



Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Operativos

“Práctica 5. Comunicación interprocesos (IPC) en Linux y Windows”

Grupo: 2CM8

Integrantes:

- Martínez Coronel Brayan Yosafat.
- Monteros Cervantes Miguel Angel.
- Ramírez Olvera Guillermo.
- Sánchez Méndez Edmundo Josue.

Profesor: Cortés Galicia Jorge



Práctica 5. Comunicación interprocesos (IPC) en Linux y Windows

Introducción

La comunicación entre procesos es una función básica de los sistemas operativos que 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. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC).

En un sistema, los procesos pueden ejecutarse independientemente o cooperando entre sí. Los intérpretes de comandos son ejemplos típicos de procesos que no precisan la cooperación de otros para realizar sus funciones. En cambio, los procesos que sí cooperan necesitan comunicarse entre sí para poder completar sus tareas. La comunicación entre procesos puede estar motivada por la competencia o el uso de recursos compartidos o porque varios procesos deban ejecutarse sincronizadamente para completar una tarea. Para que puedan realizarse ambos tipos de interacciones, es necesario que el sistema operativo provea de servicios para posibilitar la comunicación entre procesos. El sistema operativo provee mínimamente dos primitivas, "enviar" y "recibir", normalmente llamadas *send* y *receive*. Asimismo, debe implementarse un enlace de comunicación entre los procesos de la comunicación (*pipe*). Este enlace puede ser unidireccional o multidireccional según permita la comunicación en uno o en varios sentidos.

La comunicación puede ser:

- Síncrona: Quien envía permanece bloqueado esperando a que llegue una respuesta del receptor antes de realizar cualquier otro ejercicio.
- Asíncrona: Quien envía continúa con su ejecución inmediatamente después de enviar el mensaje al receptor.
- Persistente: El receptor no tiene que estar operativo al mismo tiempo que se realiza la comunicación, el mensaje se almacena tanto tiempo como sea necesario para poder ser entregado (por ejemplo, un e-mail).
- Momentánea (transient): El mensaje se descarta si el receptor no está operativo al tiempo que se realiza la comunicación. Por lo tanto, no será entregado.
- Directa: Las primitivas "enviar" y "recibir" especifican el nombre del proceso con el que se comunican.

Las operaciones básicas *send* y *receive* se definen de la siguiente manera:

send(*P*, *mensaje*): envía un mensaje al proceso *P* (*P* es el proceso destino).

receive(*Q*, *mensaje*): espera la recepción de un mensaje por parte del proceso *Q* (*Q* es el proceso fuente).

Nota: *receive* puede esperar de un proceso cualquiera un mensaje, pero el *send* sí debe especificar a quién va dirigido y cuál es el mensaje.

- Indirecta: Es aquella donde la comunicación está basada en una herramienta o instrumento ya que el emisor y el perceptor están a distancia.
- Simétrica: Todos los procesos pueden enviar o recibir. También llamada bidireccional para el caso de dos procesos.
- Asimétrica: Un proceso puede enviar, los demás procesos sólo reciben. También llamada unidireccional. Suele usarse para hospedar servidores en Internet.
- Uso de búfer automático: El transmisor se bloquea hasta que el receptor recibe el mensaje (capacidad cero).

Hablando un poco sobre las tuberías y memorias compartidas las cuales usamos en esta práctica:

Esencialmente, las tuberías, ya sean con nombre o anónimas, se utilizan para transmisión de mensajes. Alguien envía una información al destinatario y el destinatario puede recibirla. La memoria compartida es más como publicar datos: alguien coloca datos en la memoria compartida y los lectores (potencialmente muchos) deben usar la sincronización, por ejemplo, a través de semáforos para conocer el hecho de que hay nuevos datos y debe saber cómo leer la región de la memoria para encontrar la información.

Con las tuberías, la sincronización es simple y está integrada en el propio mecanismo de la tubería: sus lecturas y escrituras congelarán y descongelarán la aplicación cuando suceda algo interesante. Con la memoria compartida, es más fácil trabajar de forma asincrónica y buscar nuevos datos solo de vez en cuando, pero a costa de un código mucho más complejo. Además, puede obtener comunicación de muchos a muchos, pero requiere más trabajo nuevamente. Además, debido a lo anterior, la depuración de la comunicación basada en canalizaciones es más fácil que la depuración de la memoria compartida.

La memoria compartida también le brinda más control sobre el almacenamiento en búfer y el uso de recursos; dentro de los límites permitidos por el sistema operativo, es usted quien decide cuánta memoria asignar y cómo usarla. Con las tuberías, el sistema operativo controla las cosas automáticamente, por lo que una vez más pierde algo de flexibilidad, pero se libera de mucho trabajo.

Resumen de los puntos más importantes: uso de tuberías para la comunicación uno a uno, menos codificación y menos dejar que el sistema operativo maneje las cosas, memoria compartida para muchos a muchos, más control manual sobre las cosas, pero a costa de más trabajo y una depuración más difícil.

1. Competencias.

El alumno comprende el funcionamiento de las tuberías (pipes) sin nombre y de la memoria compartida como mecanismos de comunicación entre procesos tanto en el sistema operativo Linux como Windows para el desarrollo de aplicaciones concurrentes con soporte de comunicación.

2. Desarrollo

2.1. Sección Linux

2.1.1. Información de las llamadas al sistema

2.1.1.1. pipe()

```
01. #include <fcntl.h>
02. #include <unistd.h>
03.
04. int pipe(int pipefd[2]);
```

La llamada **pipe()** crea una tubería, es decir, un canal de comunicación unidireccional que puede ser usado para la comunicación interprocesos.

- **pipefd[0]** se refiere al descriptor de archivo del lado de la lectura de la tubería.
- **pipefd[1]** se refiere al descriptor de archivo del lado de la escritura de la tubería.
- En caso de éxito, el valor de retorno es 0. En caso de error, devuelve -1.

2.1.1.2. shmget()

```
01. #include <sys/ipc.h>
02. #include <sys/shm.h>
03.
04. int shmget(key_t key, size_t size, int shmflg);
```

La llamada **shmget()** devuelve el identificador del segmento de memoria compartida asociado con el valor de la llave **key**.

- **key_t key**: La llave del segmento de memoria compartida.
- **size_t size**: El tamaño de la memoria compartida, redondeado al múltiplo más cercano de **PAGE_SIZE**.
- **int shmflg**: En sus últimos 9 bits menos significativos guarda los permisos de la memoria compartida. Además, puede tomar los siguientes valores usando máscara de bits:
 - **IPC_CREAT**: Crea un nuevo segmento. Si este **flag** no se especifica, la función tratará de encontrar el segmento asociado a **key** y revisar si tenemos permisos para acceder al segmento.
 - **IPC_EXCL**: Se usa con el **flag** anterior para garantizar que el segmento se cree, es decir, si no se crea ocurrirá un error.

- En caso de éxito, devuelve un identificador válido de la memoria compartida. En caso de error, devuelve -1.

2.1.1.3. shmat()

```
01. #include <sys/types.h>
02. #include <sys/shm.h>
03.
04. void *shmat(int shmid, const void *shmaddr, int shmflg);
```

La llamada **shmat()** adjunta el segmento de memoria compartida asociado con el identificador al espacio de direcciones del proceso que realiza la llamada.

- int **shmid**: El identificador de la memoria compartida.
- const void ***shmaddr**: La dirección de memoria del proceso actual en donde se adjuntará la memoria compartida. El comportamiento es el siguiente:
 - Si **shmaddr** es un puntero nulo, el segmento se adjunta en la primera dirección disponible seleccionada por el sistema operativo.
 - Si **shmaddr** no es nulo y el flag **SHM_RND** está activado, el segmento se adjunta en la dirección dada por $shmaddr - ((uintptr_t)shmaddr \% SHMLBA)$.
 - Si **shmaddr** no es nulo y el flag **SHM_RND** está desactivado, el segmento se adjunta en la dirección dada por **shmaddr**.
 - Si el flag **SHM_RDONLY** está activado y el proceso actual tiene permisos de lectura, el segmento se adjunta para solo lectura. Si el flag está desactivado y el proceso actual tiene permisos de lectura y escritura, el segmento se adjunta para lectura y escritura.
- En caso de éxito, devuelve el comienzo de la dirección de memoria del segmento que se adjuntó. En caso de error, la memoria no se adjunta y devuelve -1.

2.1.2. Ejemplo de tuberías en Linux

Código(EjCodigo1.c)

```
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include <unistd.h>
04. #include <string.h>
05. #define VALOR 1
06. int main(void)
07. {
08.     int desc_arch[2];
09.     char bufer[100];
10.     if (pipe(desc_arch) != 0)
11.         exit(1);
12.     if (fork() == 0)
13.     {
14.         while (VALOR)
15.         {
16.             read(desc_arch[0], bufer, sizeof(bufer));
17.             printf("Se recibió: %s\n", bufer);
18.         }
19.     }
20.     while (VALOR)
21.     {
22.         fgets(bufer, 100, stdin);
23.         write(desc_arch[1], bufer, strlen(bufer) + 1);
24.     }
25. }
```

Compilación y ejecución del programa:

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$ gcc EjCodigo1.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$ ./a.out
Sistemas Operativos
Se recibió: Sistemas Operativos

2CM8
Se recibió: 2CM8

IPN-ESCOM
Se recibió: IPN-ESCOM

Esto es Linux
Se recibió: Esto es Linux
```

En este código podemos ver el uso de la llamada al sistema pipe() la cual funciona para crear una tubería y compartir una cadena que se escriba en la terminal entre dos procesos los cuales tienen un ciclo que siempre se va a cumplir y la única manera de poder finalizar el programa es pulsando las combinación de teclas Ctrl+Z, además, podemos observar cómo recibe una cadena “bufere” el proceso padre con la función fgets() y la comparte por medio de un pipe “desc_arch[1]” el cual es la entrada y escribe en la tubería con la función write() y en su respectiva salida del pipe que es “des_rch[0]” se lee la cadena “bufere” con la función read(). Finalmente hay que mencionar que se añadió una librería al programa original y se cambió la función gets por fgets ya que se generaba errores de compilación.

2.1.3. Programa con tuberías en Linux (Usando Matrices)

Código(MatricesPipes.c)

```
01. #include <stdlib.h>
02. #include <stdio.h>
03. #include <sys/types.h>
04. #include <sys/wait.h>
05. #include <unistd.h>
06. #include <string.h>
07. #include "EstructuraMatriz.h"
08.
09. int main(void) {
10.     int tuberial[2];
11.     int tuberia2[2];
12.     pid_t pidpadre, pidnieto;
13.     int status;
14.     int status1;
15.
16.     Matriz A = NuevaMatriz();
17.     Matriz B = NuevaMatriz();
18.     Matriz C = NuevaMatriz();
19.     Matriz D = NuevaMatriz();
20.     Matriz E = NuevaMatriz();
21.     Matriz F = NuevaMatriz();
22.
23.     if(pipe(tuberial) != 0)
24.         exit(1);
25.     if(pipe(tuberia2) != 0)
26.         exit(1);
27.     if((pidpadre = fork()) == 0){
28.         if((pidnieto = fork()) == 0){ //nieto
29.             LeerPipe(tuberia2[0], D);
30.             LeerPipe(tuberia2[0], E);
31.             F = Suma(D, E);
32.             EscribirPipe(tuberial[1], F);
33.         }else{ //padre
34.             LeerPipe(tuberial[0], A);
35.             LeerPipe(tuberial[0], B);
36.             C = Multiplicacion(A, B);
37.
38.             D = GeneradorMatriz();
39.             E = GeneradorMatriz();
40.
41.             EscribirPipe(tuberia2[1], D);
42.             EscribirPipe(tuberia2[1], E);
43.             EscribirPipe(tuberial[1], C);
44.         }
45.     }else{
46.         A = GeneradorMatriz();
47.         B = GeneradorMatriz();
48.
49.         EscribirPipe(tuberial[1], A);
50.         EscribirPipe(tuberial[1], B);
51.
52.         wait(&status);
53.         LeerPipe(tuberial[0], C);
54.         printf("\nInversa de la multiplicacion de dos matrices\n");
55.         C = Inversa(C);
56.         EscribirMatriz(C, "inversaMultiplicacion.txt");
57.         printMatriz(C);
58.
59.         LeerPipe(tuberial[0], F);
60.         printf("\nInversa de la suma de dos matrices\n");
61.         F = Inversa(F);
62.         EscribirMatriz(F, "inversaSuma.txt");
63.         printMatriz(F);
64.     }
65. }
```

Código(EstructuraMatriz.h)

```
01. #include <stdbool.h>
02. #include <math.h>
03. #include <time.h>
04. typedef double* Vector;
05. typedef Vector* Matriz;
06. #define N 10
07.
08. Matriz NuevaMatriz(){
09.     int M = N;
10.     if(N%1)
11.         M++;
12.     Matriz A = calloc(M, sizeof(Vector));
13.     for(int i = 0; i < N; ++i)
14.         A[i] = calloc(M, sizeof(double));
15.     return A;
16. }
17.
18. bool esCero(double x){
19.     return fabs(x) < 1e-8;
20. }
21.
22. Matriz Suma(Matriz A, Matriz B){
23.     Matriz C = NuevaMatriz();
24.     for(int i = 0; i < N; ++i)
25.         for(int j = 0; j < N; ++j)
26.             C[i][j] = A[i][j] + B[i][j];
27.     return C;
28. }
29.
30.
31. Matriz Multiplicacion(Matriz A, Matriz B){
32.     Matriz C = NuevaMatriz();
33.     for(int i = 0; i < N; ++i)
34.         for(int j = 0; j < N; ++j)
35.             for(int k = 0; k < N; ++k)
36.                 C[i][j] += A[i][k] * B[k][j];
37.     return C;
38. }
39.
40. Matriz Inversa(Matriz A){
41.     Matriz inv = NuevaMatriz();
42.     for(int i = 0; i < N; ++i)
43.         inv[i][i] = 1;
44.     int i = 0, j = 0;
45.     while(i < N && j < N){
46.         if(esCero(A[i][j])){
47.             for(int k = i + 1; k < N; ++k){
48.                 if(!esCero(A[k][j])){
49.                     Vector tmp = A[i];
50.                     A[i] = A[k];
51.                     A[k] = tmp;
52.                     tmp = inv[i];
53.                     inv[i] = inv[k];
54.                     inv[k] = tmp;
55.                     break;
56.                 }
57.             }
58.         }
59.         if(!esCero(A[i][j])){
60.             for(int l = 0; l < N; ++l)
61.                 inv[i][l] /= A[i][j];
62.             for(int l = N - 1; l >= j; --l)
63.                 A[i][l] /= A[i][j];
64.             for(int k = 0; k < N; ++k){
65.                 if(i == k) continue;
66.                 for(int l = 0; l < N; ++l)
67.                     inv[k][l] -= inv[i][l] * A[k][j];
68.                 for(int l = N; l >= j; --l)
69.                     A[k][l] -= A[i][l] * A[k][j];
70.             }
71.             ++i;
72.         }
73.         ++j;
74.     }
75.     return inv;
76. }
77.
78. Matriz GeneradorMatriz(){
79.     srand(time(NULL));
80.     Matriz A = NuevaMatriz();
81.     for(int i = 0; i < N; ++i)
82.         for(int j = 0; j < N; ++j)
83.             A[i][j] = rand() % 10;
84.     return A;
85. }
86.
87. void EscribirMatriz(Matriz A, char* nombre){
88.     FILE * fp = fopen(nombre, "w");
89.     for(int i = 0; i < N; ++i){
90.         for(int j = 0; j < N; ++j)
91.             fprintf(fp, "%0.31f ", A[i][j]);
92.         fprintf(fp, "\n");
93.     }
94.     fclose(fp);
95. }
96.
97. void printMatriz(Matriz A){
98.     for(int i = 0; i < N; ++i){
99.         for(int j = 0; j < N; ++j)
100.            printf("%0.31f\t", A[i][j]);
101.        printf("\n");
102.    }
103. }
104.
105. void EscribirPipe(int tuberia, Matriz A){
106.     for(int i = 0; i < N; i++)
107.         for(int k = 0; k < N; k++)
108.             write(tuberia, &A[i][k], sizeof (double));
109. }
110.
111. void LeerPipe(int tuberia, Matriz A){
112.     for(int i = 0; i < N; i++)
113.         for(int k = 0; k < N; k++)
114.             read(tuberia, &A[i][k], sizeof (double));
115. }
```

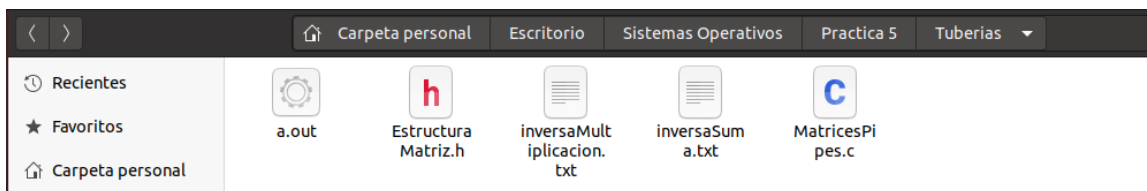
Compilación y ejecución del programa:

```
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 5/Tuberias$ gcc MatricesPipes.c
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 5/Tuberias$ ./a.out

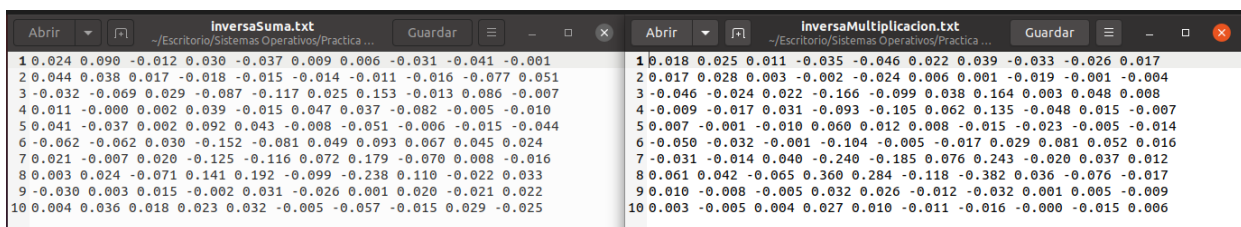
Inversa de la multiplicacion de dos matrices
0.018 0.025 0.011 -0.035 -0.046 0.022 0.039 -0.033 -0.026 0.017
0.017 0.028 0.003 -0.002 -0.024 0.006 0.001 -0.019 -0.001 -0.004
-0.046 -0.024 0.022 -0.166 -0.099 0.038 0.164 0.003 0.048 0.008
-0.009 -0.017 0.031 -0.093 -0.105 0.062 0.135 -0.048 0.015 -0.007
0.007 -0.001 -0.010 0.060 0.012 0.008 -0.015 -0.023 -0.005 -0.014
-0.050 -0.032 -0.001 -0.104 -0.005 -0.017 0.029 0.081 0.052 0.016
-0.031 -0.014 0.040 -0.240 -0.185 0.076 0.243 -0.020 0.037 0.012
0.061 0.042 -0.065 0.360 0.284 -0.118 -0.382 0.036 -0.076 -0.017
0.010 -0.008 -0.005 0.032 0.026 -0.012 -0.032 0.001 0.005 -0.009
0.003 -0.005 0.004 0.027 0.010 -0.011 -0.016 -0.000 -0.015 0.006

Inversa de la suma de dos matrices
0.024 0.090 -0.012 0.030 -0.037 0.009 0.006 -0.031 -0.041 -0.001
0.044 0.038 0.017 -0.018 -0.015 -0.014 -0.011 -0.016 -0.077 0.051
-0.032 -0.069 0.029 -0.087 -0.117 0.025 0.153 -0.013 0.086 -0.007
0.011 -0.000 0.002 0.039 -0.015 0.047 0.037 -0.082 -0.005 -0.010
0.041 -0.037 0.002 0.092 0.043 -0.008 -0.051 -0.006 -0.015 -0.044
-0.062 -0.062 0.030 -0.152 -0.081 0.049 0.093 0.067 0.045 0.024
0.021 -0.007 0.020 -0.125 -0.116 0.072 0.179 -0.070 0.008 -0.016
0.003 0.024 -0.071 0.141 0.192 -0.099 -0.238 0.110 -0.022 0.033
-0.030 0.003 0.015 -0.002 0.031 -0.026 0.001 0.020 -0.021 0.022
0.004 0.036 0.018 0.023 0.032 -0.005 -0.057 -0.015 0.029 -0.025
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 5/Tuberias$
```

Archivos generados en la carpeta raíz.



Archivos generados exitosamente con el mismo contenido que el impreso.



Mediante el uso de las llamadas al sistema pipe(), write() y read() nos ayuda a la manipulación de tuberías para el traspaso de datos entre procesos que en este caso serán datos de matrices que se manipularán. Donde la llamada al sistema pipe() crea la tubería, write() escribe los datos que se desea enviar en la entrada de la tubería y read() recibe los datos de la salida de la tubería. Se observa cómo se envía dos matrices entre tres procesos, el primer proceso (proceso abuelo) le manda dos matrices a su proceso hijo a través de una tubería y obtendrá la multiplicación de ambas matrices que a su vez mandara el resultado a su proceso padre, después este hijo creara un hijo de él y le mandara dos matrices a su hijo a

través de una tubería de las cuales obtendrá la suma de las dos matrices recibidas y mandara al primer proceso creado (proceso abuelo) el resultado de la suma. Finalmente, el primer proceso obtendrá la matriz inversa de la matriz resultante de la multiplicación y de la suma. Donde se va a escribir ambas inversas resultantes en un archivo .txt distinto, la matriz inversa de la multiplicación con el nombre “inversaMultiplicacion.txt” y la matriz inversa de la suma con el nombre “inversaSuma.txt”.

2.1.4. Ejemplo de memoria compartida en Linux

Código(EjemploMemoriaCliente.c)

```
01. #include <sys/types.h> /* Cliente de la memoria compartida */
02. #include <sys/ipc.h>
03. #include <sys/shm.h>
04. #include <stdio.h>
05. #include <stdlib.h>
06. #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
07. int main()
08. {
09.     int shmid;
10.     key_t llave;
11.     char *shm, *s;
12.     llave = 5678;
13.     if ((shmid = shmget(llave, TAM_MEM, 0666)) < 0)
14.     {
15.         perror("Error al obtener memoria compartida: shmget");
16.         exit(-1);
17.     }
18.     if ((shm = shmat(shmid, NULL, 0)) == (char *)-1)
19.     {
20.         perror("Error al enlazar la memoria compartida: shmat");
21.         exit(-1);
22.     }
23.     for (s = shm; *s != '\0'; s++)
24.         putchar(*s);
25.     putchar('\n');
26.     *shm = '***';
27.     exit(0);
28. }
```

Código(EjemploMemoriaServidor.c)

```
01. #include <sys/types.h> /* Servidor de la memoria compartida */
02. #include <sys/ipc.h> /* (ejecutar el servidor antes de ejecutar el cliente)*/
03. #include <sys/shm.h>
04. #include <stdio.h>
05. #include <stdlib.h>
06. #include <unistd.h>
07. #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
08. int main()
09. {
10.     char c;
11.     int shmid;
12.     key_t llave;
13.     char *shm, *s;
14.     llave = 5678;
15.     if ((shmid = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0)
16.     {
17.         perror("Error al obtener memoria compartida: shmget");
18.         exit(-1);
19.     }
20.     if ((shm = shmat(shmid, NULL, 0)) == (char *)-1)
21.     {
22.         perror("Error al enlazar la memoria compartida: shmat");
23.         exit(-1);
24.     }
25.     s = shm;
26.     for (c = 'a'; c <= 'z'; c++)
27.         *s++ = c;
28.     *s = '\0';
29.     while (*shm != '***')
30.         sleep(1);
31.     exit(0);
32. }
```

Compilación y ejecución del código.

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$ gcc EjemploMemoriaCliente.c -o Cliente
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$ gcc EjemploMemoriaServidor.c -o Servidor
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$ ./Servidor & ./Cliente
[18] 2327
abcdefghijklmnopqrstuvwxyz
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 5$
```

Con los siguientes códigos mostramos la función de la memoria compartida con las llamadas al sistema `shmget()` para obtener la memoria compartida y `shmat()` para enlazar con la memoria compartida. Finalmente observamos su funcionamiento a través de la impresión del abecedario, el cual lo escribe en la memoria compartida “EjemploMemoriaServidor.c” y lo lee e imprime “EjemploMemoriaCliente.c”.

2.1.5. Programa con memoria compartida en Linux (Usando Matrices)

Como se vuelve a usar de nueva manera el archivo “EstructuraMatriz.h” sin las funciones creadas para el desarrollo de la parte de tuberías, se decidió no volver a agregar el código, para ahorrar mas espacio en el documento y que no esté tan pesado.

Código(Memoria.c)

```
01. #include "EstructuraMatriz.h"
02. #define TAM_MEM 27
03. double *empezarMemoria(key_t llave){
04.     int shmid;
05.     double *shm;
06.     if((shmid = shmget(llave, TAM_MEM, IPC_CREAT|0666)) < 0){
07.         perror("Error al obtener memoria compartida: shmget");
08.         exit(-1);
09.     }
10.     if((shm = shmat(shmid, NULL, 0)) == (double *)-1){
11.         perror("Error al enlazar la memoria compartida: shmat");
12.         exit(-1);
13.     }
14.     return shm;
15. }
16.
17. void escribirMemoria(double *shm, Matriz A){
18.     for(int i = 0; i < N; i++)
19.         for(int k = 0; k < N; k++)
20.             *shm++ = A[i][k];
21. }
22.
23. Matriz MemoriaToMatriz(double *shm){
24.     Matriz A = NuevaMatriz();
25.     for(int i = 0; i < N; i++)
26.         for(int k = 0; k < N; k++)
27.             A[i][k] = *shm++;
28.     return A;
29. }
```

Código(PrincipalMatriz.c)

```
01. #include <sys/types.h>
02. #include <sys/ipc.h>
03. #include <sys/shm.h>
04. #include <sys/wait.h>
05. #include <unistd.h>
06. #include <stdlib.h>
07. #include <stdio.h>
08. #include "Memoria.c"
09. int main(){
10.     int status, status1;
11.     Matriz A, B, C, D, E, F;
12.     double *AuxMemoria1, *AuxMemoria2, *MultiplicacionMatrices, *SumaMatrices;
13.     key_t llave1 = 1000;
14.     key_t llave2 = 2000;
15.     key_t llave3 = 3000;
16.     key_t llave4 = 4000;
17.     AuxMemoria1 = empezarMemoria(llave1);
18.     AuxMemoria2 = empezarMemoria(llave2);
19.     MultiplicacionMatrices = empezarMemoria(llave3);
20.     SumaMatrices = empezarMemoria(llave4);
21.
22.     if(fork() == 0){
23.         if(fork() == 0){ //Nieta
24.             D = MemoriatoMatriz(AuxMemoria1);
25.             E = MemoriatoMatriz(AuxMemoria2);
26.             F = Suma(D, E);
27.             escribirMemoria(SumaMatrices, F);
28.         }else{ //Padre
29.             A = MemoriatoMatriz(AuxMemoria1);
30.             B = MemoriatoMatriz(AuxMemoria2);
31.             C = Multiplicacion(A, B);
32.             D = GeneradorMatriz();
33.             E = GeneradorMatriz();
34.
35.             escribirMemoria(AuxMemoria1, D);
36.             escribirMemoria(AuxMemoria2, E);
37.             escribirMemoria(MultiplicacionMatrices, C);
38.         }
39.     }else{ //Abuelo
40.         A = GeneradorMatriz();
41.         B = GeneradorMatriz();
42.         escribirMemoria(AuxMemoria1, A);
43.         escribirMemoria(AuxMemoria2, B);
44.
45.         wait(&status);
46.         printf("\nInversa de la multiplicacion de dos matrices\n");
47.         C = Inversa(MemoriatoMatriz(MultiplicacionMatrices));
48.         EscribirMatriz(C, "inversaMultiplicacionMemoria.txt");
49.         printMatriz(C);
50.
51.         printf("\nInversa de la suma de dos matrices\n");
52.         F = Inversa(MemoriatoMatriz(SumaMatrices));
53.         EscribirMatriz(F, "inversaSumaMemoria.txt");
54.         printMatriz(F);
55.     }
56.     exit(0);
57. }
```

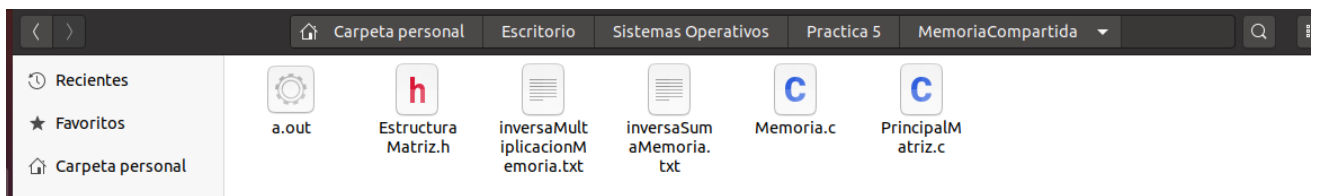
Compilación y ejecución del código.

```
edmundojasm@edmundojasm-VB:~/Escritorio/Sistemas Operativos/Practica 5/MemoriaCompartida$ gcc PrincipalMatriz.c
edmundojasm@edmundojasm-VB:~/Escritorio/Sistemas Operativos/Practica 5/MemoriaCompartida$ ./a.out

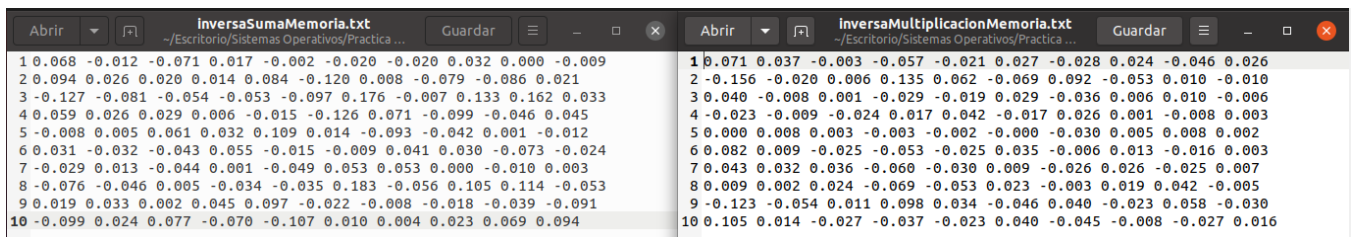
Inversa de la multiplicacion de dos matrices
0.071 0.037 -0.003 -0.057 -0.021 0.027 -0.028 0.024 -0.046 0.026
-0.156 -0.020 0.006 0.135 0.062 -0.069 0.092 -0.053 0.010 -0.010
0.040 -0.008 0.001 -0.029 -0.019 0.029 -0.036 0.006 0.010 -0.006
-0.023 -0.009 -0.024 0.017 0.042 -0.017 0.026 0.001 -0.008 0.003
0.000 0.008 0.003 -0.003 -0.002 -0.000 -0.030 0.005 0.008 0.002
0.082 0.009 -0.025 -0.053 -0.025 0.035 -0.006 0.013 -0.016 0.003
0.043 0.032 0.036 -0.060 -0.030 0.009 -0.026 0.026 -0.025 0.007
0.009 0.002 0.024 -0.069 -0.053 0.023 -0.003 0.019 0.042 -0.005
-0.123 -0.054 0.011 0.098 0.034 -0.046 0.040 -0.023 0.058 -0.030
0.105 0.014 -0.027 -0.037 -0.023 0.040 -0.045 -0.008 -0.027 0.016

Inversa de la suma de dos matrices
0.068 -0.012 -0.071 0.017 -0.002 -0.020 -0.020 0.032 0.000 -0.009
0.094 0.026 0.020 0.014 0.084 -0.120 0.008 -0.079 -0.086 0.021
-0.127 -0.081 -0.054 -0.053 -0.097 0.176 -0.007 0.133 0.162 0.033
0.059 0.026 0.029 0.006 -0.015 -0.126 0.071 -0.099 -0.046 0.045
-0.008 0.005 0.061 0.032 0.109 0.014 -0.093 -0.042 0.001 -0.012
0.031 -0.032 -0.043 0.055 -0.015 -0.009 0.041 0.030 -0.073 -0.024
-0.029 0.013 -0.044 0.001 -0.049 0.053 0.053 0.000 -0.010 0.003
-0.076 -0.046 0.005 -0.034 -0.035 0.183 -0.056 0.105 0.114 -0.053
0.019 0.033 0.002 0.045 0.097 -0.022 -0.008 -0.018 -0.039 -0.091
-0.099 0.024 0.077 -0.070 -0.107 0.010 0.004 0.023 0.069 0.094
```

Archivos generados en la carpeta raíz.



Archivos generados exitosamente con el mismo contenido que el impreso.



Tiene el mismo objetivo que el programa de “Programa con tuberías en Linux (Usando Matrices)”, a excepción de que la información se va transfiriendo por memoria compartida los datos de las matrices. La memoria compartida se crea a través de las llamadas al sistema `shmget()` para obtener el identificador del segmento de memoria compartida y `shmat()` adjuntar a la memoria compartida con el identificador generado por `shmget()` en donde esta se conectará entre procesos por medio de la llamada al sistema `fork()`.

2.2. Sección Windows

2.2.1. Ejemplo de tuberías en Windows

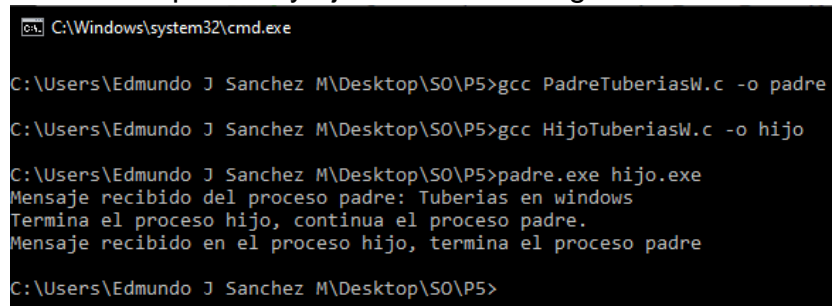
Código(PadreTuberiasW.c)

```
01. #include "windows.h"
02. #include "stdio.h"
03. #include "string.h"
04.
05. int main(int argc, char *argv[]){
06.     char mensaje[] = "Tuberías en windows";
07.     DWORD escritos;
08.     HANDLE hLecturaPipe, hEscrituraPipe;
09.     PROCESS_INFORMATION piHijo;
10.     STARTUPINFO siHijo;
11.     SECURITY_ATTRIBUTES pipeSeg = {sizeof(SEcurity_ATTRIBUTES), NULL, TRUE};
12.     GetStartupInfo(&siHijo);
13.     CreatePipe(&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);
14.     WriteFile(hEscrituraPipe, mensaje, strlen(mensaje) + 1, &escritos, NULL);
15.     siHijo.hStdInput = hLecturaPipe;
16.     siHijo.hStdError = GetStdHandle(STD_ERROR_HANDLE);
17.     siHijo.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
18.     siHijo.dwFlags = STARTF_USESTDHANDLES;
19.     CreateProcess(NULL, argv[1], NULL, NULL, TRUE, 0, NULL, NULL, &siHijo, &piHijo);
20.     WaitForSingleObject(piHijo.hProcess, INFINITE);
21.     printf("Mensaje recibido en el proceso hijo, termina el proceso padre\n");
22.     CloseHandle(hLecturaPipe);
23.     CloseHandle(hEscrituraPipe);
24.     CloseHandle(piHijo.hThread);
25.     CloseHandle(piHijo.hProcess);
26.     return 0;
27. }
```

Código(HijoTuberiasW.c)

```
01. #include "windows.h"
02. #include "stdio.h"
03.
04. int main(){
05.     char mensaje[20];
06.     DWORD leidos;
07.     HANDLE hStdIn = GetStdHandle(STD_INPUT_HANDLE);
08.     SECURITY_ATTRIBUTES pipeSeg = {sizeof(SEcurity_ATTRIBUTES), NULL, TRUE};
09.     ReadFile(hStdIn, mensaje, sizeof(mensaje), &leidos, NULL);
10.     printf("Mensaje recibido del proceso padre: %s\n", mensaje);
11.     CloseHandle(hStdIn);
12.     printf("Termina el proceso hijo, continua el proceso padre.\n");
13.     return 0;
14. }
```

Compilación y ejecución del código.



```
C:\Windows\system32\cmd.exe
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>gcc PadreTuberiasW.c -o padre
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>gcc HijoTuberiasW.c -o hijo
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>padre.exe hijo.exe
Mensaje recibido del proceso padre: Tuberias en windows
Termina el proceso hijo, continua el proceso padre.
Mensaje recibido en el proceso hijo, termina el proceso padre
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>
```

En el programa PadreTuberiasW.c se crea un proceso que ejecuta el programa HijoTuberiasW.c. Para que se puedan comunicar es necesario la creación de la tubería mediante función: CreatePipe(), que mediante una serie de HANDLE como argumentos se pueden comunicar, aunque sean programas distintos.

2.2.2. Programa con tuberías en Windows (Usando Matrices)

Código(MatricesPipes.c)

```
01. #include <windows.h>
02. #include <stdio.h>
03. #include <time.h>
04. #include "EstructuraMatriz.h"
05.
06. HANDLE proceso(char *name, HANDLE hRead, int nivel){
07.     STARTUPINFO si;
08.     PROCESS_INFORMATION pi;
09.     ZeroMemory(&pi, sizeof(pi));
10.     ZeroMemory(&si, sizeof(si));
11.     GetStartupInfo(&si);
12.     si.hStdInput = hRead;
13.     si.hStdError = GetStdHandle(STD_ERROR_HANDLE);
14.     si.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
15.     si.dwFlags = STARTF_USESTDHANDLES;
16.     si.cb = sizeof(si);
17.     char args[100];
18.     sprintf(args, "%s %d", name, nivel);
19.     CreateProcess(NULL, args, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi);
20.     return pi.hProcess;
21. }
22.
23. void creaTuberia(HANDLE *hRead, HANDLE *hWrite){
24.     SECURITY_ATTRIBUTES pipeSeg = {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};
25.     CreatePipe(hRead, hWrite, &pipeSeg, 0);
26. }
27.
28. void escribir(Matriz A, HANDLE hWrite){
29.     for(int i = 0; i < N; ++i)
30.         for(int j = 0; j < N; ++j)
31.             WriteFile(hWrite, &A[i][j], sizeof(double), NULL, NULL);
32. }
33.
34. void leer(Matriz A, HANDLE hRead){
35.     for(int i = 0; i < N; ++i)
36.         for(int j = 0; j < N; ++j)
37.             ReadFile(hRead, &A[i][j], sizeof(double), NULL, NULL);
38. }
39.
40. int main(int argc, char *argv[]){
41.     srand(time(NULL));
42.     int nivel = 0; //0-padre, 1-hijo, 2-nieto
43.     HANDLE hRead, hWrite;
44.     if(argc > 1)
45.         sscanf(argv[1], "%d", &nivel);
46.     if(nivel == 0){ //padre
47.         creaTuberia(&hRead, &hWrite);
48.         Matriz A = GeneradorMatriz();
49.         Matriz B = GeneradorMatriz();
50.         escribir(A, hWrite);
51.         escribir(B, hWrite);
52.         WriteFile(hWrite, &hWrite, sizeof(HANDLE), NULL, NULL);
53.         HANDLE hProc = proceso(argv[0], hRead, 1);
54.         WaitForSingleObject(hProc, INFINITE);
55.         Matriz AB = NuevaMatriz();
56.         leer(AB, hRead);
57.         Matriz C_D = NuevaMatriz();
58.         leer(C_D, hRead);
59.
60.         Matriz AB_inv = Inversa(AB);
61.         Matriz C_D_inv = Inversa(C_D);
62.         EscribirMatriz(AB_inv, "inversaMultiplicacion.txt");
63.         EscribirMatriz(C_D_inv, "inversaSuma.txt");
64.         printf("Inversa de la multiplicacion de dos matrices:\n");
65.         printMatriz(AB_inv);
66.         printf("Inversa de la suma de dos matrices:\n");
67.         printMatriz(C_D_inv);
    }
```

```

68.     }else if(nivel == 1){ //hijo
69.         hRead = GetStdHandle(STD_INPUT_HANDLE);
70.         Matriz A = NuevaMatriz();
71.         Matriz B = NuevaMatriz();
72.         leer(A, hRead);
73.         leer(B, hRead);
74.         Matriz AB = Multiplicacion(A, B);
75.         ReadFile(hRead, &hWrite, sizeof(HANDLE), NULL, NULL);
76.         escribir(AB, hWrite);
77.
78.         HANDLE hRead2, hWrite2;
79.         creaTuberia(&hRead2, &hWrite2);
80.         Matriz C = GeneradorMatriz();
81.         Matriz D = GeneradorMatriz();
82.         escribir(C, hWrite2);
83.         escribir(D, hWrite2);
84.         WriteFile(hWrite2, &hWrite, sizeof(HANDLE), NULL, NULL);
85.         HANDLE hProc = proceso(argv[0], hRead2, 2);
86.         WaitForSingleObject(hProc, INFINITE);
87.     }else if(nivel == 2){ //nieto
88.         hRead = GetStdHandle(STD_INPUT_HANDLE);
89.         Matriz C = NuevaMatriz();
90.         Matriz D = NuevaMatriz();
91.         leer(C, hRead);
92.         leer(D, hRead);
93.         ReadFile(hRead, &hWrite, sizeof(HANDLE), NULL, NULL);
94.         Matriz C_D = Suma(C, D);
95.         escribir(C_D, hWrite);
96.     }
97.     return 0;
98. }

```

Código(EstructuraMatriz.h)

```

01. #include <stdbool.h>
02. #include <math.h>
03. #include <time.h>
04. typedef double* Vector;
05. typedef Vector* Matriz;
06. #define N 10
07.
08. Matriz NuevaMatriz(){
09.     int M = N;
10.     if(N%1)
11.         M+=1;
12.     Matriz A = calloc(M, sizeof(Vector));
13.     for(int i = 0; i < N; ++i)
14.         A[i] = calloc(M, sizeof(double));
15.     return A;
16. }
17.
18. bool esCero(double x){
19.     return fabs(x) < 1e-8;
20. }
21.
22. Matriz Suma(Matriz A, Matriz B){
23.     Matriz C = NuevaMatriz();
24.     for(int i = 0; i < N; ++i)
25.         for(int j = 0; j < N; ++j)
26.             C[i][j] = A[i][j] + B[i][j];
27.     return C;
28. }
29.
30. Matriz Multiplicacion(Matriz A, Matriz B){
31.     Matriz C = NuevaMatriz();
32.     for(int i = 0; i < N; ++i)
33.         for(int j = 0; j < N; ++j)
34.             for(int k = 0; k < N; ++k)
35.                 C[i][j] += A[i][k] * B[k][j];
36.     return C;
37. }
38.
39.
40. Matriz Inversa(Matriz A){
41.     Matriz inv = NuevaMatriz();
42.     for(int i = 0; i < N; ++i)
43.         inv[i][i] = 1;
44.     int i = 0, j = 0;
45.     while(i < N && j < N){
46.         if(esCero(A[i][j])){
47.             for(int k = i + 1; k < N; ++k){
48.                 if(!esCero(A[k][j])){
49.                     Vector tmp = A[i];
50.                     A[i] = A[k];
51.                     A[k] = tmp;
52.                     tmp = inv[i];
53.                     inv[i] = inv[k];
54.                     inv[k] = tmp;
55.                     break;
56.                 }
57.             }
58.         }
59.         if(!esCero(A[i][j])){
60.             for(int l = 0; l < N; ++l)
61.                 inv[i][l] /= A[i][j];
62.             for(int l = N - 1; l >= j; --l)
63.                 A[i][l] /= A[i][j];
64.             for(int k = 0; k < N; ++k){
65.                 if(i == k) continue;
66.                 for(int l = 0; l < N; ++l)
67.                     inv[k][l] -= inv[i][l] * A[k][j];
68.                 for(int l = N; l >= j; --l)
69.                     A[k][l] -= A[i][l] * A[k][j];
70.             }
71.             ++i;
72.         }
73.         ++j;
74.     }
75.     return inv;
76. }
77.

```



```

78. Matriz GeneradorMatriz(){
79.     srand(time(NULL));
80.     Matriz A = NuevaMatriz();
81.     for(int i = 0; i < N; ++i)
82.         for(int j = 0; j < N; ++j)
83.             A[i][j] = rand() % 10;
84.     return A;
85. }
86.
87. void EscribirMatriz(Matriz A, char* nombre){
88.     FILE * fp = fopen(nombre, "w");
89.     for(int i = 0; i < N; ++i){
90.         for(int j = 0; j < N; ++j)
91.             fprintf(fp, "%0.3lf ", A[i][j]);
92.         fprintf(fp, "\n");
93.     }
94.     fclose(fp);
95. }
96.
97. void printMatriz(Matriz A){
98.     for(int i = 0; i < N; ++i){
99.         for(int j = 0; j < N; ++j)
100.            printf("%0.3lf\t", A[i][j]);
101.        printf("\n");
102.    }
103. }

```

Compilación y ejecución del programa:

```






C:\Users\Edmundo J Sanchez M\Desktop\SO\P5\Tuberias>gcc MatricesPipes.c

C:\Users\Edmundo J Sanchez M\Desktop\SO\P5\Tuberias>a.exe
Inversa de la multiplicacion de dos matrices:
0.041  -0.028  -0.097  0.166  -0.065  -0.203  0.050  -0.018  0.159  -0.056
0.005  0.009  -0.043  0.058  -0.035  -0.068  0.010  0.005  0.073  -0.013
-0.025  -0.024  0.039  0.026  0.008  -0.012  0.003  -0.045  -0.012  0.006
-0.008  -0.023  -0.010  0.079  0.005  -0.052  0.058  -0.058  -0.036  -0.032
-0.029  0.014  0.066  -0.092  0.072  0.134  -0.019  -0.024  -0.148  0.023
0.010  0.010  0.017  -0.014  0.004  0.014  -0.039  0.013  0.003  0.003
-0.022  0.034  0.083  -0.151  0.012  0.179  -0.091  0.053  -0.046  0.059
-0.011  -0.028  -0.018  0.050  -0.019  -0.025  0.013  0.010  0.018  -0.001
0.011  0.031  -0.007  -0.106  0.012  0.074  -0.010  0.046  -0.033  0.037
0.014  -0.007  -0.045  0.047  -0.006  -0.086  0.050  -0.005  0.034  -0.034
Inversa de la suma de dos matrices:
0.066  0.080  -0.095  -0.198  0.089  0.039  0.082  0.035  -0.058  -0.007
-0.012  0.087  -0.055  -0.029  -0.003  0.010  0.026  0.023  0.013  -0.017
-0.051  -0.010  0.031  0.005  0.006  0.120  -0.015  0.044  -0.061  -0.020
-0.036  0.056  0.125  -0.186  0.091  0.159  -0.055  -0.035  -0.133  0.055
-0.040  -0.044  0.185  0.059  -0.006  0.039  -0.092  -0.124  -0.011  0.012
0.021  -0.017  -0.043  0.049  0.019  -0.122  0.046  -0.069  0.033  0.013
0.002  -0.065  -0.062  0.240  -0.078  -0.189  -0.030  0.033  0.188  -0.050
-0.042  -0.023  0.049  -0.008  -0.046  0.079  0.001  0.039  0.007  0.000
0.016  -0.035  -0.048  0.119  -0.082  -0.058  -0.029  0.026  0.118  0.011
0.031  0.033  -0.028  -0.099  0.026  0.050  0.059  0.046  -0.128  0.021

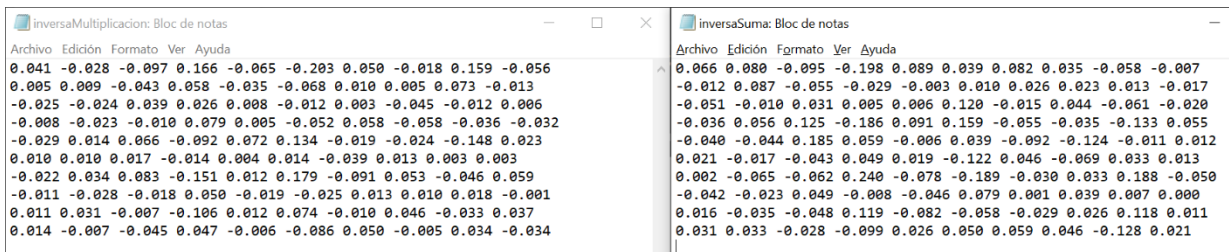
C:\Users\Edmundo J Sanchez M\Desktop\SO\P5\Tuberias>

```

Archivos generados en la carpeta raíz.

te equipo > Escritorio > SO > P5 > Tuberias	
Nombre	Fecha de modificación
 a	02/12/2020 08:00 a. m.
 EstructuraMatriz	01/12/2020 09:15 p. m.
 inversaMultiplicacion	02/12/2020 08:00 a. m.
 inversaSuma	02/12/2020 08:00 a. m.
 MatricesPipes	02/12/2020 07:59 a. m.

Archivos generados exitosamente con el mismo contenido que el impreso.



La aplicación es de forma similar a la del punto anterior, sin embargo, debido a que en Windows los procesos creados no comparten las variables, tuvimos que enviar como tal el HANDLE de las tuberías del lado de la escritura por la misma tubería para poder escribir en ella desde el proceso hijo. El programa recibirá un argumento por consola que indicara si es el proceso abuelo, padre o hijo (nivel 0, 1 y 2 respectivamente) esto enviado por el mismo programa.

2.2.3. Ejemplo de memoria compartida en Windows

Código(EjemploMemoriaClienteW.c)

```
01. #include <windows.h> /* Cliente de la memoria compartida */
02. #include <stdio.h>
03. #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
04. int main(void){
05.     HANDLE hArchMapeo;
06.     char *idMemCompartida = "MemoriaCompartida";
07.     char *apDatos, *apTrabajo, c;
08.     if ((hArchMapeo = OpenFileMapping(
09.         FILE_MAP_ALL_ACCESS, // acceso lectura/escritura de la memoria compartida
10.         FALSE, // no se hereda el nombre
11.         idMemCompartida) // identificador de la memoria compartida
12.     ) == NULL){
13.         printf("No se abrio archivo de mapeo de la memoria compartida: (%i)\n", GetLastError());
14.         exit(-1);
15.     }
16.     if ((apDatos = (char *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
17.         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
18.         0,
19.         0,
20.         TAM_MEM)) == NULL){
21.         printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
22.         CloseHandle(hArchMapeo);
23.         exit(-1);
24.     }
25.     for (apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++)
26.         putchar(*apTrabajo);
27.     putchar('\n');
28.     *apDatos = '*';
29.     UnmapViewOfFile(apDatos);
30.     CloseHandle(hArchMapeo);
31.     exit(0);
32. }
```

Código(EjemploMemoriaServidorW.c)

```
01. #include <windows.h> /* Servidor de la memoria compartida */
02. #include <stdio.h> /* (ejecutar el servidor antes de ejecutar el cliente)*/
03. #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
04. int main(void)
05. {
06.     HANDLE hArchMapeo;
07.     char *idMemCompartida = "MemoriaCompartida";
08.     char *apDatos, *apTrabajo, c;
09.     if ((hArchMapeo = CreateFileMapping(
10.         INVALID_HANDLE_VALUE, // usa memoria compartida
11.         NULL, // seguridad por default
12.         PAGE_READWRITE, // acceso lectura/escritura a la memoria
13.         0, // tamaño maximo parte alta de un DWORD
14.         TAM_MEM, // tamaño maximo parte baja de un DWORD
15.         idMemCompartida) // identificador de la memoria compartida
16.     ) == NULL){
17.         printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
18.         exit(-1);
19.     }
20.     if ((apDatos = (char *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
21.         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
22.         0,
23.         0,
24.         TAM_MEM)) == NULL){
25.         printf("No se creo la memoria compartida: (%i)\n", GetLastError());
26.         CloseHandle(hArchMapeo);
27.         exit(-1);
28.     }
29.     apTrabajo = apDatos;
30.     for (c = 'a'; c <= 'z'; c++)
31.         *apTrabajo++ = c;
32.     *apTrabajo = '\0';
33.     while (*apDatos != '*')
34.         sleep(1);
35.     UnmapViewOfFile(apDatos);
36.     CloseHandle(hArchMapeo);
37.     exit(0);
38. }
```

Compilación y ejecución del código, hay que mencionar que hay que ejecutar en do consolas separadas, una se usara para el servidor y otra para el cliente.

```
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>gcc EjemploMemoriaServidorW.c -o Servidor
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>Servidor.exe
```

Compilación y ejecución del servidor de nuestro programa.

```
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>gcc EjemploMemoriaClienteW.c -o Cliente
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>Cliente.exe
abcdefghijklmnopqrstuvwxyz
C:\Users\Edmundo J Sanchez M\Desktop\S0\P5>
```

Compilación y ejecución del servidor de nuestro programa.

A diferencia de las tuberías, aquí no es necesario crear procesos para que envíe la información a comunicar. Para que algún programa pueda acceder a la memoria compartida, donde se guarda la información, debe tener el mismo ID (clave) del programa que creo la memoria compartida, mencionar que el programa servidor se detiene cuando el cliente termina de ejecutarse.

2.2.4. Programa con memoria compartida en Windows (Usando Matrices)

Usando de nueva manera el archivo “EstructuraMatriz.h”, tenemos el siguiente código para el programa.

Código(MemoriaCompartidaW.c)

```
01. #include <windows.h>
02. #include <stdio.h>
03. #include <time.h>
04. #include "EstructuraMatriz.h"
05. #define TAM_MEM ((3 * N * N + 10) * sizeof(double))
06.
07. double inf = 1e9;
08.
09. void proceso(char *name, int nivel){
10.     STARTUPINFO si;
11.     PROCESS_INFORMATION pi;
12.     ZeroMemory(&si, sizeof(si));
13.     si.cb = sizeof(si);
14.     ZeroMemory(&pi, sizeof(pi));
15.     char args[100];
16.     sprintf(args, "%s %d", name, nivel);
17.     CreateProcess(NULL, args, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
18. }
19.
20. void escribir(Matriz A, double **p){
21.     for(int i = 0; i < N; ++i){
22.         for(int j = 0; j < N; ++j){
23.             **p = A[i][j];
24.             (*p)++;
25.         }
26.     }
27. }
28.
29. void leer(Matriz A, double **p){
30.     for(int i = 0; i < N; ++i){
31.         for(int j = 0; j < N; ++j){
32.             A[i][j] = **p;
33.             (*p)++;
34.         }
35.     }
36. }
37.
38. double *crearMemoria(char *idMem){
39.     HANDLE hMem = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, TAM_MEM, idMem);
40.     return MapViewOfFile(hMem, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM);
41. }
42.
43. double *leerMemoria(char *idMem){
44.     HANDLE hMem = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, idMem);
45.     return MapViewOfFile(hMem, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM);
46. }
47.
48. int main(int argc, char *argv[]){
49.     srand(time(NULL));
50.     int nivel = 0; //0-padre, 1-hijo, 2-nieto
51.     if(argc > 1)
52.         sscanf(argv[1], "%d", &nivel);
53.     if(nivel == 0){ //padre
54.         double *p = crearMemoria("Producto AB");
55.         Matriz A = GeneradorMatriz();
56.         Matriz B = GeneradorMatriz();
57.         escribir(A, &p);
58.         escribir(B, &p);
59.         proceso(argv[0], 1);
60.         while(*p != inf)
61.             Sleep(1);
62.         *p++;
    }
```

```

63.     Matriz AB = NuevaMatriz();
64.     leer(AB, &p);
65.     while(*p != inf)
66.         Sleep(1);
67.     double *p2 = leerMemoria("Suma C+D");
68.     Matriz C_D = NuevaMatriz();
69.     leer(C_D, &p2);
70.     leer(C_D, &p2);
71.     *p2++;
72.     leer(C_D, &p2);
73.     *p2++ = inf;
74.
75.     Matriz AB_inv = Inversa(AB);
76.     Matriz C_D_inv = Inversa(C_D);
77.     EscribirMatriz(AB_inv, "inversaMultiplicacion.txt");
78.     EscribirMatriz(C_D_inv, "inversaSuma.txt");
79.     printf("Inversa de la multiplicacion de dos matrices:\n");
80.     printMatriz(AB_inv);
81.     printf("Inversa de la suma de dos matrices:\n");
82.     printMatriz(C_D_inv);
83. }else if(nivel == 1){ //hijo
84.     double *p = leerMemoria("Producto AB");
85.     Matriz A = NuevaMatriz();
86.     Matriz B = NuevaMatriz();
87.     leer(A, &p);
88.     leer(B, &p);
89.     Matriz AB = Multiplicacion(A, B);
90.     *p++ = inf;
91.     escribir(AB, &p);
92.
93.     double *p2 = crearMemoria("Suma C+D");
94.     Matriz C = GeneradorMatriz();
95.     Matriz D = GeneradorMatriz();
96.     escribir(C, &p2);
97.     escribir(D, &p2);
98.     proceso(argv[0], 2);
99.     while(*p2 != inf)
100.         Sleep(1);
101.     *p++ = inf;
102. }else if(nivel == 2){ //nieto
103.     double *p = leerMemoria("Suma C+D");
104.     Matriz C = NuevaMatriz();
105.     Matriz D = NuevaMatriz();
106.     leer(C, &p);
107.     leer(D, &p);
108.     Matriz C_D = Suma(C, D);
109.     *p++ = inf;
110.     escribir(C_D, &p);
111.     while(*p != inf)
112.         Sleep(1);
113. }
114. return 0;
115. }

```

Compilación y ejecución del programa:

```

C:\Users\Edmundo J Sanchez M\Desktop\SO\PS\MemoriaCompartida>gcc MemoriaCompartidaW.c
C:\Users\Edmundo J Sanchez M\Desktop\SO\PS\MemoriaCompartida>a.exe
Inversa de la multiplicacion de dos matrices:
0.019  0.013  -0.013  -0.019  0.023  0.012  -0.011  -0.017  -0.022  0.011
0.008  0.073  -0.094  0.035  -0.037  -0.004  0.032  -0.060  0.057  -0.021
0.030  0.056  -0.074  0.010  -0.039  0.007  0.020  -0.012  0.032  -0.030
0.004  0.001  -0.001  -0.000  0.012  0.024  -0.005  -0.001  -0.028  -0.005
-0.025  -0.067  0.090  -0.016  0.018  -0.013  -0.009  0.044  -0.029  0.018
0.018  -0.024  0.031  -0.004  0.004  0.001  -0.027  0.030  -0.020  0.004
-0.007  0.039  -0.042  -0.016  -0.006  0.007  0.015  -0.006  -0.002  0.011
-0.011  -0.047  0.061  0.003  0.018  -0.020  -0.019  0.040  -0.005  -0.007
0.001  0.001  -0.001  0.008  0.001  0.009  0.001  -0.026  0.012  -0.008
-0.022  -0.009  -0.008  0.017  -0.015  -0.014  0.018  -0.019  0.033  0.009
Inversa de la suma de dos matrices:
-0.045  -0.024  0.071  0.017  0.001  -0.064  -0.007  0.062  0.041  -0.017
0.045  0.180  -0.223  -0.074  -0.006  0.065  0.043  -0.103  -0.017  0.049
0.031  0.134  -0.129  -0.105  0.014  0.076  0.023  -0.016  -0.087  0.047
-0.065  -0.091  0.076  0.013  0.009  -0.007  -0.013  -0.009  0.018  0.059
0.001  -0.102  0.112  0.051  0.011  -0.011  0.002  0.010  0.043  -0.086
-0.036  -0.076  0.052  0.002  -0.039  -0.009  0.026  0.096  -0.023  0.021
0.042  0.042  0.002  -0.006  -0.013  0.020  -0.014  -0.042  -0.008  0.001
0.026  -0.077  0.068  0.016  0.008  -0.036  -0.039  0.058  0.011  -0.008
-0.013  0.015  -0.018  0.024  0.036  -0.022  0.014  -0.031  -0.011  0.009
0.023  0.098  -0.113  0.025  0.005  0.040  -0.003  -0.052  0.005  -0.026
C:\Users\Edmundo J Sanchez M\Desktop\SO\PS\MemoriaCompartida>

```

Archivos generados en la carpeta raíz.

te equipo > Escritorio > SO > P5 > MemoriaCompartida	
Nombre	Fecha de modificación
a	02/12/2020 05:29 p. m.
EstructuraMatriz	01/12/2020 09:15 p. m.
inversaMultiplicacion	02/12/2020 05:29 p. m.
inversaSuma	02/12/2020 05:29 p. m.
MemoriaCompartidaW	02/12/2020 05:26 p. m.

Archivos generados exitosamente con el mismo contenido que el impreso.

inversaMultiplicacion: Bloc de notas	
Archivo Edición Formato Ver Ayuda	
0.019 0.013 -0.013 -0.019 0.023 0.012 -0.011 -0.017 -0.022 0.011	
0.008 0.073 -0.094 0.035 -0.037 -0.004 0.032 -0.060 0.057 -0.021	
0.030 0.056 -0.074 0.010 -0.039 0.007 0.020 -0.012 0.032 -0.030	
0.004 0.001 -0.001 -0.000 0.012 0.024 -0.005 -0.001 -0.028 -0.005	
-0.025 -0.067 0.090 -0.016 0.018 -0.013 -0.009 0.044 -0.029 0.018	
0.018 -0.024 0.031 -0.004 0.004 0.001 -0.027 0.030 -0.020 0.004	
-0.007 0.039 -0.042 -0.016 -0.006 0.007 0.015 -0.006 -0.002 0.011	
-0.011 -0.047 0.061 0.003 0.018 -0.020 -0.019 0.040 -0.005 -0.007	
0.001 0.001 -0.001 0.008 0.001 0.009 0.001 -0.026 0.012 -0.008	
-0.022 -0.009 -0.008 0.017 -0.015 -0.014 0.018 -0.019 0.033 0.009	

inversaSuma: Bloc de notas	
Archivo Edición Formato Ver Ayuda	
-0.045 -0.024 0.071 0.017 0.001 -0.064 -0.007 0.062 0.041 -0.017	
0.045 0.180 -0.223 -0.074 -0.006 0.065 0.043 -0.103 -0.017 0.049	
0.031 0.134 -0.129 -0.105 0.014 0.076 0.023 -0.016 -0.087 0.047	
-0.065 -0.091 0.076 0.013 0.009 -0.007 -0.013 -0.009 0.018 0.059	
0.001 -0.102 0.112 0.051 0.011 -0.011 0.002 0.010 0.043 -0.086	
-0.036 -0.076 0.052 0.002 -0.039 -0.009 0.026 0.096 -0.023 0.021	
0.042 -0.042 0.002 -0.006 -0.013 0.020 -0.014 -0.042 -0.008 0.001	
0.026 -0.077 0.068 0.016 0.008 -0.036 -0.039 0.058 0.011 -0.008	
-0.013 0.015 -0.018 0.024 0.036 -0.022 0.014 -0.031 -0.011 0.009	
0.023 0.098 -0.113 0.025 0.005 0.040 -0.003 -0.052 0.005 -0.026	

Aquí se complicaron las cosas, porque cuando el proceso padre que crea la memoria compartida finaliza, también lo hacen sus memorias compartidas, por lo que tuvimos que simular una pequeña sincronización.

El programa recibirá un argumento por consola que indicara si es el proceso abuelo, padre o hijo (nivel 0, 1 y 2 respectivamente). Iniciando en el nivel 0 haremos lo siguiente:

1. Crearemos una memoria compartida llamada Producto AB con tamaño para tres matrices. Mandaremos dos matrices aleatorias A y B y crearemos un nuevo proceso con nivel 1. Luego, tendremos un while esperando a que el proceso de nivel 1 ya tenga listo el producto AB, usando un valor especial de infinito ∞ como carácter de control ($\approx 10^9$).
2. Al comienzo del proceso de nivel 1 accederemos a la memoria compartida Producto AB, leemos A y B, calculamos el producto AB, escribimos el carácter de control ∞ e inmediatamente escribimos la matriz AB.
3. Lo anterior provoca que el proceso de nivel 0 se desbloquee, entonces leemos la matriz AB que nos acaba de enviar el nivel 1 y de nuevo volvemos a esperar por el carácter de control.
4. De vuelta en el nivel 1 creamos otra memoria compartida llamada Suma C+D, creamos dos matrices aleatorias C y D y las mandamos por ahí; creamos el proceso de nivel 2 y nos ponemos en espera por el carácter de control y por el resultado de C + D.

5. En el nivel 2 accedemos a la memoria compartida Suma C+D, leemos C y D, hallamos $C + D$, escribimos el carácter de control e inmediatamente escribimos la matriz $C + D$.
6. Con la acción anterior regresamos al nivel 1, el cual se desbloquea y manda el carácter de control por la memoria compartida Producto AB. Aquí termina el proceso de nivel 1.
7. De esa forma, regresamos al nivel 0, el cual se vuelve a desbloquear. Imprimimos el resultado de AB y accedemos a la memoria compartida Suma C+D, avanzamos el puntero dos matrices, un carácter de control (recordemos que estas matrices eran C y D), y ahora si ya leemos la matriz $C + D$, mandando un carácter de control inmediatamente después.
8. Así, el proceso de nivel 2 se desbloquea y finaliza. 9. De regreso al nivel 0, calculamos $(AB) - 1$ y $(C+D) - 1$, las guardamos en un archivo y las imprimimos. Aquí termina el proceso de nivel 0.

3. Análisis de la práctica

3.1 Linux

Usamos las llamadas al sistema para la creación y manipulación, se pueden crear varios al mismo momento. Las funciones de las llamadas al sistema de las tuberías son:

- Crear una tubería o pipe con `pipe()`.
- Para identificar a la tubería es por medio de su nombre.
- Escribir datos en la tubería con `write()`.
- Obtener o leer los datos de una tubería con `read()`

Si se quiere volver a ocupar los mismos datos de una tubería se tiene que volver a escribir en ella los datos, ya que una vez obtenidos los datos de la tubería con la llamada al sistema `read()` se borran.

Por otra parte, las llamadas al sistema para la creación y manipulación de memorias compartidas, de las cuales se pueden crear varias en el mismo código. El funcionamiento que se observa de las llamadas al sistema es:

- Obtener el identificador del segmento de memoria compartida con `shmget()`.
- Adjuntar la memoria compartida con el identificador por medio de `shmat()`.
- Se escribe en el segmento de memoria compartida al igual que en un apuntador.
- Se obtienen los datos igual que un apuntador.

Cuando un proceso escribe en una memoria compartida se puede hacer uso de estos datos varias veces, los datos permanecen estáticos.

Además, realizar la aplicación de memoria compartida fue mucho más fácil en Linux que en Windows, porque la creación de procesos por copia exacta nos permite

que la memoria compartida permanezca aun cuando finalizan dichos procesos, en cambio en Windows al finalizar los procesos creados se borran las memorias compartidas que había creado.

3.2 Windows

Tanto para creación y manejo de tuberías se deben ocupar llamadas al sistema; también se usan nuevos parámetros, tales como:

- CreatePipe(): Crea la tubería que se necesita ocupar.
- WriteFile(): Permite la escritura en la tubería que se creó.
- ReadFile(): Permite la lectura de la tubería que se mandó al proceso hijo dentro del mismo proceso hijo.
- GetStdHandle(): Retorna un HANDLE hacia un “dispositivo” dependiendo de qué tipo le especifiquemos como argumento.
- SECURITY_ATTRIBUTES: Es una estructura donde lo más importante que se especifica es si se hereda, de un proceso padre al proceso hijo, los HANDLE (manejadores) con los que se crea la tubería (hLecturaPipe, hEscrituraPipe) para que el proceso hijo pueda manipular la tubería.

4. Observaciones

4.1. Linux

- Para ingresar y extraer los datos de una tubería son diferentes los accesos para cada una, ya que cuando se crea por medio de la llamada al sistema pipe que recibe un arreglo de tamaño dos y uno representa la entrada y otro la salida.
- La manipulación de una tubería hace uso de las llamadas al sistema pipe () para crearla, write() para escribir en ella y read() para leer sus datos.
- Se puede hacer uso de varios tubería en un código, no tiene un límite de invocación solo que tiene que ser distinto el nombre de la tubería.
- El identificador de la memoria compartida son las llaves key_t.
- La forma de manipular una memoria compartida y una tubería son similares.
- Para acceder a los datos de la tubería y de la memoria es por medio de su nombre.
- Se debe tener cuidado con la llave key_t a la hora de crear y acceder a los datos de la memoria compartida ya que puede mandar error o acceder a otra memoria diferente que la deseada.
- Con la llamada al sistema shmget() obtenemos un identificador para el segmento de la memoria compartida y shmat() para enlazar la memoria compartida con el identificador dado por shmget().
- Tanto en tuberías como en memoria compartida se tiene que copiar dato por dato, pero a la hora de obtener el dato estos tienen diferente comportamiento, por lo que hay que hacer un cast.

- La memoria compartida y tubería puede ser creada con la mayoría de los tipos de datos.
- Al igual que la tubería, se puede hacer uso de varias memorias compartidas al mismo tiempo solo que debe tener distinto nombre y key_t la memoria.
- En Linux, la memoria compartida se destruye al finalizar la ejecución del **último** proceso padre.

4.2. Windows

- Las tuberías se deben asociar a un proceso en particular para que sean manipuladas.
- El proceso en donde se ocupa la tubería debe ocupar un HANDLE asociado a la entrada estándar (STD_INPUT_HANDLE); el cual es el buffer de entrada de la consola, ahora usado por la tubería.
- Para que haya retroalimentación para comunicar las tuberías, se debe mandar el HANDLE de escritura de la tubería adicional a la información que queremos mandar por la misma tubería, y leerlo del otro lado de forma sincronizada.
- Al finalizar un proceso en Windows, todas las memorias compartidas que creo se destruyen **inmediatamente** de forma intercalada (a diferencia de Linux), por lo que, si requerimos que otro proceso use alguna memoria compartida, el proceso que la creo debe mantenerse en ejecución.

4.3. Generales

- La forma de navegar dentro de una tubería es mediante un stream (usamos funciones tales como read o write, y seek para movernos); mientras que en la memoria compartida es a través de un apuntador (usando la aritmética usual de apuntadores).
- En ambos sistemas operativos se cuenta con las llamadas al sistema necesarias para la manipulación de tubería y memorias, nada más que con distinta forma de invocación. La lógica de las aplicaciones no cambio.
- Es factible, tanto en tuberías como en memoria compartida, mandar todos los datos que se necesiten mediante estructuras: arreglos, structs, etc.

5. Conclusiones

En ambos sistemas operativos contamos con distintas llamadas al sistema para la creación y manipulación de tuberías y memoria compartida, ambos sistemas crean, escriben y obtienen los datos de ambos canales de comunicación o de transferencia de datos entre procesos. Las llamadas al sistema de Linux y Windows varían en sus parámetros y forma de invocación. Pudimos observar cómo varía el comportamiento entre las tuberías y la memoria compartida, incluso entre los sistemas operativos varía el comportamiento. Si se busca ocupar datos entre dos procesos se recomienda usar tuberías, si en caso de que se busca ocupar los mismos datos entre varios se recomienda utilizar memoria compartida.