de transferencia de hipertexto (capa de aplicación).

Los procesos pueden ejecutarse en una o más computadoras conectadas a una red. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envío de la misma) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.

Un proceso puede ser de dos tipos:

* *Proceso independiente.*
* *Proceso de cooperación.*

Un proceso independiente no se ve afectado por la ejecución de otros procesos, mientras que un proceso cooperativo puede verse afectado por otros procesos en ejecución. Aunque uno puede pensar que esos procesos, que se ejecutan de forma independiente, se ejecutarán de manera muy eficiente, en realidad, hay muchas situaciones en las que la naturaleza cooperativa se puede utilizar para aumentar la velocidad, la conveniencia y el modularidad computacional. La comunicación entre procesos (IPC) es un mecanismo que permite a los procesos comunicarse entre sí y sincronizar sus acciones.

La comunicación entre estos procesos puede verse como un método de cooperación entre ellos. Los procesos pueden comunicarse entre sí a través de ambos:

1. Memoria compartida
2. Paso de mensajes

La IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red Subyacente.

Un sistema operativo puede implementar ambos métodos de comunicación. Primero, discutiremos los métodos de comunicación de memoria compartida y luego el paso de mensajes. La comunicación entre procesos que utilizan memoria compartida requiere que los procesos compartan alguna variable y depende completamente de cómo la implementará el programador. Una forma de comunicación que utiliza la memoria compartida se puede imaginar así: Supongamos que proceso1 y proceso2 se están ejecutando simultáneamente y comparten algunos recursos o usan información de otro proceso. Process1 genera información sobre ciertos cálculos o recursos que se utilizan y la mantiene como un registro en la memoria compartida. Cuando process2 necesita usar la información compartida, verificará el registro almacenado en la memoria compartida y tomará nota de la información generada por process1 y actuará en consecuencia.

Ahora, comenzaremos nuestra discusión sobre la comunicación entre procesos a través del paso de mensajes. En este método, los procesos se comunican entre sí sin utilizar ningún tipo de memoria compartida. Si dos procesos p1 y p2 quieren comunicarse entre sí, proceden de la siguiente manera:

* Establezca un enlace de comunicación (si ya existe un enlace, no es necesario volver a establecerse).
* Empiece a intercambiar mensajes utilizando primitivas básicas.
* Necesitamos al menos dos primitivas:

°° Enviar (mensaje, destino) o enviar (mensaje)

°° Recibir (mensaje, host) o recibir (mensaje)

# Competencia.

## **Desarrollo.**

1. A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de la función: pipe(), shmget(), shmat(). Explique los argumentos y retorno de la función.

## pipe()

Esta función se emplea para crear una tubería o pipe

Conceptualmente, un tubo o pipe es una conexión entre dos procesos, de manera que la salida estándar de un proceso se convierte en la entrada estándar del otro proceso. En el sistema operativo

UNIX, las tuberías son útiles para la comunicación entre procesos relacionados,

Una tubería tiene en realidad dos descriptores de fichero: uno para el extremo de escritura y otro para el extremo de lectura. Como los descriptores de fichero de UNIX son simplemente enteros, un pipe o tubería no es más que un array de dos enteros:

Para crear la tubería se emplea la función pipe(), que abre dos descriptores de fichero y almacena su valor en los dos enteros que contiene el array de descriptores de fichero.

El primer descriptor de fichero es abierto como O\_RDONLY, es decir, sólo puede ser empleado para lecturas. El segundo se abre como O\_WRONLY, limitando su uso a la escritura. De esta manera se asegura que el pipe sea de un solo sentido: por un extremo se escribe y por el otro se lee, pero nunca al revés. Ya hemos dicho que si se precisa una comunicación “full-duplex”, será necesario crear dos tuberías para ello.

### Sintaxis

int pipe(int fds[2]);

### Parámetros

fds[0] : Sera el descriptor de fichero para el extremo de lectura

fds[1] : Sera el descriptor de fichero para el extremo de escritura

### Retorno

Se retorna el entero 0 en caso de éxito y se retorna -1 en caso de algún error

**Imagen 1 Parámetros**

## **shmget()**

Esta función se emplea para la creación o acceso a una zona de memoria compartida. La memoria compartida se crea por un proceso mediante una llamada al sistema, la zona que se reserva en memoria no está en el espacio de direcciones del proceso, es una zona de memoria gestionada por el sistema operativo. Después, otros procesos a los que se les dé permiso para acceder a esa zona de memoria podrán también leer o escribir de ella.

### Sintaxis

int shmget(key\_t key, int size, int shmflg);

### Parámetros

key Es una clave que genera el sistema y que será un elemento esencial para poder acceder a la zona de memoria que crearemos. La clave se obtiene previamente utilizando la función ftok(); que produce siempre una clave fija con los mismos argumentos: Para Usarla:

key = ftok(“.”, ‘S’);

La clave debe ser la misma para todos los procesos que quieran usar esta zona de memoria

size Indicará el tamaño de la memoria compartida. Se suele utilizar la opción de sizeof (tipo\_variable) para que se tome el tamaño del tipo de dato que habrá en la memoria compartida.

Shmflg Indicará los derechos para acceder a la memoria, si se crea si no existe y caso de que exista, se da o no un error.

#### Retorno

Se retorna el identificador de la memoria compartida si no ha habido error Se retorna -1 en caso de un error

## shmat()

Una vez ya creada la memoria compartida, pero para utilizarla necesitamos saber su dirección (observe que shmget no nos la indica). Para utilizar la memoria compartida debemos antes vincularla con alguna variable de nuestro código. De esta manera, siempre que usemos la variable vinculada estaremos utilizando la variable compartida. Para Para establecer un vínculo utilizamos la función shmat.

### Sintaxis

void \*shmat(int shmid, char \*shmaddr, int shmflag);

### Parámetros

Shmid Es el identificador de la memoria compartida. Lo habremos obtenido con la llamada Shmget

Shmaddr Normalmente, valdrá 0 o NULL que indicará al sistema operativo que busque esa zona de memoria en una zona de memoria libre, no en una dirección absoluta

shmflg permisos. Por ejemplo. Aunque hayamos obtenido una zona de memoria con permiso para escritura o lectura, podemos vincularla a una variable para solo lectura. Si no queremos cambiar los permisos usamos 0

#### Retorno

Se retorna un puntero a la zona de memoria compartida

Se retorna -1 en caso de algún error

2.Capture, compile y ejecute el siguiente programa. Observe su funcionamiento.

Texto

Descripción generada automáticamente

**Imagen 2 Ejecución del código de Linux**

Texto

Descripción generada automáticamente

**Imagen 3 gcc hola.c -o h**

Texto

Descripción generada automáticamente

**Imagen 4 ./h**

El programa funciona perfectamente pero al principio no entendía cómo funcionaba una vez que revise el código lo entendí más claramente, el programa obtiene del buffer en este caso de la consola de comandos, en este caso escribimos una oración y el programa obtiene la cadena mediante el buffer de la consola.

1. Capture, compile y ejecute los siguientes programas. Observe su funcionamiento. Ejecute de la siguiente forma: C:\>nombre\_programa\_padre nombre\_programa\_h o

Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Imagen 5 **Código**

# Compilación

El programa tiene un funcionamiento correcto, como podemos observar, el proceso h o es el primero en ser ejecutado y muestra el mensaje que hemos enviado mediante el programa principal, posteriormente se da la notificación del momento en que ha concluido el proceso para así saber en qué momento se está ejecutando el programa principal.

Texto

Descripción generada automáticamente

4. Programe una aplicación que cree un proceso h o a partir de un proceso padre, el proceso padre enviará al proceso h o, a través de una tubería, dos matrices de 10 x 10 a multiplicar por parte del h o, mientras tanto el proceso h o creará un h o de él, al cual enviará dos matrices de 10 x 10 a sumar en el proceso h o creado, nuevamente el envío de estos valores será a través de una tubería. Una vez calculado el resultado de la suma, el proceso h o del h o devolverá la matriz resultante a su abuelo (vía tubería). A su vez, el proceso h o devolverá la matriz resultante de la multiplicación que realizó a su padre. Finalmente, el proceso padre obtendrá la matriz inversa de cada una de las matrices recibidas y el resultado lo guardará en un archivo para cada matriz inversa obtenida.

Programe esta aplicación tanto para Linux como para Windows utilizando las tuberías de cada sistema operativo.

Texto

Descripción generada automáticamente

Texto

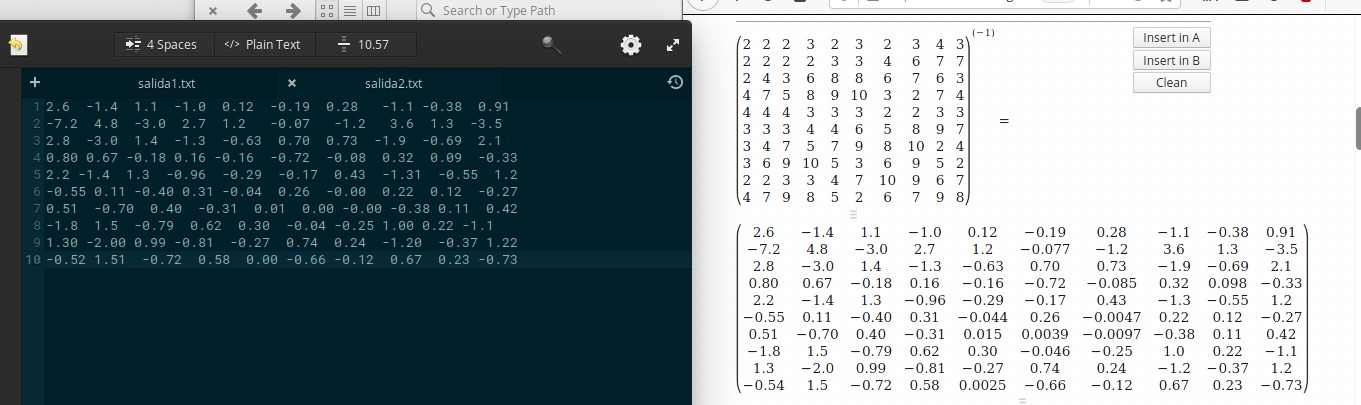
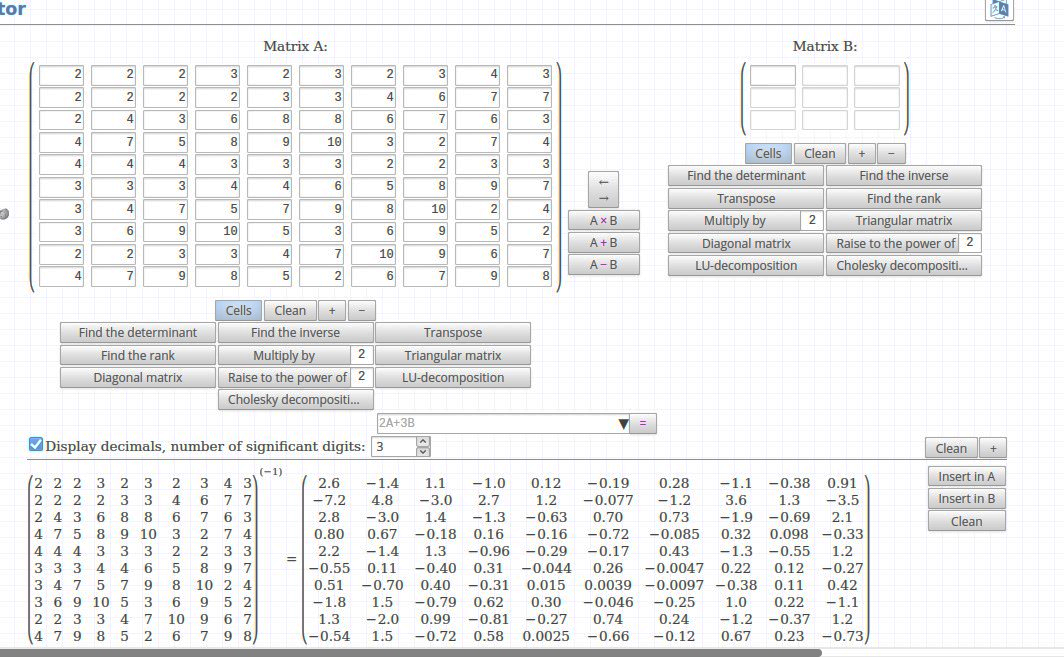
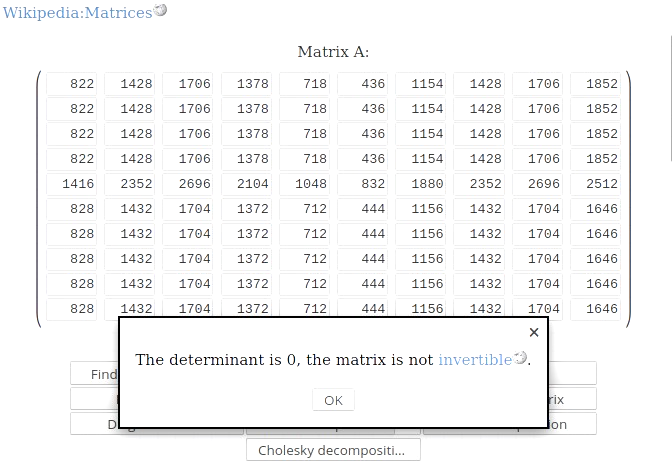
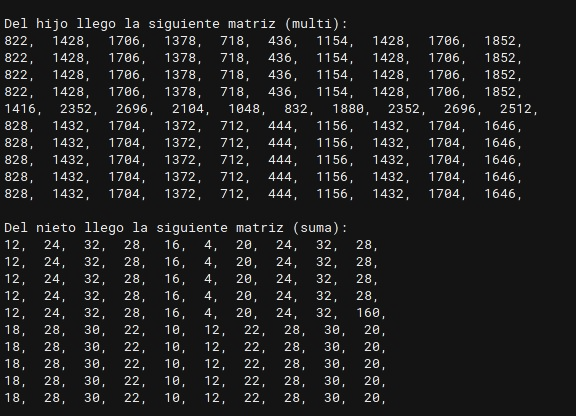
Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente



Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

# Sección Windows

## Texto Descripción generada automáticamenteTexto Descripción generada automáticamentePadre

**Imagen 6 Padre.c Imagen 7 Padre1.c**

Texto

Descripción generada automáticamente

**Imagen 8 Padre2.c**

## Hijo

Texto

Descripción generada automáticamente

**Imagen 9 Hijo.c**

## Nieto

Texto

Descripción generada automáticamente

**Imagen 10 Nieto.c**

Tabla

Descripción generada automáticamente

Un conjunto de letras negras en un fondo blanco

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

5. Capture, compile y ejecute los siguientes programas para Linux. Observe su funcionamiento.

Texto

Descripción generada automáticamente

**Imagen 11 Captura de código**

Texto

Descripción generada automáticamente

**Imagen 12 Ejecución de código**

Texto

Descripción generada automáticamente

**Imagen 13 Ejecución servidor**

Texto

Descripción generada automáticamente

**Imagen 14 Ejecución cliente**

El programa funciona de manera optimo, el servidor crea el espacio de memoria compartida que guarda en dicho espacio de memoria las letras del abecedario una vez que lo ejecutamos el servidor espera a que otro programa acceda al espacio para leer la información guardarla en un arreglo de caracteres e imprimirlo en la consola del cliente

6. Capture, compile y ejecute los siguientes programas para Windows. Observe su funcionamiento.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteImagen 15 **Código de windows servidor**

**Imagen 16 Código de windows cliente**

Texto

Descripción generada automáticamente

**Imagen 17 Compilación código windows desde CMD**

Como podemos observar el programa en cuestión funciona de forma adecuada, vemos que el Servidor crea el espacio de memoria correspondiente para almacenar las letras del abecedario las cuales permanecen ahí hasta que otro programa intente acceder a esta memoria compartida, en este caso el Cliente al momento de ser ejecutado obtiene las letras del abecedario y las imprime en pantalla como debería ser

7. Programe nuevamente la aplicación del punto cuatro utilizando en esta ocasión memoria compartida en lugar de tuberías (utilice tantas memorias compartidas cómo requiera). Programe esta aplicación tanto para Linux como para Windows utilizando la memoria compartida de cada sistema operativo.

# Linux

## Texto Descripción generada automáticamenteCódigo - compilación

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## Código hijo

La mayor parte del código es similar al anterior por ello solo se mostrarán las funciones declaradas con un comentario que describe lo que hace

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## Código nieto.c

Texto

Descripción generada automáticamente

## Ejecución del código

Texto

Descripción generada automáticamente

## Archivo de texto creado

Imagen que contiene Texto

Descripción generada automáticamente

# Windows

## Texto Descripción generada automáticamenteCódigo padre.c

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## Texto Descripción generada automáticamenteCódigo hijo.c

Texto

Descripción generada automáticamente

## Código nieto.c

Texto

Descripción generada automáticamente

## Ejecución

Imagen que contiene Texto

Descripción generada automáticamente

## Archivo de texto creado

Texto

Descripción generada automáticamente

# Conclusiones