



INSTITUTO POLITÉCNICO NACIONAL  
ESCUELA SUPERIOR DE CÓMPUTO

Práctica 4 - Administrador de procesos en Linux  
y Windows (1)

Unidad de aprendizaje: Sistemas Operativos

Grupo: 2CM8

*Alumnos(a):*

Briones Tapia Mariana  
Méndez Mejía Sergio Ernesto  
Nicolás Sayago Abigail  
Ramos Diaz Enrique

*Profesor(a):*

Cortes Galicia Jorge

22 de Octubre 2018

# Índice

<b>1</b>	<b>Competencias</b>	<b>2</b>
<b>2</b>	<b>Desarrollo</b>	<b>2</b>
2.1	Puntos a observar y reportar	2
2.1.1	Sección Linux:	2
2.1.2	Sección Windows:	4
2.2	Códigos fuente de los programas desarrollados	6
2.2.1	Sección Linux	6
2.2.2	Sección Windows	42
2.3	Pantallas de ejecución de los programas desarrollados	68
2.3.1	Sección Linux:	68
2.3.2	Sección Windows	83
<b>3</b>	<b>Observaciones</b>	<b>90</b>
<b>4</b>	<b>Análisis Crítico</b>	<b>91</b>
<b>5</b>	<b>Conclusiones</b>	<b>92</b>

# 1. Competencias

El alumno aprende a familiarizarse con el administrados de procesos del sistema operativo Linux y Windows a través de la creación de nuevos procesos por copia exacta de código y/o por sustitución de código para el desarrollo de aplicaciones concurrentes sencillas.

- ✓ Revisión de la creación de procesos en Linux y Windows.
- ✓ Revisión de las llamadas al sistema para la creación de procesos en Linux y Windows.
- ✓ Desarrollo de aplicaciones concurrentes mediante la creación de procesos en ambos sistemas operativos.

## 2. Desarrollo

### 2.1. Puntos a observar y reportar

#### 2.1.1. Sección Linux:

- ✓ Investigación de los siguientes comandos
  - **ps:** Muestra el identificador del proceso que ejecuta la terminal, además del tiempo de ejecución.
  - **ps -fea:** Muestra información de todos los procesos del sistema incluyendo ID, identificador del proceso actual, anterior, tiempo de ejecución, ruta o nombre del proceso, y salida estándar.

```
enrike@enrike:~$ ps
  PID TTY          TIME CMD
 2026 pts/0    00:00:00 bash
 2039 pts/0    00:00:00 ps
enrike@enrike:~$ ps -fea
UID          PID    PPID  C STIME TTY          TIME CMD
root         1      0  4 15:07 ?        00:00:03 /sbin/init splash
root         2      0  0 15:07 ?        00:00:00 [kthreadd]
root         3      2  0 15:07 ?        00:00:00 [kworker/0:0]
root         4      2  0 15:07 ?        00:00:00 [kworker/0:0H]
root         5      2  0 15:07 ?        00:00:00 [kworker/u2:0]
root         6      2  0 15:07 ?        00:00:00 [mm_percpu_wq]
root         7      2  0 15:07 ?        00:00:00 [ksoftirqd/0]
root         8      2  0 15:07 ?        00:00:00 [rcu_sched]
root         9      2  0 15:07 ?        00:00:00 [rcu_bh]
root        10      2  0 15:07 ?        00:00:00 [migration/0]
root        11      2  0 15:07 ?        00:00:00 [watchdog/0]
root        12      2  0 15:07 ?        00:00:00 [cpuhp/0]
root        13      2  0 15:07 ?        00:00:00 [kdevtmpfs]
root        14      2  0 15:07 ?        00:00:00 [netns]
root        15      2  0 15:07 ?        00:00:00 [rcu_tasks_kthre]
root        16      2  0 15:07 ?        00:00:00 [kauditd]
root        17      2  0 15:07 ?        00:00:00 [khungtaskd]
root        18      2  0 15:07 ?        00:00:00 [oom_reaper]
```

✓ Investigación de las siguientes llamadas al sistema

- **ps:** “process status”, permite visualizar el estado de un proceso. Al comando ps se le pueden agregar modificadores como los siguientes:
  - A: Muestra todos los procesos (de todos los usuarios en el sistema).
  - a: Muestra todos los procesos de una consola determinada.
  - d: Muestra todo excepto los líderes de la sesión.
  - e: Muestra todos los procesos (equivalente a -A).
  - T: Muestra todos los procesos de la terminal actual.
  - a: Muestra todos los procesos de la terminal actual incluyendo los de otros usuarios.
  - r: Muestra solamente los procesos corriendo.
  - x: Muestra los procesos en un estilo BSD (sin controlar la terminal).
- **fork():** Crea un proceso hijo duplicando la llamada del proceso, este nuevo proceso está referenciado como hijo mientras que el proceso que lo llamó está referenciado como proceso padre.
- **execv():** Crea o reemplaza la imagen de proceso con una nueva imagen de procesos, esta función provee de un arreglo de punteros a cadenas con terminación null la cual representa la lista de argumentos disponibles para la ejecución de un archivo.
- **getpid():** Obtiene el identificador del proceso que lo ejecuta.
- **getppid():** Obtiene el identificador del padre del proceso que lo ejecuta.
- **wait():** Espera o hace esperar al proceso hasta que su estado y/o funciones terminen.
- **Similares a execv():** exec(), execvpe(), execvp(), execl, execlp, execl.

✓ **Punto 3:** Creación de procesos por copia exacta de código

• Primer código

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4
5  int main(void)
6  {
7      int id_proc;
8      id_proc = fork();
9
10     if(id_proc == 0)
11     {
12         printf("Soy el proceso hijo\n");
13         exit(0);
14     }
15     else
16     {
17         printf("Soy el proceso padre\n");
18         exit(0);
19     }
20 }
```

• Segundo código

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4
5  int main(void)
6  {
7      int id_proc;
8      id_proc = fork();
9
10     if(id_proc == 0)
11     {
12         printf("Soy el proceso hijo\n");
13     }
14     else
15     {
16         printf("Soy el proceso padre\n");
17     }
18     printf("Mensaje en ambos\n");
19     exit(0);
20 }
```

### ✓ Punto 6: Creación de procesos por sustitución de código

#### • Código de sustitución

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5  #include <stdlib.h>
6
7  int main()
8  {
9      pid_t pid;
10     char *argv[3];
11     argv[0] =
12     "/home/enrike/Escritorio/P4-SO/6/hola";
13     // Ruta
14     argv[1] = "Desde el Hijo";
15     argv[2] = NULL;
16     if((pid = fork()) == 0)
17     {
18         printf("Soy el hijo ejecutando:
19             ↪ %s\n", argv[0]);
20         execv(argv[0], argv);
21     }
22     else
23     {
24         wait(0);
25         printf("Soy el Padre\n");
26         exit(0);
27     }

```

#### • Código a ejecutar

```

1  /* hola.c Programa que será invocado */
2  #include <string.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(int argc, char const *argv[])
7  {
8      char mensaje[100];
9      strcpy(mensaje, "Hola Mundo");
10     strcat(mensaje, argv[1]);
11     printf("%s\n", mensaje);
12     exit(0);
13 }

```

### 2.1.2. Sección Windows:

### ✓ Punto 3: Creación de un nuevo proceso

```

1  #include <windows.h>
2  #include <stdio.h>
3  int main(int argc, char *argv[])
4  {
5      STARTUPINFO si;           //Estructura de informacion inicial para Windows
6      PROCESS_INFORMATION pi;   //Estructura de informacion para el adm. de procesos
7      int i;
8      ZeroMemory(&si, sizeof(si));
9      si.cb=sizeof(si);
10     ZeroMemory(&pi, sizeof(pi));
11     if(argc!=2)
12     {
13         printf("Usar: %s Nombre_programa_hijo \n", argv[0]);
14         return 0;
15     }
16     //Creacion proceso hijo
17     if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
18     {
19         printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
20         return 0;
21     }
22     //Proceso Padre
23     printf("Soy el proceso padre\n");
24     WaitForSingleObject(pi.hProcess, INFINITE);
25
26     //Terminacion controlada del proceso e hilo asociado de ejecucion
27     CloseHandle(pi.hProcess);
28     CloseHandle(pi.hThread);
29 }

```

✓ **Punto 4:** Programa que contendrá al proceso hijo

```
1  #include <windows.h>
2  #include <stdio.h>
3  int main(void)
4  {
5      printf("Soy el hijo \n");
6      exit(0);
7  }
```

✓ **Punto 6:** Diferencias y similitudes de creación de procesos en Linux y Windows.

Al igual que en Windows, en el sistema operativo Unix existe la creación de procesos por sustitución de código, ambos sistemas operativos al utilizar la creación de procesos por sustitución son muy parecidos, al recibir los argumentos en el main y la forma en la que trabaja.

Por otra parte las diferencias encontradas son en la parte de la creación de los procesos, que por obvias razones no puede ser igual debido a las diferencias entre estos dos sistemas operativos. Para el sistema operativo Unix se utilizó la llamada al sistema fork() a diferencia de Windows donde se utilizó la llamada al sistema CreateProcess().

✓ **Punto 7:** Función **GetCurrentProcessId()**

GetCurrentProcessId devuelve el identificador de proceso del proceso de llamada.

SINTAXIS

DWORD GetCurrentProcessId (VOID);

VALOR DEVUELTO

El identificador de proceso del proceso de llamada.

OBSERVACIONES

Hasta que el proceso finaliza, el identificador del proceso identifica de manera única el proceso en todo el sistema.

RTSSrun devuelve el valor del ID de proceso actual.

## 2.2. Códigos fuente de los programas desarrollados

### 2.2.1. Sección Linux

✓ **Punto 4:** Árbol de procesos del pizarrón por copia exacta de código.

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main(int argc, char const *argv[])
6  {
7      int ayuda1;
8      int ayuda2;
9      int ayuda3;
10     int ayuda4;
11     int ultimoNum = 10; //Me ayuda para crear la ultima parte de los arboles los ultimos que
        ↳ seran creados por el hijo
12
13     for(int i = 0; i < 10; i++)
14     { //Creo los primeros diez procesos ---- En horizontal
15         ayuda1 = fork();
16         if(ayuda1)
17         { //Si es el padre no hago nada, solo imprimo que soy la main
18             printf("M. Soy el main y mi hijo es: %i\n", ayuda1);
19         }
20         else
21         { //Si soy el hijo, soy del primer nivel asi que debo crear mas hijos
22             printf("1. Soy el hijo numero %i del proceso ->%i\n", i, getpid());
23             for(int x = 1; x <= (10-i); x++)
24             { //Debo crear 10 procesos primero y luego 9 y asi, dependiendo el valor de i,
                ↳ donde i es mi contador del for anterior
25                 ayuda2 = fork(); //Creo el proceso
26                 if(ayuda2)
27                 {
28                     printf("2. Soy padre del proceso %i, y soy hijo del proceso %i\n", ayuda2,
                ↳ getpid());
29                     //Si es el padre, ya no hago nada no me interesa hacer nada mas aqui
30                     exit(0);
31                 }
32                 else
33                 { //Si soy el hijo hago mas hijos para hacer el diagrama hacia abajo
34                     printf("3. Soy el hijo %i del proceso %i\n", x-1, getpid());
35                     ayuda3 = fork();//Hago mas hijos
36                     if(ayuda3)
37                     {
38                         printf("4. Soy padre del proceso %i y soy hijo del proceso %i\n", ayuda3,
                ↳ getpid());
39                         exit(0);
40                     }
41                     else
42                     {
43                         printf("Yo el hijo %i, y mi padre es el proceso: %i\n", x-1, getpid());
44                         if(x == ultimoNum)
45                         { //Comparo con 10 porque el primer tiene que ser de 10 elementos
46                             for(int y = 1; y < x; y++)
47                             {
48                                 ayuda4 = fork();
49                                 if (ayuda4)
50                                 {
51                                     printf("4. Soy padre del proceso %i y soy hijo del proceso
                ↳ %i\n", ayuda4, getpid());
52                                     //Si soy padre solo continuo para crear mas
53                                     exit(0);
54                                 }
55                                 else
56                                 {
57                                     printf("5. Soy el hijo %i del proceso %i\n", y, getpid());

```

```

58         exit(0);
59         //Si es hijo solo continuo, no me interesa mas
60     }
61 }
62 ultimoNum--;
63 //Decremento este numero porque para la siguiente rama debo crear un
64   ↳ hijo menos
65 exit(0);
66 }
67 }
68 }
69 }
70 }
71 return 0;
72 }

```

- ✓ **Punto 5:** Operaciones con matrices de 10x10 con creación de seis procesos por copia exacta de código y de forma secuencial. Medición de ambos tiempos.

#### • Aplicación secuencial

```

1  //  Compilación:
2  //  gcc tiempo.c -c
3  //  gcc operacionesMatrices.c tiempo.o -o o
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8  #include <stdbool.h>
9  #include <math.h>
10 #include <sys/wait.h>
11 #include <sys/types.h>
12 #include <sys/stat.h>
13 #include <fcntl.h>
14 #include <errno.h>
15 #include <unistd.h>
16 #include <string.h>
17 #include "tiempo.h"
18
19 // Declaracion de funciones
20 char* leerDirectorio();
21 int potencia(int base, int pot);
22 void imprimir(double **m, int n);
23 void llenar(double **m, int n);
24 void sumar(double **m1, double **m2, double **resultado, int n);
25 void restar(double **m1, double **m2, double **resultado, int n);
26 void multiplicar(double **m1, double **m2, double **resultado, int n);
27 void transpuesta(double **m, double **resultado, int n);
28 int inversa(double **matriz, double **resultado, int n);
29 double determinante(double **matriz, int n);
30 void crearArchivo(double **matriz, int n, char *nombre, char *directorio);
31 void imprimirArchivo(char *directorio, char *nombre);
32
33 int main(int argc, char const *argv[])
34 {
35     // CREAMOS DIRECTORIO
36     char* path = leerDirectorio(); //Obtenemos el directorio desde la entrada de teclado
37
38     //Variables para medición de tiempos
39     double utime0, stime0, wtime0, utime1, stime1, wtime1;
40     usrtime(&utime0, &stime0, &wtime0);
41
42     int i, n;
43     double **matriz1, **matriz2, **suma, **resta, **mul, **tran1, **tran2, **inv1, **inv2;
44     time_t t;
45     srand((unsigned) time(&t));
46     n = 10;

```



```

47
48 // Inicializa las matrices.
49 matriz1 = (double**)calloc(n, sizeof(double*));
50 for (i = 0; i < n; i++)
51     matriz1[i] = (double*)calloc(n, sizeof(double));
52
53 matriz2 = (double**)calloc(n, sizeof(double*));
54 for (i = 0; i < n; i++)
55     matriz2[i] = (double*)calloc(n, sizeof(double));
56
57 suma = (double**)calloc(n, sizeof(double*));
58 for (i = 0; i < n; i++)
59     suma[i] = (double*)calloc(n, sizeof(double));
60
61 resta = (double**)calloc(n, sizeof(double*));
62 for (i = 0; i < n; i++)
63     resta[i] = (double*)calloc(n, sizeof(double));
64
65 mul = (double**)calloc(n, sizeof(double*));
66 for (i = 0; i < n; i++)
67     mul[i] = (double*)calloc(n, sizeof(double));
68
69 tran1 = (double**)calloc(n, sizeof(double*));
70 for (i = 0; i < n; i++)
71     tran1[i] = (double*)calloc(n, sizeof(double));
72
73 tran2 = (double**)calloc(n, sizeof(double*));
74 for (i = 0; i < n; i++)
75     tran2[i] = (double*)calloc(n, sizeof(double));
76
77 inv1 = (double**)calloc(n, sizeof(double*));
78 for (i = 0; i < n; i++)
79     inv1[i] = (double*)calloc(n, sizeof(double));
80
81 inv2 = (double**)calloc(n, sizeof(double*));
82 for (i = 0; i < n; i++)
83     inv2[i] = (double*)calloc(n, sizeof(double));
84
85 //Llamda al sistema mkdir recibe la ruta del directorio a crear, y los permisos de
86 ↪ escritura, lectura y ejecucion para cada tipo de usuario
87 //Retorna -1 si ocurrieron errores
88 if(mkdir(path, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
89 {
90     perror(path);
91     exit(EXIT_FAILURE);
92 }
93 else
94 {
95     // Llena matriz 1 y matriz 2
96     llenar(matriz1, n);
97     llenar(matriz2, n);
98
99     printf("MATRIZ 1\n"); imprimir(matriz1, n);
100    printf("MATRIZ 2\n"); imprimir(matriz2, n);
101
102    printf("SUMA\n"); sumar(matriz1, matriz2, suma, n);
103    crearArchivo(suma, n, "/suma.txt", path);
104
105    printf("RESTA\n"); restar(matriz1, matriz2, resta, n);
106    crearArchivo(resta, n, "/resta.txt", path);
107
108    printf("MULTIPLICAR\n"); multiplicar(matriz1, matriz2, mul, n);
109    crearArchivo(mul, n, "/mul.txt", path);
110
111    printf("TRANSPUESTA MATRIZ 1\n"); transpuesta(matriz1, tran1, n);
112    crearArchivo(tran1, n, "/tran1.txt", path);
113
114    printf("TRANSPUESTA MATRIZ 2\n"); transpuesta(matriz2, tran2, n);

```

```

115     crearArchivo(tran2, n, "/tran2.txt", path);
116
117     printf("INVERSA MATRIZ 1\n");
118     //Revisamos si la matriz tiene inversa
119     if(inversa(matriz1, inv1, n) != 0)
120         crearArchivo(inv1, n, "/inversa_1.txt", path);
121
122     printf("INVERSA MATRIZ 2\n");
123     if(inversa(matriz2, inv2, n) != 0)
124         crearArchivo(inv2, n, "/inversa_2.txt", path);
125
126     printf(" ----- \n");
127     printf(" -----  RESULTADOS  ----- \n");
128     printf(" ----- \n");
129
130     printf("SUMA\n"); imprimirArchivo(path, "/suma.txt");
131     printf("\nRESTA\n"); imprimirArchivo(path, "/resta.txt");
132     printf("\nMULTIPLICAR\n"); imprimirArchivo(path, "/mul.txt");
133     printf("\nTRANSPUESTA MATRIZ 1\n"); imprimirArchivo(path, "/tran1.txt");
134     printf("\nTRANSPUESTA MATRIZ 2\n"); imprimirArchivo(path, "/tran2.txt");
135     printf("\nINVERSA MATRIZ 1\n"); imprimirArchivo(path, "/inversa_1.txt");
136     printf("\nINVERSA MATRIZ 2\n"); imprimirArchivo(path, "/inversa_2.txt");
137 }
138
139 uswtime(&utime1, &stime1, &wtime1);
140
141 //Cálculo del tiempo de ejecución del programa
142 printf("\n\nTiempo ejecucion: %.4f s\n", wtime1 - wtime0);
143
144 return 0;
145 }
146
147 void crearArchivo(double **matriz, int n, char *nombre, char *directorio)
148 {
149     int i, j;
150     char* dir = (char *)calloc(2000, sizeof(char));
151     char* aux = (char *)calloc(2000, sizeof(char));
152     char num[15];
153     strcpy(aux, directorio);
154     strcat(directorio, nombre);
155     strcpy(dir, directorio);
156
157     // Llamada al sistema creat, recibe la ruta del archivo a crear y los permisos
158     // Retorna -1 si hay errores
159     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
160     {
161         perror(directorio);
162         exit(EXIT_FAILURE);
163     }
164     else
165     {
166         int a = open(directorio, O_WRONLY | O_APPEND);
167         if(a == -1)
168         {
169             perror(directorio);
170             exit(EXIT_FAILURE);
171         }
172         else
173         {
174             for (i = 0; i < n; i++)
175             {
176                 for(j = 0; j < n; j++)
177                 {
178                     sprintf(num, "%.3f\t", matriz[i][j]);
179                     if(!write(a, num, strlen(num)) == strlen (num))
180                     {
181                         perror(directorio);
182                         exit(EXIT_FAILURE);
183                     }

```

```

184         }
185         if(!write(a, "\n", strlen("\n")) == strlen ("\n"))
186         {
187             perror(directorio);
188             exit(EXIT_FAILURE);
189         }
190     }
191 }
192     close(a);
193 }
194     strcpy(directorio, aux);
195     free(aux); free(dir);
196 }
197
198 void imprimirArchivo(char *directorio, char *nombre)
199 {
200     char* dir = (char *)calloc(2000, sizeof(char));
201     char* aux = (char *)calloc(2000, sizeof(char));
202     strcpy(aux, directorio);
203     strcpy(dir, directorio);
204     strcat(dir, nombre);
205
206     int archivo = open(dir, O_RDONLY);
207     if(archivo == -1)
208     {
209         perror(dir);
210         exit(EXIT_FAILURE);
211     }
212     struct stat sb;
213     if(stat(dir, &sb) == -1)
214     {
215         perror(dir);
216         exit(EXIT_FAILURE);
217     }
218     long long longitud = (long long) sb.st_size;
219     char *contenido = (char *)calloc(longitud, sizeof(char));
220
221     if(read(archivo, contenido, longitud) == longitud)
222     {
223         printf("%s", contenido);
224     }
225     if(close(archivo) == -1)
226     {
227         perror(dir);
228         exit(EXIT_FAILURE);
229     }
230     free(contenido);
231     strcpy(directorio, aux);
232 }
233
234 char* leerDirectorio()
235 {
236     char* directorio = (char*)calloc(2000, sizeof(char));
237     printf("Ingresa el nuevo directorio: ");
238     scanf("%s", directorio);
239     return directorio;
240 }
241
242 void imprimir(double **m, int n)
243 {
244     int i, j;
245     for(i = 0; i < n; i++)
246     {
247         for(j = 0; j < n; j++)
248             printf("%.3f\t", m[i][j]);
249         printf("\n");
250     }
251     printf("\n");
252 }

```

```

253
254 // Llena con numeros random
255 void llenar(double **m, int n)
256 {
257     int i, j;
258     for (i = 0; i < n; i++)
259     {
260         for (j = 0; j < n; j++)
261         {
262             m[i][j] = (rand()%11);
263         }
264     }
265 }
266
267 void sumar(double **m1, double **m2, double **resultado, int n)
268 {
269     int i, j;
270     for(i = 0; i < n; i++)
271     {
272         for(j = 0; j < n; j++)
273             resultado[i][j] = m1[i][j] + m2[i][j];
274     }
275 }
276
277 void restar(double **m1, double **m2, double **resultado, int n)
278 {
279     int i, j;
280     for(i = 0; i < n; i++)
281     {
282         for(j = 0; j < n; j++)
283             resultado[i][j] = m1[i][j] - m2[i][j];
284     }
285 }
286
287 void multiplicar(double **m1, double **m2, double **resultado, int n)
288 {
289     int i, j, k, aux;
290     for(i = 0; i < n; i++)
291     {
292         for(j = 0; j < n; j++)
293         {
294             aux = 0;
295             for(k = 0; k < n; k++)
296                 aux = m1[i][k] * m2[k][j] + aux;
297             resultado[i][j] = aux;
298         }
299     }
300 }
301
302 void transpuesta(double **m, double **resultado, int n)
303 {
304     int i, j;
305     for(i = 0; i < n; i++)
306     {
307         for(j = 0; j < n; j++)
308             resultado[i][j] = m[j][i];
309     }
310 }
311
312 bool esCero(double x)
313 {
314     return fabs(x) < 1e-8;
315 }
316
317 double determinante(double **m, int n)
318 {
319     double det = 0, aux = 0;
320     int c;
321     //Si el orden es de 2, multiplica cruzadon directamente

```

```

322     if(n==2)
323         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
324     else
325     {
326         for(int j=0; j<n; j++)
327         {
328             //Crea arreglo dinamico temporal
329             double **menor = (double **)malloc(sizeof(double)*(n-1));
330             //Redimensiona
331             for(int i=0; i<(n-1); i++)
332                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
333             for(int k=1; k<n; k++)
334             {
335                 c = 0;
336                 for(int l=0; l<n; l++)
337                 {
338                     if(l!=j)
339                     {
340                         /*Parte matriz principal en matrices de 3
341                         y multiplica cruzado*/
342                         menor[k-1][c] = m[k][l];
343                         c++;
344                     }
345                 }
346             }
347             //Recurividad, repite la funcion
348             aux = potencia(-1, 2+j)*m[0][j]*determinante(menor, n-1);
349             det += aux;
350
351             for(int x = 0; x<(n-1); x++)
352                 free(menor[x]); //Libera espacio en memoria
353             free(menor);
354         }
355         return det; //Devuelve resultado
356     }
357 }
358
359 // Usando definicion de la adjunta
360 int inversa(double **A, double **resultado, int n)
361 {
362     int tieneInversa;
363     if(determinante(A, n) == 0)
364     {
365         tieneInversa=0;
366         printf("La matriz no tiene inversa. Determinante = 0\n\n");
367     }
368     else{
369         tieneInversa=1;
370         int i, j, k, l;
371         double *tmp;
372         tmp = (double*)malloc(sizeof(double)*n);
373
374         for(i = 0; i < n; ++i)
375             resultado[i][i] = 1;
376         i = 0; j = 0;
377         while(i < n && j < n)
378         {
379             if(esCero(A[i][j]))
380             {
381                 for(k = i + 1; k < n; ++k)
382                 {
383                     if(!esCero(A[k][j]))
384                     {
385                         tmp = A[i];
386                         A[i] = A[k];
387                         A[k] = tmp;
388                         tmp = resultado[i];
389                         resultado[i] = resultado[k];
390                         resultado[k] = tmp;

```

```

391         break;
392     }
393 }
394 }
395 if(!esCero(A[i][j]))
396 {
397     for(l = 0; l < n; ++l)
398         resultado[i][l] /= A[i][j];
399     for(l = n - 1; l >= j; --l)
400         A[i][l] /= A[i][j];
401     for(k = 0; k < n; ++k)
402     {
403         if(i == k) continue;
404         for(l = 0; l < n; ++l)
405             resultado[k][l] -= resultado[i][l] * A[k][j];
406         for(l = n; l >= j; --l)
407             A[k][l] -= A[i][l] * A[k][j];
408     }
409     ++i;
410 }
411 ++j;
412 }
413 }
414 return tieneInversa;
415 }
416
417 int potencia(int base, int pot)
418 {
419     int i, resultado = 1;
420     for(i = 0; i < pot; i++)
421         resultado = base * resultado;
422
423     return resultado;
424 }

```

- Aplicación con seis procesos por copia exacta de código

```

1  // Compilación:
2  // gcc tiempo.c -c
3  // gcc 5.c tiempo.o -o 5
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <time.h>
8  #include <stdbool.h>
9  #include <math.h>
10 #include <sys/wait.h>
11 #include <sys/types.h>
12 #include <sys/stat.h>
13 #include <fcntl.h>
14 #include <errno.h>
15 #include <unistd.h>
16 #include <string.h>
17 #include "tiempo.h"
18
19 // Declaracion de funciones
20 char* leerDirectorio();
21 void imprimir(double **m, int n);
22 void llenar(double **m, int n);
23 void sumar(double **m1, double **m2, double **resultado, int n);
24 void restar(double **m1, double **m2, double **resultado, int n);
25 void multiplicar(double **m1, double **m2, double **resultado, int n);
26 void transpuesta(double **m, double **resultado, int n);
27 int inversa(double **matriz, double **resultado, int n);
28 double determinante(double **matriz, int n);
29 void crearArchivo(double **matriz, int n, char *nombre, char *directorio);
30 int potencia(int base, int pot);
31 void imprimirArchivo(char *directorio, char *nombre);
32
33 int main(int argc, char const *argv[])

```

```

34 {
35     // CREAM DIRECTORIO
36     // Obtenemos el directorio desde la entrada de teclado
37     char* path = leerDirectorio();
38
39     //Variables para medición de tiempos
40     double utime0, stime0, wtime0, utime1, stime1, wtime1;
41     uswtime(&utime0, &stime0, &wtime0);
42
43     int i, n;
44     double **matriz1, **matriz2, **suma, **resta, **mul, **tran1, **tran2, **inv1, **inv2;
45     time_t t;
46     srand((unsigned) time(&t));
47     n = 10; // Tam de la matriz cuadrada
48
49     // Inicializa las matrices.
50     matriz1 = (double**)calloc(n, sizeof(double*));
51     for (i = 0; i < n; i++)
52         matriz1[i] = (double*)calloc(n, sizeof(double));
53
54     matriz2 = (double**)calloc(n, sizeof(double*));
55     for (i = 0; i < n; i++)
56         matriz2[i] = (double*)calloc(n, sizeof(double));
57
58     suma = (double**)calloc(n, sizeof(double*));
59     for (i = 0; i < n; i++)
60         suma[i] = (double*)calloc(n, sizeof(double));
61
62     resta = (double**)calloc(n, sizeof(double*));
63     for (i = 0; i < n; i++)
64         resta[i] = (double*)calloc(n, sizeof(double));
65
66     mul = (double**)calloc(n, sizeof(double*));
67     for (i = 0; i < n; i++)
68         mul[i] = (double*)calloc(n, sizeof(double));
69
70     tran1 = (double**)calloc(n, sizeof(double*));
71     for (i = 0; i < n; i++)
72         tran1[i] = (double*)calloc(n, sizeof(double));
73
74     tran2 = (double**)calloc(n, sizeof(double*));
75     for (i = 0; i < n; i++)
76         tran2[i] = (double*)calloc(n, sizeof(double));
77
78     inv1 = (double**)calloc(n, sizeof(double*));
79     for (i = 0; i < n; i++)
80         inv1[i] = (double*)calloc(n, sizeof(double));
81
82     inv2 = (double**)calloc(n, sizeof(double*));
83     for (i = 0; i < n; i++)
84         inv2[i] = (double*)calloc(n, sizeof(double));
85
86
87     // Llamda al sistema mkdir recibe la ruta del directorio a crear,
88     // y los permisos de escritura, lectura y ejecucion para cada tipo de usuario
89     // Retorna -1 si ocurrieron errores
90     if(mkdir(path, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) ==-1)
91     {
92         perror(path);
93         exit(EXIT_FAILURE);
94     }
95     else
96     {
97         // Llena matriz 1 y matriz 2
98         llenar(matriz1, n); llenar(matriz2, n);
99
100         printf("MATRIZ 1\n"); imprimir(matriz1, n);
101         printf("MATRIZ 2\n"); imprimir(matriz2, n);
102

```

```

103     // CREAMOS LOS PROCESOS
104     int id_proc;
105     for(i = 0; i<5; i++)
106     {
107         id_proc = fork();
108         if(id_proc == 0)
109         {
110             if(i == 0)
111             {
112                 // PROCESO SUMA
113                 printf("----- SUMA\n");
114                 sumar(matriz1, matriz2, suma, n);
115                 crearArchivo(suma, n, "/suma.txt", path);
116             }
117             if(i == 1)
118             {
119                 // PROCESO RESTA
120                 printf("----- RESTA\n");
121                 restar(matriz1, matriz2, resta, n);
122                 crearArchivo(resta, n, "/resta.txt", path);
123             }
124             if(i == 2)
125             {
126                 // PROCESO MULTIPLICACION
127                 printf("----- MULTIPLICACION\n");
128                 multiplicar(matriz1, matriz2, mul, n);
129                 crearArchivo(mul, n, "/mul.txt", path);
130             }
131             if(i == 3)
132             {
133                 // PROCESO TRANSPUESTA
134                 printf("----- TRANSPUESTA\n");
135                 transpuesta(matriz1, tran1, n);
136                 crearArchivo(tran1, n, "/tran1.txt", path);
137
138                 transpuesta(matriz2, tran2, n);
139                 crearArchivo(tran2, n, "/tran2.txt", path);
140             }
141             if(i == 4)
142             {
143                 // PROCESO INVERSA
144                 printf("----- INVERSA\n");
145
146                 //Revisamos si la matriz tiene inversa
147                 if(inversa(matriz1, inv1, n) != 0)
148                     crearArchivo(inv1, n, "/inversa_1.txt", path);
149
150                 if(inversa(matriz2, inv2, n) != 0)
151                     crearArchivo(inv2, n, "/inversa_2.txt", path);
152             }
153             exit(0); // Para que no cree hijos de hijos
154         }
155     }
156     // Para que el padre haga el proceso de leer los archivos
157     for(i = 0; i < 5; i++)
158         wait(0); // Espera 5 procesos
159
160     printf(" ----- \n");
161     printf(" ----- RESULTADOS ----- \n");
162     printf(" ----- \n");
163
164     printf("SUMA\n"); imprimirArchivo(path, "/suma.txt");
165     printf("\nRESTA\n"); imprimirArchivo(path, "/resta.txt");
166     printf("\nMULTIPLICAR\n"); imprimirArchivo(path, "/mul.txt");
167     printf("\nTRANSPUESTA MATRIZ 1\n"); imprimirArchivo(path, "/tran1.txt");
168     printf("\nTRANSPUESTA MATRIZ 2\n"); imprimirArchivo(path, "/tran2.txt");
169     printf("\nINVERSA MATRIZ 1\n"); imprimirArchivo(path, "/inversa_1.txt");
170     printf("\nINVERSA MATRIZ 2\n"); imprimirArchivo(path, "/inversa_2.txt");
171 }

```



```

172     uswtime(&utime1, &stime1, &wtime1);
173
174     //Cálculo del tiempo de ejecución del programa
175     printf("\n\nTiempo ejecucion: %.4f s\n", wtime1 - wtime0);
176
177     return 0;
178 }
179 void crearArchivo(double **matriz, int n, char *nombre, char *directorio)
180 {
181     int i, j;
182     char* dir = (char *)calloc(2000, sizeof(char));
183     char* aux = (char *)calloc(2000, sizeof(char));
184     char num[15];
185     strcpy(aux, directorio);
186     strcat(directorio, nombre);
187     strcpy(dir, directorio);
188
189     // LLamada al sistema cret, recibe la ruta del archivo a crear y los permisos
190     // Retorna -1 si hay errores
191     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
192     {
193         perror(directorio);
194         exit(EXIT_FAILURE);
195     }
196     else
197     {
198         int a = open(directorio, O_WRONLY | O_APPEND);
199         if(a == -1)
200         {
201             perror(directorio);
202             exit(EXIT_FAILURE);
203         }
204         else
205         {
206             for (i = 0; i < n; i++)
207             {
208                 for(j = 0; j < n; j++)
209                 {
210                     sprintf(num, "%.3f\t", matriz[i][j]);
211                     if(!write(a, num, strlen(num)) == strlen (num))
212                     {
213                         perror(directorio);
214                         exit(EXIT_FAILURE);
215                     }
216                 }
217                 if(!write(a, "\n", strlen("\n")) == strlen (" \n"))
218                 {
219                     perror(directorio);
220                     exit(EXIT_FAILURE);
221                 }
222             }
223             close(a);
224         }
225     }
226     strcpy(directorio, aux);
227     free(aux); free(dir);
228 }
229
230 void imprimirArchivo(char *directorio, char *nombre)
231 {
232     char* dir = (char *)calloc(2000, sizeof(char));
233     char* aux = (char *)calloc(2000, sizeof(char));
234     strcpy(aux, directorio);
235     strcpy(dir, directorio);
236     strcat(dir, nombre);
237
238     int archivo = open(dir, O_RDONLY);
239     if(archivo == -1)
240     {

```

```

241     perror(dir);
242     exit(EXIT_FAILURE);
243 }
244 struct stat sb;
245 if(stat(dir, &sb) == -1)
246 {
247     perror(dir);
248     exit(EXIT_FAILURE);
249 }
250 long long longitud = (long long) sb.st_size;
251 char *contenido = (char *)calloc(longitud, sizeof(char));
252
253 if(read(archivo, contenido, longitud) == longitud)
254 {
255     printf("%s", contenido);
256 }
257 if(close(archivo) == -1)
258 {
259     perror(dir);
260     exit(EXIT_FAILURE);
261 }
262 free(contenido);
263 strcpy(directorio, aux);
264 }
265
266 char* leerDirectorio()
267 {
268     char* directorio = (char*)calloc(2000, sizeof(char));
269     printf("Ingresa el nuevo directorio: ");
270     scanf("%s", directorio);
271     return directorio;
272 }
273
274 void imprimir(double **m, int n)
275 {
276     int i, j;
277     for(i = 0; i < n; i++)
278     {
279         for(j = 0; j < n; j++)
280             printf("%.3f\t", m[i][j]);
281         printf("\n");
282     }
283     printf("\n");
284 }
285
286 // Llena con numeros random
287 void llenar(double **m, int n)
288 {
289     int i, j;
290     for (i = 0; i < n; i++)
291     {
292         for (j = 0; j < n; j++)
293         {
294             m[i][j] = (rand()%11);
295         }
296     }
297 }
298
299 void sumar(double **m1, double **m2, double **resultado, int n)
300 {
301     int i, j;
302     for(i = 0; i < n; i++)
303     {
304         for(j = 0; j < n; j++)
305             resultado[i][j] = m1[i][j] + m2[i][j];
306     }
307 }
308
309 void restar(double **m1, double **m2, double **resultado, int n)

```

```

310 {
311     int i, j;
312     for(i = 0; i < n; i++)
313     {
314         for(j = 0; j < n; j++)
315             resultado[i][j] = m1[i][j] - m2[i][j];
316     }
317 }
318
319 void multiplicar(double **m1, double **m2, double **resultado, int n)
320 {
321     int i, j, k, aux;
322     for(i = 0; i < n; i++)
323     {
324         for(j = 0; j < n; j++)
325         {
326             aux = 0;
327             for(k = 0; k < n; k++)
328                 aux = m1[i][k] * m2[k][j] + aux;
329             resultado[i][j] = aux;
330         }
331     }
332 }
333
334 void transpuesta(double **m, double **resultado, int n)
335 {
336     int i, j;
337     for(i = 0; i < n; i++)
338     {
339         for(j = 0; j < n; j++)
340             resultado[i][j] = m[j][i];
341     }
342 }
343
344 bool esCero(double x)
345 {
346     return fabs(x) < 1e-8;
347 }
348
349 double determinante(double **m, int n)
350 {
351     double det = 0, aux = 0;
352     int c;
353     // Si el orden es de 2, multiplica cruzadon directamente
354     if(n==2)
355         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
356     else
357     {
358         for(int j=0; j<n; j++)
359         {
360             // Crea arreglo dinamico temporal
361             double **menor = (double **)malloc(sizeof(double)*(n-1));
362             // Redimensiona
363             for(int i=0; i<(n-1); i++)
364                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
365             for(int k=1; k<n; k++)
366             {
367                 c = 0;
368                 for(int l=0; l<n; l++)
369                 {
370                     if(l!=j)
371                     {
372                         /*Parte matriz principal en matrices de 3
373                         y multiplica cruzado*/
374                         menor[k-1][c] = m[k][l];
375                         c++;
376                     }
377                 }
378             }

```

```

379         // Recursividad, repite la funcion
380         aux = potencia(-1, 2+j)*m[0][j]*determinante(menor, n-1);
381         det += aux;
382
383         for(int x = 0; x<(n-1); x++)
384             free(menor[x]); // Libera espacio en memoria
385         free(menor);
386     }
387     return det; // Devuelve resultado
388 }
389 }
390
391 // Usando definicion de la adjunta
392 int inversa(double **A, double **resultado, int n)
393 {
394     int tieneInversa;
395     if(determinante(A, n) == 0)
396     {
397         tieneInversa=0;
398         printf("La matriz no tiene inversa. Determinante = 0\n\n");
399     }
400     else
401     {
402         tieneInversa=1;
403         int i, j, k, l;
404         double *tmp;
405         tmp = (double*)malloc(sizeof(double)*n);
406
407         for(i = 0; i < n; ++i)
408             resultado[i][i] = 1;
409         i = 0; j = 0;
410         while(i < n && j < n)
411         {
412             if(esCero(A[i][j]))
413             {
414                 for(k = i + 1; k < n; ++k)
415                 {
416                     if(!esCero(A[k][j]))
417                     {
418                         tmp = A[i];
419                         A[i] = A[k];
420                         A[k] = tmp;
421                         tmp = resultado[i];
422                         resultado[i] = resultado[k];
423                         resultado[k] = tmp;
424                         break;
425                     }
426                 }
427             }
428             if(!esCero(A[i][j]))
429             {
430                 for(l = 0; l < n; ++l)
431                     resultado[i][l] /= A[i][j];
432                 for(l = n - 1; l >= j; --l)
433                     A[i][l] /= A[i][j];
434                 for(k = 0; k < n; ++k)
435                 {
436                     if(i == k) continue;
437                     for(l = 0; l < n; ++l)
438                         resultado[k][l] -= resultado[i][l] * A[k][j];
439                     for(l = n; l >= j; --l)
440                         A[k][l] -= A[i][l] * A[k][j];
441                 }
442                 ++i;
443             }
444             ++j;
445         }
446     }
447     return tieneInversa;

```

```

448 }
449
450 int potencia(int base, int pot)
451 {
452     int i, resultado = 1;
453     for(i = 0; i < pot; i++)
454         resultado = base * resultado;
455
456     return resultado;
457 }

```

- Código para el tiempo

```
1 void uswtime(double *usertime, double *systime, double *walltime);
```

- Código objeto para el tiempo

```

1  #include <sys/resource.h>
2  #include <sys/time.h>
3  #include "tiempo.h"
4
5  void uswtime(double *usertime, double *systime, double *walltime)
6  {
7      double mega = 1.0e-6;
8      struct rusage buffer;
9      struct timeval tp;
10     struct timezone tzp;
11     getrusage(RUSAGE_SELF, &buffer);
12     gettimeofday(&tp, &tzp);
13     *usertime = (double) buffer.ru_utime.tv_sec + 1.0e-6 * buffer.ru_utime.tv_usec;
14     *systime = (double) buffer.ru_stime.tv_sec + 1.0e-6 * buffer.ru_stime.tv_usec;
15     *walltime = (double) tp.tv_sec + 1.0e-6 * tp.tv_usec;
16 }
17
18 /*Modo de Empleo:
19 La función uswtime se puede emplear para medir los tiempos de ejecución de determinados
20 segmentos de código en nuestros programas.
21 De forma esquemática, el empleo de esta función constaría de los siguientes pasos:
22 1.- Invocar a uswtime para fijar el instante a partir del cual se va a medir el
   ↳ tiempo.
23     uswtime(&utime0, &stime0, &wtime0);
24 2.- Ejecutar el código cuyo tiempo de ejecución se desea medir.
25 3.- Invocar a uswtime para establecer el instante en el cual finaliza la medición
   del tiempo de ejecución.
26     uswtime(&utime1, &stime1, &wtime1);
27 4.- Calcular los tiempos de ejecución como la diferencia entre la primera y segunda
   invocación a uswtime:
28
29     real: wtime1 - wtime0*/
30
31

```

- ✓ **Punto 7:** Creación de tres procesos por sustitución de código que ejecutarán: Expresión aritmética, cambio de permisos a archivos e inversas de dos matrices.

- Aplicación con tres procesos por sustitución de código

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <fcntl.h>
5  #include <sys/stat.h>
6  #include <sys/wait.h>
7  #include <math.h>
8
9  int main() {
10     pid_t pid;
11     char *argv[4];
12     argv[0] = "/home/enrike/Escritorio/P4-SO/7/expresion";
13     argv[1] = "/home/enrike/Escritorio/P4-SO/7/permisos";
14     argv[2] = "/home/enrike/Escritorio/P4-SO/7/inversa";

```

```

15     argv[3] = NULL;
16
17     if(pid = fork() == 0) //Proceso Hijo-Padre
18     {
19         if(pid = fork() == 0) //Proceso Hijo 1: Expresion
20         {
21             printf("Soy el hijo 1 ejecutando: %s\n", argv[0]);
22             int e = execv(argv[0], argv);
23             if(e == -1) {
24                 perror(argv[0]);
25                 exit(EXIT_FAILURE);
26             }
27         }
28         //Evitar concurrencias entre procesos
29         wait(0);
30         wait(0);
31
32         if(pid = fork() == 0) //Proceso Hijo 2: Permisos
33         {
34             printf("-----\n\n");
35             printf("Soy el hijo 2 ejecutando: %s\n", argv[1]);
36             int e = execv(argv[1], argv);
37             if(e == -1) {
38                 perror(argv[1]);
39                 exit(EXIT_FAILURE);
40             }
41         }
42
43         wait(0);
44         wait(0);
45
46         if(pid = fork() == 0) //Proceso Hijo 3: Inversa Matriz
47         {
48             printf("-----\n\n");
49             printf("Soy el hijo 3 ejecutando: %s\n", argv[2]);
50             int e = execv(argv[2], argv);
51             if(e == -1) {
52                 perror(argv[2]);
53                 exit(EXIT_FAILURE);
54             }
55         }
56         else //Primero proceso hijo, padre de los hijos de arriba
57         {
58             wait(0);
59             printf("Soy el padre de los hijitos\n");
60             exit(0);
61         }
62     }
63 }
64 else //Proceso padre
65 {
66     wait(0);
67     printf("Soy el primer padre del hijo\n");
68     exit(0);
69 }
70
71 }

```

### • Expresión aritmética

```

1 //Compilar: gcc expresion.c -lm -o expresion
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5
6 char arr[1000][5];
7 int actx, acty;
8 char acta;
9 float aux, aux1, res, resaux=0;
10

```

```

11 void operacion() {
12     aux=acta-48;
13     if(arr[actx-3][acty]=='(') {
14         if(resaux!=0)
15             aux1=resaux;
16         else
17             aux1=arr[actx-3][1]-48;
18     }
19     else{
20         aux1=arr[actx-3][acty]-48;
21     }
22     if(arr[actx-2][0]=='+')
23         res=aux+aux1;
24     else if(arr[actx-2][0]=='-')
25         res=aux1-aux;
26     else if(arr[actx-2][0]=='*')
27         res=aux*aux1;
28     else if(arr[actx-2][0]=='/')
29         res=aux1/aux;
30     else if(arr[actx-2][0]=='^')
31         res=pow(aux1,aux);
32     if(arr[actx-3][acty]=='(') {
33         if(actx-3==0) {
34             arr[0][0]=res+48;
35         }
36         else if(arr[actx-4][acty]=='(') {
37             arr[actx-4][acty+1]=res+48;
38             resaux=res;
39             actx-=3;
40             acta='(';
41         }
42         else{
43             arr[actx-4][acty]=res+48;
44             actx-=3;
45             acta='1';
46         }
47     }
48     if(arr[actx-4][acty]=='(') {
49         if(actx-4==0) {
50             arr[0][0]=res+48;
51             actx-=3;
52             acta='1';
53             return;
54         }
55         else if(arr[actx-5][acty]=='(') {
56             arr[actx-5][acty+1]=res+48;
57             resaux=res;
58             actx=actx-4;
59             acta='(';
60         }
61     }
62 }
63 void estado(char c) {
64     if(acta=='(') {
65         if(c=='(') {
66             arr[actx][acty]=c;
67             actx++;
68             acta=c;
69         }
70         else if (c>47 && c<58) {
71             arr[actx][acty]=c;
72             actx++;
73             acta=c;
74         }
75         else if (c=='^' || c=='+' || c=='-' || c=='*' || c=='/') {
76             arr[actx][acty]=c;
77             actx++;
78             acta=c;
79         }

```

```

80     }
81     else if (acta>47 && acta<58){
82         if (c==' '){
83             operacion();
84         }
85         else if (c=='^' || c=='+' || c=='-' || c=='*' || c=='/'){
86             arr[actx][acty]=c;
87             actx++;
88             acta=c;
89         }
90     }
91     else if (acta=='^' || acta=='+' || acta=='-' || acta=='*' || acta=='/'){
92         if (c>47 && c<58){
93             arr[actx][acty]=c;
94             actx++;
95             acta=c;
96         }
97         else if (c==' '){
98             arr[actx][acty]=c;
99             actx++;
100            acta=c;
101        }
102    }
103    else if (acta==' '){
104        if (c=='^' || c=='+' || c=='-' || c=='*' || c=='/'){
105            arr[actx][acty]=c;
106            actx++;
107            acta=c;
108        }
109    }
110    else{
111        printf("Caracter no aceptado\n");
112    }
113 }
114 int main(){
115     char c;
116     int n=0;
117     char *linea = (char*)malloc(1*sizeof(char));
118     printf("Ingrese la expresion: ");
119     do
120     {
121         c = getchar(); //Guardar la cadena que se ingresa
122         linea[n] = c;
123         n++;
124         linea = (char*)realloc(linea, (n+1)*sizeof(char));
125     }while(c != '\n');
126
127     linea[n-1] = '\0'; //Carácter que finaliza una cadena
128     n--;
129     actx=0;
130     acty=0;
131     arr[actx][acty]=linea[0];
132     actx=1;
133     acty=0;
134     acta=linea[0];
135     for(int i=0;i<n;i++){
136         estado(linea[i]);
137     }
138     printf("Resultado: %.2f\n", res);
139 }

```



### • Permisos de archivos

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <fcntl.h>
5  #include <sys/stat.h>
6  /*Permisos:
7      Propietario Lectura: 00400
8      Propietario Escritura: 00200
9      Propietario Ejecucion/Busqueda: 00100
10
11      Grupo Lectura: 00040
12      Grupo Escritura: 00020
13      Grupo Ejecucion/Busqueda: 00010
14
15      Otros Lectura: 00004
16      Otros Escritura: 00002
17      Otros Ejecucion/Busqueda: 00001
18 */
19
20 char* leerNombreArchivo()
21 {
22     char* path = (char*)calloc(2000, sizeof(char));
23     printf("Ruta o nombre del archivo: ");
24     scanf("%s", path);
25     return path;
26 }
27
28 int main()
29 {
30     int permiso = 00, r = 0;
31     //Definimos arreglos para cada tipo de usuario y permisos
32     char usuarios [3][12] = {"Propietario", "Grupo", "Otros"};
33     char permisos [3][20] = {"lectura", "escritura", "ejecucion/busqueda"};
34     //Obtenemos el directorio del archivo desde la entrada de teclado
35     char* archivo = leerNombreArchivo();
36     /*Llamada al sistema open recibe la ruta del archivo a abrir y el modo
37     (lectura, escritura, ejecucion). Devuelve un descriptor de archivo.
38     Si devuelve -1 existieron errores*/
39     if(open(archivo, O_RDWR) == -1){
40         perror(archivo);
41         exit(EXIT_FAILURE);
42     }
43     else
44     {
45         printf("\nIngrese 1 para responde SI e ingrese 0 para responder NO\n\n");
46         for(int i=0; i<3; i++){//Con este for recorremos el arreglo de usuarios
47             for(int j=0; j<3; j++){//Con este for recorremos el arreglo de permisos
48                 printf("%s tiene pemiso de: %s? ", usuarios[i], permisos[j]);
49                 scanf("%d", &r);
50                 if(r==1)//Si la respuesta fue afirmativa
51                 {
52                     switch (j)
53                     {
54                         case 0: permiso = permiso + 04;//4 para lectura
55                         break;
56                         case 1: permiso = permiso + 02;//2 para escritura
57                         break;
58                         case 2: permiso = permiso + 01;//1 para ejecucion/busqueda
59                         break;
60                     }
61                 }
62             }
63             printf("\n");
64             /*Vamos multiplicando por 10 cuando los usuarios sean
65             propietarios y grupos*/
66             if(i<2) permiso=permiso*010;
67         }
68         printf("Permisos: %o\n", permiso);

```

```

69         /*Llamada al sistema chmod recibe la ruta del archivo y los permisos
70         en bits de modo de archivo. Retorna -1 si existieron errores*/
71         if(chmod(archivo, (mode_t)permiso) == -1){
72             perror(archivo);
73             exit(EXIT_FAILURE);
74         }
75         else{
76             printf("Permisos de %s cambiados.\n", archivo);
77         }
78     }
79     return 0;
80 }

```

### • Inversas de matrices

```

1  //Compilar Linux: gcc operacionesMatrices.c -lm -o operacionesMatrices
2  //Ejecutar: ./operacionesMatrices
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <stdbool.h>
7  #include <math.h>
8
9  // Declaracion de funciones
10 void imprimir(double **m, int n);
11 void llenar(double **m, int n);
12 int inversa(double **matriz, double **resultado, int n);
13 double determinante(double **matriz, int n);
14
15 int main(int argc, char const *argv[])
16 {
17     int i, n;
18     double **matriz1, **matriz2, **suma, **resta, **mul, **tran1, **tran2, **inv1, **inv2;
19     time_t t;
20     srand((unsigned) time(&t));
21     n = 10;
22
23     // Inicializa las matrices.
24     matriz1 = (double**)calloc(n, sizeof(double*));
25     for (i = 0; i < n; i++)
26         matriz1[i] = (double*)calloc(n, sizeof(double));
27
28     matriz2 = (double**)calloc(n, sizeof(double*));
29     for (i = 0; i < n; i++)
30         matriz2[i] = (double*)calloc(n, sizeof(double));
31
32     inv1 = (double**)calloc(n, sizeof(double*));
33     for (i = 0; i < n; i++)
34         inv1[i] = (double*)calloc(n, sizeof(double));
35
36     inv2 = (double**)calloc(n, sizeof(double*));
37     for (i = 0; i < n; i++)
38         inv2[i] = (double*)calloc(n, sizeof(double));
39
40     // Llena matriz 1 y matriz 2
41     llenar(matriz1, n);
42     llenar(matriz2, n);
43
44     printf("MATRIZ 1\n"); imprimir(matriz1, n);
45     printf("MATRIZ 2\n"); imprimir(matriz2, n);
46
47     printf("INVERSA MATRIZ 1\n");
48     //Revisamos si la matriz tiene inversa
49     if(inversa(matriz1, inv1, n) != 0)
50         imprimir(inv1, n);
51
52     printf("INVERSA MATRIZ 2\n");
53     if(inversa(matriz2, inv2, n) != 0)
54         imprimir(inv2, n);
55     return 0;

```

```

56 }
57
58 void imprimir(double **m, int n)
59 {
60     int i, j;
61     for(i = 0; i < n; i++)
62     {
63         for(j = 0; j < n; j++)
64             printf("%.3f\t", m[i][j]);
65         printf("\n");
66     }
67     printf("\n");
68 }
69
70 // Llena con numeros random
71 void llenar(double **m, int n)
72 {
73     int i, j;
74     for (i = 0; i < n; i++)
75     {
76         for (j = 0; j < n; j++)
77         {
78             m[i][j] = (rand()%11);
79         }
80     }
81 }
82
83 bool esCero(double x)
84 {
85     return fabs(x) < 1e-8;
86 }
87
88 double determinante(double **m, int n)
89 {
90     double det = 0, aux = 0;
91     int c;
92     //Si el orden es de 2, multiplica cruzadon directamente
93     if(n==2)
94         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
95     else
96     {
97         for(int j=0; j<n; j++)
98         {
99             //Crea arreglo dinamico temporal
100             double **menor = (double **)malloc(sizeof(double)*(n-1));
101             //Redimensiona
102             for(int i=0; i<(n-1); i++)
103                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
104             for(int k=1; k<n; k++)
105             {
106                 c = 0;
107                 for(int l=0; l<n; l++)
108                 {
109                     if(l!=j)
110                     {
111                         /*Parte matriz principal en matrices de 3
112                         y multiplica cruzado*/
113                         menor[k-1][c] = m[k][l];
114                         c++;
115                     }
116                 }
117             }
118             //Recurividad, repite la funcion
119             aux = pow(-1, 2+j)*m[0][j]*determinante(menor, n-1);
120             det += aux;
121         }
122         for(int x = 0; x<(n-1); x++)
123             free(menor[x]); //Libera espacio en memoria
124         free(menor);

```

```

125     }
126     return det; //Devuelve resultado
127 }
128 }
129
130 // Usando definicion de la adjunta
131 int inversa(double **A, double **resultado, int n)
132 {
133     int tieneInversa;
134     if(determinante(A, n) == 0)
135     {
136         tieneInversa=0;
137         printf("La matriz no tiene inversa. Determinante = 0\n\n");
138     }
139     else{
140         tieneInversa=1;
141         int i, j, k, l;
142         double *tmp;
143         tmp = (double*)malloc(sizeof(double)*n);
144
145         for(i = 0; i < n; ++i)
146             resultado[i][i] = 1;
147         i = 0; j = 0;
148         while(i < n && j < n)
149         {
150             if(esCero(A[i][j]))
151             {
152                 for(k = i + 1; k < n; ++k)
153                 {
154                     if(!esCero(A[k][j]))
155                     {
156                         tmp = A[i];
157                         A[i] = A[k];
158                         A[k] = tmp;
159                         tmp = resultado[i];
160                         resultado[i] = resultado[k];
161                         resultado[k] = tmp;
162                         break;
163                     }
164                 }
165             }
166             if(!esCero(A[i][j]))
167             {
168                 for(l = 0; l < n; ++l)
169                     resultado[i][l] /= A[i][j];
170                 for(l = n - 1; l >= j; --l)
171                     A[i][l] /= A[i][j];
172                 for(k = 0; k < n; ++k)
173                 {
174                     if(i == k) continue;
175                     for(l = 0; l < n; ++l)
176                         resultado[k][l] -= resultado[i][l] * A[k][j];
177                     for(l = n; l >= j; --l)
178                         A[k][l] -= A[i][l] * A[k][j];
179                 }
180                 ++i;
181             }
182             ++j;
183         }
184     }
185     return tieneInversa;
186 }

```

- ✓ **Punto 8:** Operaciones con matrices de 10x10 con creación de seis procesos por sustitución código.

• **Aplicación con seis procesos por sustitución de código**

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <fcntl.h>
5  #include <sys/stat.h>
6  #include <sys/wait.h>
7
8  int main() {
9      pid_t pid;
10     char *argv[7];
11     argv[0] = "/home/enrike/Escritorio/P4-SO/8/suma";
12     argv[1] = "/home/enrike/Escritorio/P4-SO/8/resta";
13     argv[2] = "/home/enrike/Escritorio/P4-SO/8/multiplicacion";
14     argv[3] = "/home/enrike/Escritorio/P4-SO/8/transpuesta";
15     argv[4] = "/home/enrike/Escritorio/P4-SO/8/inversa";
16     argv[5] = "/home/enrike/Escritorio/P4-SO/8/leerArchivos";
17     argv[6] = NULL;
18
19     if(pid = fork() == 0) //Proceso Hijo 1: Suma
20     {
21         printf("Soy el hijo 1 ejecutando: %s\n", argv[0]);
22         int e = execv(argv[0], argv);
23         if(e == -1)
24         {
25             perror(argv[0]);
26             exit(EXIT_FAILURE);
27         }
28     }
29     //Evitar concurrencias entre procesos
30     wait(0);
31     wait(0);
32
33     if(pid = fork() == 0) //Proceso Hijo 2: Resta
34     {
35         printf("-----\n\n");
36         printf("Soy el hijo 2 ejecutando: %s\n", argv[1]);
37         int e = execv(argv[1], argv);
38         if(e == -1)
39         {
40             perror(argv[1]);
41             exit(EXIT_FAILURE);
42         }
43     }
44
45     wait(0);
46     wait(0);
47
48     if(pid = fork() == 0) //Proceso Hijo 3: Multiplicacion
49     {
50         printf("-----\n\n");
51         printf("Soy el hijo 3 ejecutando: %s\n", argv[2]);
52         int e = execv(argv[2], argv);
53         if(e == -1)
54         {
55             perror(argv[2]);
56             exit(EXIT_FAILURE);
57         }
58     }
59
60     wait(0);
61     wait(0);
62
63     if(pid = fork() == 0) //Proceso Hijo 4: Transpuesta
64     {

```

```

65     printf("-----\n\n");
66     printf("Soy el hijo 4 ejecutando: %s\n", argv[3]);
67     int e = execv(argv[3], argv);
68     if(e == -1)
69     {
70         perror(argv[3]);
71         exit(EXIT_FAILURE);
72     }
73 }
74
75 wait(0);
76 wait(0);
77
78 if(pid = fork() == 0) //Proceso Hijo 5: Inversa
79 {
80     printf("-----\n\n");
81     printf("Soy el hijo 4 ejecutando: %s\n", argv[4]);
82     int e = execv(argv[4], argv);
83     if(e == -1)
84     {
85         perror(argv[4]);
86         exit(EXIT_FAILURE);
87     }
88 }
89 else //Proceso Padre: Leer Archivos
90 {
91     //Espero que terminen 5 procesos
92     wait(0); wait(0); wait(0); wait(0); wait(0); wait(0);
93     printf("-----\n\n");
94     printf("Soy el padre ejecutando: %s\n", argv[5]);
95     int e = execv(argv[5], argv);
96     if(e == -1)
97     {
98         perror(argv[5]);
99         exit(EXIT_FAILURE);
100    }
101 }
102 }

```

### • Suma de matrices

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 void sumar(double **m1, double **m2, double **resultado, int n);
18 void crearArchivo(double **matriz, int n, char *directorio);
19
20 int main(int argc, char const *argv[])
21 {
22     int i, n;
23     double **matriz1, **matriz2, **suma;
24     time_t t;
25     srand((unsigned) time(&t));
26     n = 10;
27
28     // Inicializa las matrices.
29     matriz1 = (double**) calloc(n, sizeof(double*));

```

```

30     for (i = 0; i < n; i++)
31         matriz1[i] = (double*)calloc(n, sizeof(double));
32
33     matriz2 = (double**)calloc(n, sizeof(double*));
34     for (i = 0; i < n; i++)
35         matriz2[i] = (double*)calloc(n, sizeof(double));
36
37     suma = (double**)calloc(n, sizeof(double*));
38     for (i = 0; i < n; i++)
39         suma[i] = (double*)calloc(n, sizeof(double));
40
41     // CREAM DIRECTORIO
42     char* path = (char*)calloc(2000, sizeof(char));
43     path = "/home/enrike/Escritorio/P4-SO/8/Resultados/suma.txt";
44
45     // Llena matriz 1 y matriz 2
46     llenar(matriz1, n);
47     llenar(matriz2, n);
48
49     printf("MATRIZ 1\n"); imprimir(matriz1, n);
50     printf("MATRIZ 2\n"); imprimir(matriz2, n);
51
52
53     sumar(matriz1, matriz2, suma, n);
54     crearArchivo(suma, n, path);
55
56     printf("ARCHIVO SUMA ESCRITO\n");
57     printf("%s", path);
58     printf("\n");
59
60     return 0;
61 }
62
63 void crearArchivo(double **matriz, int n, char *directorio)
64 {
65     int i, j;
66     char num[15];
67
68     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
69     {
70         perror(directorio);
71         exit(EXIT_FAILURE);
72     }
73     else
74     {
75         int a = open(directorio, O_WRONLY | O_APPEND);
76         if(a == -1)
77         {
78             perror(directorio);
79             exit(EXIT_FAILURE);
80         }
81         else
82         {
83             for (i = 0; i < n; i++)
84             {
85                 for(j = 0; j < n; j++)
86                 {
87                     sprintf(num, "%.3f\t", matriz[i][j]);
88                     if(!write(a, num, strlen(num)) == strlen (num))
89                     {
90                         perror(directorio);
91                         exit(EXIT_FAILURE);
92                     }
93                 }
94                 if(!write(a, "\n", strlen("\n")) == strlen (" \n"))
95                 {
96                     perror(directorio);
97                     exit(EXIT_FAILURE);
98                 }

```

```

99         }
100     }
101     close(a);
102 }
103 }
104
105 void imprimir(double **m, int n)
106 {
107     int i, j;
108     for(i = 0; i < n; i++)
109     {
110         for(j = 0; j < n; j++)
111             printf("%.3f\t", m[i][j]);
112         printf("\n");
113     }
114     printf("\n");
115 }
116
117 // Llena con numeros random
118 void llenar(double **m, int n)
119 {
120     int i, j;
121     for (i = 0; i < n; i++)
122     {
123         for (j = 0; j < n; j++)
124         {
125             m[i][j] = (rand()%11);
126         }
127     }
128 }
129
130 void sumar(double **m1, double **m2, double **resultado, int n)
131 {
132     int i, j;
133     for(i = 0; i < n; i++)
134     {
135         for(j = 0; j < n; j++)
136             resultado[i][j] = m1[i][j] + m2[i][j];
137     }
138 }

```

### • Resta de matrices

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 void restar(double **m1, double **m2, double **resultado, int n);
18 void crearArchivo(double **matriz, int n, char *directorio);
19
20 int main(int argc, char const *argv[])
21 {
22     int i, n;
23     double **matriz1, **matriz2, **resta;
24     time_t t;
25     srand((unsigned) time(&t));
26     n = 10;
27

```



```

28     // Inicializa las matrices.
29     matriz1 = (double**)calloc(n, sizeof(double*));
30     for (i = 0; i < n; i++)
31         matriz1[i] = (double*)calloc(n, sizeof(double));
32
33     matriz2 = (double**)calloc(n, sizeof(double*));
34     for (i = 0; i < n; i++)
35         matriz2[i] = (double*)calloc(n, sizeof(double));
36
37     resta = (double**)calloc(n, sizeof(double*));
38     for (i = 0; i < n; i++)
39         resta[i] = (double*)calloc(n, sizeof(double));
40
41     // CREAM DIRECTORIO
42     char* path = (char*)calloc(2000, sizeof(char));
43     path = "/home/enrike/Escritorio/P4-SO/8/Resultados/resta.txt";
44
45     // Llena matriz 1 y matriz 2
46     llenar(matriz1, n);
47     llenar(matriz2, n);
48
49     printf("MATRIZ 1\n"); imprimir(matriz1, n);
50     printf("MATRIZ 2\n"); imprimir(matriz2, n);
51
52
53     restar(matriz1, matriz2, resta, n);
54     crearArchivo(resta, n, path);
55
56     printf("ARCHIVO RESTA ESCRITO\n");
57     printf("%s", path);
58     printf("\n");
59
60     return 0;
61 }
62
63 void crearArchivo(double **matriz, int n, char *directorio)
64 {
65     int i, j;
66     char num[15];
67
68     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
69     {
70         perror(directorio);
71         exit(EXIT_FAILURE);
72     }
73     else
74     {
75         int a = open(directorio, O_WRONLY | O_APPEND);
76         if(a == -1)
77         {
78             perror(directorio);
79             exit(EXIT_FAILURE);
80         }
81         else
82         {
83             for (i = 0; i < n; i++)
84             {
85                 for(j = 0; j < n; j++)
86                 {
87                     sprintf(num, "%.3f\t", matriz[i][j]);
88                     if(!write(a, num, strlen(num)) == strlen (num))
89                     {
90                         perror(directorio);
91                         exit(EXIT_FAILURE);
92                     }
93                 }
94                 if(!write(a, "\n", strlen("\n")) == strlen (" \n"))
95                 {
96                     perror(directorio);

```

```

97         exit(EXIT_FAILURE);
98     }
99 }
100 }
101     close(a);
102 }
103 }
104
105 void imprimir(double **m, int n)
106 {
107     int i, j;
108     for(i = 0; i < n; i++)
109     {
110         for(j = 0; j < n; j++)
111             printf("%.3f\t", m[i][j]);
112         printf("\n");
113     }
114     printf("\n");
115 }
116
117 // Llena con numeros random
118 void llenar(double **m, int n)
119 {
120     int i, j;
121     for (i = 0; i < n; i++)
122     {
123         for (j = 0; j < n; j++)
124         {
125             m[i][j] = (rand()%11);
126         }
127     }
128 }
129
130 void restar(double **m1, double **m2, double **resultado, int n)
131 {
132     int i, j;
133     for(i = 0; i < n; i++)
134     {
135         for(j = 0; j < n; j++)
136             resultado[i][j] = m1[i][j] - m2[i][j];
137     }
138 }

```

### • Multiplicación de matrices

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 void multiplicar(double **m1, double **m2, double **resultado, int n);
18 void crearArchivo(double **matriz, int n, char *directorio);
19
20 int main(int argc, char const *argv[])
21 {
22     int i, n;
23     double **matriz1, **matriz2, **multiplicacion;
24     time_t t;
25     srand((unsigned) time(&t));

```

```

26     n = 10;
27
28     // Inicializa las matrices.
29     matriz1 = (double**)calloc(n, sizeof(double*));
30     for (i = 0; i < n; i++)
31         matriz1[i] = (double*)calloc(n, sizeof(double));
32
33     matriz2 = (double**)calloc(n, sizeof(double*));
34     for (i = 0; i < n; i++)
35         matriz2[i] = (double*)calloc(n, sizeof(double));
36
37     multiplicacion = (double**)calloc(n, sizeof(double*));
38     for (i = 0; i < n; i++)
39         multiplicacion[i] = (double*)calloc(n, sizeof(double));
40
41     // CREAM DIRECTORIO
42     char* path = (char*)calloc(2000, sizeof(char));
43     path = "/home/enrike/Escritorio/P4-SO/8/Resultados/multiplicacion.txt";
44
45     // Llena matriz 1 y matriz 2
46     llenar(matriz1, n);
47     llenar(matriz2, n);
48
49     printf("MATRIZ 1\n"); imprimir(matriz1, n);
50     printf("MATRIZ 2\n"); imprimir(matriz2, n);
51
52
53     multiplicar(matriz1, matriz2, multiplicacion, n);
54     crearArchivo(multiplicacion, n, path);
55
56     printf("ARCHIVO MULTIPLICACION ESCRITO\n");
57     printf("%s", path);
58     printf("\n");
59
60     return 0;
61 }
62
63 void crearArchivo(double **matriz, int n, char *directorio)
64 {
65     int i, j;
66     char num[15];
67
68     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
69     {
70         perror(directorio);
71         exit(EXIT_FAILURE);
72     }
73     else
74     {
75         int a = open(directorio, O_WRONLY | O_APPEND);
76         if(a == -1)
77         {
78             perror(directorio);
79             exit(EXIT_FAILURE);
80         }
81         else
82         {
83             for (i = 0; i < n; i++)
84             {
85                 for(j = 0; j < n; j++)
86                 {
87                     sprintf(num, "%.3f\t", matriz[i][j]);
88                     if(!write(a, num, strlen(num)) == strlen (num))
89                     {
90                         perror(directorio);
91                         exit(EXIT_FAILURE);
92                     }
93                 }
94                 if(!write(a, "\n", strlen("\n")) == strlen (" \n"))

```

```

95         {
96             perror(directorio);
97             exit(EXIT_FAILURE);
98         }
99     }
100 }
101 close(a);
102 }
103 }
104
105 void imprimir(double **m, int n)
106 {
107     int i, j;
108     for(i = 0; i < n; i++)
109     {
110         for(j = 0; j < n; j++)
111             printf("%.3f\t", m[i][j]);
112         printf("\n");
113     }
114     printf("\n");
115 }
116
117 // Llena con numeros random
118 void llenar(double **m, int n)
119 {
120     int i, j;
121     for (i = 0; i < n; i++)
122     {
123         for (j = 0; j < n; j++)
124         {
125             m[i][j] = (rand()%11);
126         }
127     }
128 }
129
130 void multiplicar(double **m1, double **m2, double **resultado, int n)
131 {
132     int i, j, k, aux;
133     for(i = 0; i < n; i++)
134     {
135         for(j = 0; j < n; j++)
136         {
137             aux = 0;
138             for(k = 0; k < n; k++)
139                 aux = m1[i][k] * m2[k][j] + aux;
140             resultado[i][j] = aux;
141         }
142     }
143 }

```

### • Transpuestas de matrices

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 void transpuesta(double **m, double **resultado, int n);
18 void crearArchivo(double **matriz, int n, char *directorio);

```

```

19
20 int main(int argc, char const *argv[])
21 {
22     int i, n;
23     double **matriz1, **matriz2, **transpuesta1, **transpuesta2;
24     time_t t;
25     srand((unsigned) time(&t));
26     n = 10;
27     // Inicializa las matrices.
28     matriz1 = (double**)calloc(n, sizeof(double*));
29     for (i = 0; i < n; i++)
30         matriz1[i] = (double*)calloc(n, sizeof(double));
31
32     matriz2 = (double**)calloc(n, sizeof(double*));
33     for (i = 0; i < n; i++)
34         matriz2[i] = (double*)calloc(n, sizeof(double));
35
36     transpuesta1 = (double**)calloc(n, sizeof(double*));
37     for (i = 0; i < n; i++)
38         transpuesta1[i] = (double*)calloc(n, sizeof(double));
39
40     transpuesta2 = (double**)calloc(n, sizeof(double*));
41     for (i = 0; i < n; i++)
42         transpuesta2[i] = (double*)calloc(n, sizeof(double));
43
44     // CREAR DIRECTORIO
45     char* path1 = (char*)calloc(2000, sizeof(char));
46     char* path2 = (char*)calloc(2000, sizeof(char));
47     path1 = "/home/enrike/Escritorio/P4-SO/8/Resultados/tran1.txt";
48     path2 = "/home/enrike/Escritorio/P4-SO/8/Resultados/tran2.txt";
49
50
51     // Llena matriz 1 y matriz 2
52     llenar(matriz1, n);
53     llenar(matriz2, n);
54
55     printf("MATRIZ 1\n"); imprimir(matriz1, n);
56     printf("MATRIZ 2\n"); imprimir(matriz2, n);
57
58
59     transpuesta(matriz1, transpuesta1, n);
60     transpuesta(matriz2, transpuesta2, n);
61     crearArchivo(transpuesta1, n, path1);
62     crearArchivo(transpuesta2, n, path2);
63
64     printf("ARCHIVOS TRANSPUESTAS ESCRITO\n");
65     printf("%s", path1);
66     printf("\n");
67     printf("%s", path2);
68     printf("\n");
69
70     return 0;
71 }
72
73 void crearArchivo(double **matriz, int n, char *directorio)
74 {
75     int i, j;
76     char num[15];
77
78     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
79     {
80         perror(directorio);
81         exit(EXIT_FAILURE);
82     }
83     else
84     {
85         int a = open(directorio, O_WRONLY | O_APPEND);
86         if(a == -1)
87         {

```

```

88         perror(directorio);
89         exit(EXIT_FAILURE);
90     }
91     else
92     {
93         for (i = 0; i < n; i++)
94         {
95             for(j = 0; j < n; j++)
96             {
97                 sprintf(num, "%.3f\t", matriz[i][j]);
98                 if(!write(a, num, strlen(num)) == strlen (num))
99                 {
100                     perror(directorio);
101                     exit (EXIT_FAILURE);
102                 }
103             }
104             if(!write(a, "\n", strlen("\n")) == strlen ("\n"))
105             {
106                 perror(directorio);
107                 exit (EXIT_FAILURE);
108             }
109         }
110     }
111     close(a);
112 }
113 }
114
115 void imprimir(double **m, int n)
116 {
117     int i, j;
118     for(i = 0; i < n; i++)
119     {
120         for(j = 0; j < n; j++)
121             printf("%.3f\t", m[i][j]);
122         printf("\n");
123     }
124     printf("\n");
125 }
126
127 // Llena con numeros random
128 void llenar(double **m, int n)
129 {
130     int i, j;
131     for (i = 0; i < n; i++)
132     {
133         for (j = 0; j < n; j++)
134         {
135             m[i][j] = (rand()%11);
136         }
137     }
138 }
139
140 void transpuesta(double **m, double **resultado, int n)
141 {
142     int i, j;
143     for(i = 0; i < n; i++)
144     {
145         for(j = 0; j < n; j++)
146             resultado[i][j] = m[j][i];
147     }
148 }

```

#### • Inversas de matrices

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>

```

```

7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 int inversa(double **matriz, double **resultado, int n);
18 int potencia(int base, int pot);
19 double determinante(double **matriz, int n);
20 void crearArchivo(double **matriz, int n, char *directorio);
21
22 int main(int argc, char const *argv[])
23 {
24     int i, n;
25     double **matriz1, **matriz2, **inversal, **inversa2;
26     time_t t;
27     srand((unsigned) time(&t));
28     n = 10;
29     // Inicializa las matrices.
30     matriz1 = (double**)calloc(n, sizeof(double*));
31     for (i = 0; i < n; i++)
32         matriz1[i] = (double*)calloc(n, sizeof(double));
33
34     matriz2 = (double**)calloc(n, sizeof(double*));
35     for (i = 0; i < n; i++)
36         matriz2[i] = (double*)calloc(n, sizeof(double));
37
38     inversal = (double**)calloc(n, sizeof(double*));
39     for (i = 0; i < n; i++)
40         inversal[i] = (double*)calloc(n, sizeof(double));
41
42     inversa2 = (double**)calloc(n, sizeof(double*));
43     for (i = 0; i < n; i++)
44         inversa2[i] = (double*)calloc(n, sizeof(double));
45
46     // CREAM DIRECTORIO
47     char* path1 = (char*)calloc(2000, sizeof(char));
48     char* path2 = (char*)calloc(2000, sizeof(char));
49     path1 = "/home/enrike/Escritorio/P4-SO/8/Resultados/inv_1.txt";
50     path2 = "/home/enrike/Escritorio/P4-SO/8/Resultados/inv_2.txt";
51
52
53     // Llena matriz 1 y matriz 2
54     llenar(matriz1, n);
55     llenar(matriz2, n);
56
57     printf("MATRIZ 1\n"); imprimir(matriz1, n);
58     printf("MATRIZ 2\n"); imprimir(matriz2, n);
59
60
61     inversa(matriz1, inversal, n);
62     inversa(matriz2, inversa2, n);
63     crearArchivo(inversal, n, path1);
64     crearArchivo(inversa2, n, path2);
65
66     printf("ARCHIVOS INVERSAS ESCRITO\n");
67     printf("%s", path1);
68     printf("\n");
69     printf("%s", path2);
70     printf("\n");
71
72     return 0;
73 }
74
75 void crearArchivo(double **matriz, int n, char *directorio)

```

```

76 {
77     int i, j;
78     char num[15];
79
80     if(creat(directorio, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) == -1)
81     {
82         perror(directorio);
83         exit(EXIT_FAILURE);
84     }
85     else
86     {
87         int a = open(directorio, O_WRONLY | O_APPEND);
88         if(a == -1)
89         {
90             perror(directorio);
91             exit(EXIT_FAILURE);
92         }
93         else
94         {
95             for (i = 0; i < n; i++)
96             {
97                 for(j = 0; j < n; j++)
98                 {
99                     sprintf(num, "%.3f\t", matriz[i][j]);
100                     if(!write(a, num, strlen(num)) == strlen (num))
101                     {
102                         perror(directorio);
103                         exit(EXIT_FAILURE);
104                     }
105                 }
106                 if(!write(a, "\n", strlen("\n")) == strlen ("\n"))
107                 {
108                     perror(directorio);
109                     exit(EXIT_FAILURE);
110                 }
111             }
112             close(a);
113         }
114     }
115 }
116
117 void imprimir(double **m, int n)
118 {
119     int i, j;
120     for(i = 0; i < n; i++)
121     {
122         for(j = 0; j < n; j++)
123             printf("%.3f\t", m[i][j]);
124         printf("\n");
125     }
126     printf("\n");
127 }
128
129 // Llena con numeros random
130 void llenar(double **m, int n)
131 {
132     int i, j;
133     for (i = 0; i < n; i++)
134     {
135         for (j = 0; j < n; j++)
136         {
137             m[i][j] = (rand()%11);
138         }
139     }
140 }
141
142 bool esCero(double x)
143 {
144     return fabs(x) < 1e-8;

```



```

145 }
146
147 double determinante(double **m, int n)
148 {
149     double det = 0, aux = 0;
150     int c;
151     //Si el orden es de 2, multiplica cruzadon directamente
152     if(n==2)
153         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
154     else
155     {
156         for(int j=0; j<n; j++)
157         {
158             //Crea arreglo dinamico temporal
159             double **menor = (double **)malloc(sizeof(double)*(n-1));
160             //Redimensiona
161             for(int i=0; i<(n-1); i++)
162                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
163             for(int k=1; k<n; k++)
164             {
165                 c = 0;
166                 for(int l=0; l<n; l++)
167                 {
168                     if(l!=j)
169                     {
170                         /*Parte matriz principal en matrices de 3
171                         y multiplica cruzado*/
172                         menor[k-1][c] = m[k][l];
173                         c++;
174                     }
175                 }
176             }
177             //Recurividad, repite la funcion
178             aux = potencia(-1, 2+j)*m[0][j]*determinante(menor, n-1);
179             det += aux;
180
181             for(int x = 0; x<(n-1); x++)
182                 free(menor[x]); //Libera espacio en memoria
183             free(menor);
184         }
185         return det; //Devuelve resultado
186     }
187 }
188
189 // Usando definicion de la adjunta
190 int inversa(double **A, double **resultado, int n)
191 {
192     int tieneInversa;
193     if(determinante(A, n) == 0)
194     {
195         tieneInversa=0;
196         printf("La matriz no tiene inversa. Determinante = 0\n\n");
197     }
198     else{
199         tieneInversa=1;
200         int i, j, k, l;
201         double *tmp;
202         tmp = (double*)malloc(sizeof(double)*n);
203
204         for(i = 0; i < n; ++i)
205             resultado[i][i] = 1;
206         i = 0; j = 0;
207         while(i < n && j < n)
208         {
209             if(esCero(A[i][j]))
210             {
211                 for(k = i + 1; k < n; ++k)
212                 {
213                     if(!esCero(A[k][j]))

```

```

214         {
215             tmp = A[i];
216             A[i] = A[k];
217             A[k] = tmp;
218             tmp = resultado[i];
219             resultado[i] = resultado[k];
220             resultado[k] = tmp;
221             break;
222         }
223     }
224 }
225 if(!esCero(A[i][j]))
226 {
227     for(l = 0; l < n; ++l)
228         resultado[i][l] /= A[i][j];
229     for(l = n - 1; l >= j; --l)
230         A[i][l] /= A[i][j];
231     for(k = 0; k < n; ++k)
232     {
233         if(i == k) continue;
234         for(l = 0; l < n; ++l)
235             resultado[k][l] -= resultado[i][l] * A[k][j];
236         for(l = n; l >= j; --l)
237             A[k][l] -= A[i][l] * A[k][j];
238     }
239     ++i;
240 }
241 ++j;
242 }
243 }
244 return tieneInversa;
245 }
246
247 int potencia(int base, int pot)
248 {
249     int i, resultado = 1;
250     for(i = 0; i < pot; i++)
251         resultado = base * resultado;
252
253     return resultado;
254 }

```

- Leer archivos con resultados

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdbool.h>
5  #include <math.h>
6  #include <sys/wait.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <errno.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 // Declaracion de funciones
15 void imprimirArchivo(char *directorio, char *nombre);
16
17 int main(int argc, char const *argv[])
18 {
19     char* path = (char*)calloc(2000, sizeof(char));
20     strcpy(path, "/home/enrike/Escritorio/P4-SO/8/Resultados");
21     printf(" -----\n");
22     printf(" ----- RESULTADOS -----\n");
23     printf(" -----\n");
24
25     printf("SUMA\n"); imprimirArchivo(path, "/suma.txt");
26     printf("\nRESTA\n"); imprimirArchivo(path, "/resta.txt");

```

```

27     printf("\nMULTIPLICAR\n"); imprimirArchivo(path, "/multiplicacion.txt");
28     printf("\nTRANSPUESTA MATRIZ 1\n"); imprimirArchivo(path, "/tran1.txt");
29     printf("\nTRANSPUESTA MATRIZ 2\n"); imprimirArchivo(path, "/tran2.txt");
30     printf("\nINVERSA MATRIZ 1\n"); imprimirArchivo(path, "/inv_1.txt");
31     printf("\nINVERSA MATRIZ 2\n"); imprimirArchivo(path, "/inv_2.txt");
32
33     return 0;
34 }
35
36 void imprimirArchivo(char *directorio, char *nombre)
37 {
38     char* dir = (char *)calloc(2000, sizeof(char));
39     char* aux = (char *)calloc(2000, sizeof(char));
40     strcpy(aux, directorio);
41     strcpy(dir, directorio);
42     strcat(dir, nombre);
43
44     int archivo = open(dir, O_RDONLY);
45     if(archivo == -1)
46     {
47         perror(dir);
48         exit(EXIT_FAILURE);
49     }
50     struct stat sb;
51     if(stat(dir, &sb) == -1)
52     {
53         perror(dir);
54         exit(EXIT_FAILURE);
55     }
56     long long longitud = (long long) sb.st_size;
57     char *contenido = (char *)calloc(longitud, sizeof(char));
58
59     if(read(archivo, contenido, longitud) == longitud)
60     {
61         printf("%s", contenido);
62     }
63     if(close(archivo) == -1)
64     {
65         perror(dir);
66         exit(EXIT_FAILURE);
67     }
68     strcpy(directorio, aux);
69 }

```

### 2.2.2. Sección Windows

✓ **Punto 5:** Programa que contendrá al proceso hijo, con un nuevo argumento.

- Proceso padre

```

1  //Compilar: gcc 5.c -o 5
2  //Ejecutar: 5 hijo
3  #include <windows.h>
4  #include <stdio.h>
5  int main(int argc, char *argv[])
6  {
7      STARTUPINFO si;           //Estructura de informacion inicial para Windows
8      PROCESS_INFORMATION pi;    //Estructura de informacion para el adm. de procesos
9      int i;
10     ZeroMemory(&si, sizeof(si));
11     si.cb=sizeof(si);
12     ZeroMemory(&pi, sizeof(pi));
13     if(argc!=2)
14     {
15         printf("Usar: %s Nombre_programa_hijo \n", argv[0]);
16         return 0;

```

```

17     }
18     //Creacion proceso hijo
19     if(!CreateProcess(NULL, argv[1],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
20     {
21         printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
22         return 0;
23     }
24     //Proceso Padre
25     printf("Soy el proceso padre\n");
26     WaitForSingleObject(pi.hProcess, INFINITE);
27
28     //Terminacion controlada del proceso e hilo asociado de ejecucion
29     CloseHandle(pi.hProcess);
30     CloseHandle(pi.hThread);
31 }

```

- Proceso hijo

```

1 //Compilar: gcc 5_hijo.c -o hijo
2 #include <windows.h>
3 #include <stdio.h>
4 int main(void)
5 {
6     printf("Soy el hijo \n");
7     exit(0);
8 }

```

### ✓ Punto 7: Árbol de procesos que imprimen su identificador.

- Main

```

1 //Compilar: gcc 7.c -o 7
2 //Ejecutar: 7 padre
3 #include <windows.h>
4 #include <stdio.h>
5 int main(int argc, char *argv[])
6 {
7     HANDLE hProcess;
8     HANDLE hThread;
9     STARTUPINFO si;
10    PROCESS_INFORMATION pi;
11    DWORD dwProcessId=0;
12    DWORD dwThreadId=0;
13    ZeroMemory(&si, sizeof(si));
14    ZeroMemory(&pi, sizeof(pi));
15
16    if(!CreateProcess(NULL, argv[1],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
17    {
18        printf("Fallo al crear el proceso (%d)\n",GetLastError());
19    }
20    printf("Soy el hijo-padre\n");
21    printf("ID del Proceso (%d)\n",pi.dwProcessId);
22
23    WaitForSingleObject(pi.hProcess,INFINITE);
24    CloseHandle(pi.hThread);
25    CloseHandle(pi.hProcess);
26 }

```

- Un proceso hijo-padre

```

1 //Compilar: gcc 7_padre.c -o padre
2 #include <windows.h>
3 #include <stdio.h>
4 int main(int argc, char *argv[])
5 {
6     HANDLE hProcess;
7     HANDLE hThread;
8     STARTUPINFO si;
9     PROCESS_INFORMATION pi;
10    DWORD dwProcessId=0;

```

```

11     DWORD dwThreadId=0;
12     ZeroMemory(&si, sizeof(si));
13     ZeroMemory(&pi, sizeof(pi));
14     for(int i=0;i<5;i++)
15     {
16         if(!CreateProcess(NULL,
17             "C:\\Users\\YaKerTaker\\Google Drive\\5to
18             ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\7\\hijo.exe",
19             NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
20         {
21             printf("Fallo al crear el proceso (%d)\n",GetLastError());
22         }
23         printf("ID del Proceso (%d)\n",pi.dwProcessId);
24     }
25     WaitForSingleObject(pi.hProcess,INFINITE);
26     CloseHandle(pi.hThread);
27     CloseHandle(pi.hProcess);

```

- Cinco procesos hijos

```

1  //Compilar: gcc 7_hijo.c -o hijo
2  #include <windows.h>
3  #include <stdio.h>
4  int main(int argc, char *argv[])
5  {
6      STARTUPINFO si;
7      PROCESS_INFORMATION pi;
8      int i;
9      ZeroMemory(&si, sizeof(si));
10     si.cb=sizeof(si);
11     ZeroMemory(&pi, sizeof(pi));
12     if(argc!=2)
13     {
14         printf(/*Usar: %s *//"Soy el proceso hijo \n"/*, argv[0]*);
15     }
16     //Creacion proceso hijo
17     printf("Creando nietos \n");
18     for(int i=0;i<3;i++)
19     {
20         if(!CreateProcess(NULL,
21             "C:\\Users\\YaKerTaker\\Google Drive\\5to
22             ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\7\\nieto.exe",
23             NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
24         {
25             printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
26             return 0;
27         }
28         printf("ID del Proceso (%d)\n",pi.dwProcessId);
29     }
30     //Proceso Padre
31     WaitForSingleObject(pi.hProcess, INFINITE);
32     //Terminacion controlada del proceso e hilo asociado de ejecucion
33     CloseHandle(pi.hProcess);
34     CloseHandle(pi.hThread);
35 }

```

- Tres procesos nietos por cada hijo

```

1  //Compilar: gcc 7_nieto.c -o nieto
2  #include <windows.h>
3  #include <stdio.h>
4  int main(int argc, char *argv[])
5  {
6      STARTUPINFO si;
7      PROCESS_INFORMATION pi;
8      int i;
9      ZeroMemory(&si, sizeof(si));
10     si.cb=sizeof(si);
11     ZeroMemory(&pi, sizeof(pi));

```

```

12     if(argc!=2)
13     {
14         printf(/*Usar: %s */"Soy el proceso nieto \n"/*, argv[0]*/);
15         return 0;
16     }
17     //Creacion proceso hijo
18     if(!CreateProcess(NULL, argv[0],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
19     {
20         printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
21         return 0;
22     }
23     //Proceso Padre
24     printf("Soy el proceso padre\n");
25     WaitForSingleObject(pi.hProcess, INFINITE);
26
27     //Terminacion controlada del proceso e hilo asociado de ejecucion
28     CloseHandle(pi.hProcess);
29     CloseHandle(pi.hThread);
30 }

```

- ✓ **Punto 8:** Operaciones con matrices de 10x10 con creación de seis procesos por copia exacta de código y de forma secuencial. Medición de ambos tiempos.

- **Aplicación secuencial**

```

1  //  Compilación:
2  //  gcc tiempo.c -c
3  //  gcc operacionesMatrices.c tiempo.o -o o
4  #include <windows.h>
5  #include <time.h>
6  #include <errno.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <stdbool.h>
11 #include <math.h>
12
13 // Declaracion de funciones
14 int potencia(int base, int pot);
15 void imprimir(double **m, int n);
16 void llenar(double **m, int n);
17 void sumar(double **m1, double **m2, double **resultado, int n);
18 void restar(double **m1, double **m2, double **resultado, int n);
19 void multiplicar(double **m1, double **m2, double **resultado, int n);
20 void transpuesta(double **m, double **resultado, int n);
21 int inversa(double **matriz, double **resultado, int n);
22 double determinante(double **matriz, int n);
23 void crearArchivo(double **res, char* dir, char *nombre);
24 void imprimirArchivo(char *directorio, char *nombre);
25
26 int main(int argc, char const *argv[])
27 {
28     clock_t tiempo_inicio, tiempo_final;
29     double segundos;
30
31     tiempo_inicio = clock();
32     int i, n;
33     double **matriz1, **matriz2, **suma, **resta, **mul, **tran1, **tran2, **inv1, **inv2;
34     time_t t;
35     srand((unsigned) time(&t));
36     n = 10;
37
38     // Inicializa las matrices.
39     matriz1 = (double**)calloc(n,sizeof(double*));
40     for (i = 0; i < n; i++)
41         matriz1[i] = (double*)calloc(n,sizeof(double));
42
43     matriz2 = (double**)calloc(n,sizeof(double*));

```

```

44     for (i = 0; i < n; i++)
45         matriz2[i] = (double*)calloc(n, sizeof(double));
46
47     suma = (double**)calloc(n, sizeof(double*));
48     for (i = 0; i < n; i++)
49         suma[i] = (double*)calloc(n, sizeof(double));
50
51     resta = (double**)calloc(n, sizeof(double*));
52     for (i = 0; i < n; i++)
53         resta[i] = (double*)calloc(n, sizeof(double));
54
55     mul = (double**)calloc(n, sizeof(double*));
56     for (i = 0; i < n; i++)
57         mul[i] = (double*)calloc(n, sizeof(double));
58
59     tran1 = (double**)calloc(n, sizeof(double*));
60     for (i = 0; i < n; i++)
61         tran1[i] = (double*)calloc(n, sizeof(double));
62
63     tran2 = (double**)calloc(n, sizeof(double*));
64     for (i = 0; i < n; i++)
65         tran2[i] = (double*)calloc(n, sizeof(double));
66
67     inv1 = (double**)calloc(n, sizeof(double*));
68     for (i = 0; i < n; i++)
69         inv1[i] = (double*)calloc(n, sizeof(double));
70
71     inv2 = (double**)calloc(n, sizeof(double*));
72     for (i = 0; i < n; i++)
73         inv2[i] = (double*)calloc(n, sizeof(double));
74
75     char* path = (char*)calloc(2000, sizeof(char));
76     strcpy(path, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados0");
77
78     // Llena matriz 1 y matriz 2
79     llenar(matriz1, n);
80     llenar(matriz2, n);
81
82     printf("MATRIZ 1\n"); imprimir(matriz1, n);
83     printf("MATRIZ 2\n"); imprimir(matriz2, n);
84
85     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
86
87     printf("SUMA\n"); sumar(matriz1, matriz2, suma, n);
88     crearArchivo(suma, path, "suma.txt");
89
90     printf("RESTA\n"); restar(matriz1, matriz2, resta, n);
91     crearArchivo(resta, path, "resta.txt");
92
93     printf("MULTIPLICAR\n"); multiplicar(matriz1, matriz2, mul, n);
94     crearArchivo(mul, path, "multiplicacion.txt");
95
96     printf("TRANSPUESTA MATRIZ 1\n"); transpuesta(matriz1, tran1, n);
97     crearArchivo(tran1, path, "transpuesta_1.txt");
98
99     printf("TRANSPUESTA MATRIZ 2\n"); transpuesta(matriz2, tran2, n);
100    crearArchivo(tran2, path, "transpuesta_2.txt");
101
102    printf("INVERSA MATRIZ 1\n");
103    //Revisamos si la matriz tiene inversa
104    if(inversa(matriz1, inv1, n) != 0)
105        crearArchivo(inv1, path, "inversa_1.txt");
106
107    printf("INVERSA MATRIZ 2\n");
108    if(inversa(matriz2, inv2, n) != 0)
109        crearArchivo(inv2, path, "inversa_2.txt");
110
111    printf(" ----- \n");

```

```

112     printf(" ----- RESULTADOS -----\n");
113     printf(" -----\n");
114
115     printf("SUMA\n"); imprimirArchivo(path, "suma.txt");
116     printf("\nRESTA\n"); imprimirArchivo(path, "resta.txt");
117     printf("\nMULTIPLICAR\n"); imprimirArchivo(path, "multiplicacion.txt");
118     printf("\nTRANSPUESTA MATRIZ 1\n"); imprimirArchivo(path, "transpuesta_1.txt");
119     printf("\nTRANSPUESTA MATRIZ 2\n"); imprimirArchivo(path, "transpuesta_2.txt");
120     printf("\nINVERSA MATRIZ 1\n"); imprimirArchivo(path, "inversa_1.txt");
121     printf("\nINVERSA MATRIZ 2\n"); imprimirArchivo(path, "inversa_2.txt");
122
123     tiempo_final = clock();
124     segundos = (double)(tiempo_final - tiempo_inicio) / CLOCKS_PER_SEC;
125     //Cálculo del tiempo de ejecución del programa
126     printf("\n\nTiempo ejecucion: %.4f s\n", segundos);
127
128     return 0;
129 }
130
131 void crearArchivo(double **res, char* dir, char *nombre)
132 {
133     int i,j;
134     char num[15];
135     char *name = (char *)calloc(150,sizeof(char));
136     strcpy(name, nombre);
137     char *ruta = (char *)calloc(150,sizeof(char));
138     // Contatenamos la ruta original con el nombre del archivo
139     strcat(strcat(strcpy(ruta, dir), "\\"), name);
140
141     HANDLE h = CreateFile(ruta, //ruta del archivo
142                           GENERIC_WRITE, //abrir para escribir
143                           0, //no compartir
144                           NULL, // seguridad por default
145                           CREATE_ALWAYS, //crear siempre
146                           FILE_ATTRIBUTE_TEMPORARY, //archivo normal
147                           NULL); //sin tributos
148
149     if (h == INVALID_HANDLE_VALUE)
150     {
151         perror(ruta);
152         exit(EXIT_FAILURE);
153     }
154     else
155     {
156         DWORD bytesEscritos = 0;
157
158         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
159         {
160             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo
161             {
162                 sprintf(num, "%.3f\t", res[i][j]);
163                 /*Funcion WwriteFile recibe los parametros a continuacion y devuelve un true
164                 ↪ si no existieron errores*/
165                 BOOL escribir = WriteFile(
166                     h, // abrir handle del archivo
167                     num, // informacion a escribir
168                     (DWORD)strlen(num), // tamaño de bytes a escribir
169                     &bytesEscritos, // tamaño de bytes escrit
170                     NULL); // no overlapped structure
171
172                 if(!escribir)
173                 {
174                     perror(ruta);
175                     exit(EXIT_FAILURE);
176                 }
177             }
178             BOOL espacio = WriteFile(
179                 h, // abrir handle del archivo
180                 "\n", // informacion a escribir

```



```

180             (DWORD)strlen("\n"),           // tamaño de bytes a escribir
181             &bytesEscritos,                 // tamaño de bytes escrit
182             NULL);                          // no overlapped structure
183         if(!espacio)
184         {
185             perror(ruta);
186             exit(EXIT_FAILURE);
187         }
188     }
189     //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna un
190     //↪ valor cero si han habido errores
191     if(CloseHandle(h) == 0)
192     {
193         perror(ruta);
194         exit(EXIT_FAILURE);
195     }
196 }
197
198 void imprimirArchivo(char *directorio, char *nombre)
199 {
200     char *name = (char *)calloc(150, sizeof(char));
201     strcpy(name, nombre);
202     char *dir = (char *)calloc(150, sizeof(char));
203     // Contatenamos la ruta original con el nombre del archivo
204     strcat(strcat(strcpy(dir, directorio), "\\"), name);
205
206     HANDLE file;
207     DWORD BytesEscritos = 0;
208     char *contenido = (char*)calloc(1000000, sizeof(char));
209     file = CreateFile(
210         dir,
211         GENERIC_WRITE | GENERIC_READ,
212         FILE_SHARE_READ,
213         NULL,
214         OPEN_EXISTING,
215         FILE_ATTRIBUTE_NORMAL,
216         NULL);
217
218     if(file == INVALID_HANDLE_VALUE)
219     {
220         printf("Error 1\n");
221         perror(dir);
222         exit(EXIT_FAILURE);
223     }
224     else
225     {
226         if(ReadFile(file, contenido, 1000000, &BytesEscritos, NULL))
227         {
228             printf("%s", contenido);
229         }
230         free(contenido);
231
232         if(CloseHandle(file) == 0)
233         {
234             perror(dir);
235             exit(EXIT_FAILURE);
236         }
237     }
238     free(name);
239     free(dir);
240 }
241
242 void imprimir(double **m, int n)
243 {
244     int i, j;
245     for(i = 0; i < n; i++)
246     {
247         for(j = 0; j < n; j++)

```

```

248         printf("%.3f\t", m[i][j]);
249     }
250     }
251     printf("\n");
252 }
253
254 // Llena con numeros random
255 void llenar(double **m, int n)
256 {
257     int i, j;
258     for (i = 0; i < n; i++)
259     {
260         for (j = 0; j < n; j++)
261         {
262             m[i][j] = (rand()%11);
263         }
264     }
265 }
266
267 void sumar(double **m1, double **m2, double **resultado, int n)
268 {
269     int i, j;
270     for(i = 0; i < n; i++)
271     {
272         for(j = 0; j < n; j++)
273             resultado[i][j] = m1[i][j] + m2[i][j];
274     }
275 }
276
277 void restar(double **m1, double **m2, double **resultado, int n)
278 {
279     int i, j;
280     for(i = 0; i < n; i++)
281     {
282         for(j = 0; j < n; j++)
283             resultado[i][j] = m1[i][j] - m2[i][j];
284     }
285 }
286
287 void multiplicar(double **m1, double **m2, double **resultado, int n)
288 {
289     int i, j, k, aux;
290     for(i = 0; i < n; i++)
291     {
292         for(j = 0; j < n; j++)
293         {
294             aux = 0;
295             for(k = 0; k < n; k++)
296                 aux = m1[i][k] * m2[k][j] + aux;
297             resultado[i][j] = aux;
298         }
299     }
300 }
301
302 void transpuesta(double **m, double **resultado, int n)
303 {
304     int i, j;
305     for(i = 0; i < n; i++)
306     {
307         for(j = 0; j < n; j++)
308             resultado[i][j] = m[j][i];
309     }
310 }
311 bool esCero(double x)
312 {
313     return fabs(x) < 1e-8;
314 }
315
316 double determinante(double **m, int n)

```

```

317 {
318     double det = 0, aux = 0;
319     int c, i, j, k, l, x;
320     //Si el orden es de 2, multiplica cruzadon directamente
321     if(n==2)
322         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
323     else
324     {
325         for(j=0; j<n; j++)
326         {
327             //Crea arreglo dinamico temporal
328             double **menor = (double **)malloc(sizeof(double)*(n-1));
329             //Redimensiona
330             for(i=0; i<(n-1); i++)
331                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
332             for(k=1; k<n; k++)
333             {
334                 c = 0;
335                 for(l=0; l<n; l++)
336                 {
337                     if(l!=j)
338                     {
339                         /*Parte matriz principal en matrices de 3
340                         y multiplica cruzado*/
341                         menor[k-1][c] = m[k][l];
342                         c++;
343                     }
344                 }
345             }
346             //Recurividad, repite la funcion
347             aux = potencia(-1, 2+j)*m[0][j]*determinante(menor, n-1);
348             det += aux;
349
350             for(x = 0; x<(n-1); x++)
351                 free(menor[x]); //Libera espacio en memoria
352             free(menor);
353         }
354         return det; //Devuelve resultado
355     }
356 }
357
358 // Usando definicion de la adjunta
359 int inversa(double **A, double **resultado, int n)
360 {
361     int tieneInversa;
362     if(determinante(A, n) == 0)
363     {
364         tieneInversa=0;
365         printf("La matriz no tiene inversa. Determinante = 0\n\n");
366     }
367     else{
368         tieneInversa=1;
369         int i, j, k, l;
370         double *tmp;
371         tmp = (double*)malloc(sizeof(double)*n);
372
373         for(i = 0; i < n; ++i)
374             resultado[i][i] = 1;
375         i = 0; j = 0;
376         while(i < n && j < n)
377         {
378             if(esCero(A[i][j]))
379             {
380                 for(k = i + 1; k < n; ++k)
381                 {
382                     if(!esCero(A[k][j]))
383                     {
384                         tmp = A[i];
385                         A[i] = A[k];

```

```

386         A[k] = tmp;
387         tmp = resultado[i];
388         resultado[i] = resultado[k];
389         resultado[k] = tmp;
390         break;
391     }
392 }
393 }
394 if(!esCero(A[i][j]))
395 {
396     for(l = 0; l < n; ++l)
397         resultado[i][l] /= A[i][j];
398     for(l = n - 1; l >= j; --l)
399         A[i][l] /= A[i][j];
400     for(k = 0; k < n; ++k)
401     {
402         if(i == k) continue;
403         for(l = 0; l < n; ++l)
404             resultado[k][l] -= resultado[i][l] * A[k][j];
405         for(l = n; l >= j; --l)
406             A[k][l] -= A[i][l] * A[k][j];
407     }
408     ++i;
409 }
410 ++j;
411 }
412 }
413 return tieneInversa;
414 }
415
416 int potencia(int base, int pot)
417 {
418     int i, resultado = 1;
419     for(i = 0; i < pot; i++)
420         resultado = base * resultado;
421
422     return resultado;
423 }

```

## • Aplicación con seis procesos

### o Main

```

1  #include <windows.h>
2  #include <stdio.h>
3  #include <time.h>
4  int main(int argc, char *argv[]){
5      clock_t tiempo_inicio, tiempo_final;
6      double segundos;
7      tiempo_inicio = clock();
8
9      HANDLE hProcess;
10     HANDLE hThread;
11     STARTUPINFO si;
12     PROCESS_INFORMATION pi;
13     DWORD dwProcessId=0;
14     DWORD dwThreadId=0;
15     ZeroMemory(&si, sizeof(si));
16     ZeroMemory(&pi, sizeof(pi));
17
18     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↳ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\suma.exe",
19         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
20         printf("Fallo al crear el proceso (%d)\n",GetLastError());
21     }
22     printf("\nSuma de Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
23
24     WaitForSingleObject(pi.hProcess,INFINITE);
25     CloseHandle(pi.hThread);
26     CloseHandle(pi.hProcess);

```

```

27
28     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\resta.exe",
29         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
30         printf("Fallo al crear el proceso (%d)\n",GetLastError());
31     }
32     printf("\nResta de Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
33
34     WaitForSingleObject(pi.hProcess,INFINITE);
35     CloseHandle(pi.hThread);
36     CloseHandle(pi.hProcess);
37
38     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\multiplicacion.exe",
39         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
40         printf("Fallo al crear el proceso (%d)\n",GetLastError());
41     }
42     printf("\nMultiplicacion de Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
43
44     WaitForSingleObject(pi.hProcess,INFINITE);
45     CloseHandle(pi.hThread);
46     CloseHandle(pi.hProcess);
47
48     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\transpuesta.exe",
49         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
50         printf("Fallo al crear el proceso (%d)\n",GetLastError());
51     }
52     printf("\nTranspuestas de Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
53
54     WaitForSingleObject(pi.hProcess,INFINITE);
55     CloseHandle(pi.hThread);
56     CloseHandle(pi.hProcess);
57
58     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\inversa.exe",
59         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
60         printf("Fallo al crear el proceso (%d)\n",GetLastError());
61     }
62     printf("\nInversas de Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
63
64     WaitForSingleObject(pi.hProcess,INFINITE);
65
66     CloseHandle(pi.hThread);
67     CloseHandle(pi.hProcess);
68
69     if(!CreateProcess(NULL, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\leerArchivos.exe",
70         NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
71         printf("Fallo al crear el proceso (%d)\n",GetLastError());
72     }
73     printf("\nImpresion de las Matrices, ID del Proceso (%d)\n",pi.dwProcessId);
74     //cout<<"Create process Succed"<<"\n";
75     //cout<<"process ID"<<pi.dwProcessId<<"\n";
76     WaitForSingleObject(pi.hProcess,INFINITE);
77     CloseHandle(pi.hThread);
78     CloseHandle(pi.hProcess);
79
80     WaitForSingleObject(pi.hProcess,INFINITE);
81     CloseHandle(pi.hThread);
82     CloseHandle(pi.hProcess);
83
84     tiempo_final = clock();
85     segundos = (double)(tiempo_final - tiempo_inicio) / CLOCKS_PER_SEC;
86     //Cálculo del tiempo de ejecución del programa
87     printf("\n\nTiempo ejecucion: %.4f s\n", segundos);
88 }

```

### ○ Suma de matrices

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  double mat1[10][10];
9  double mat2[10][10];
10 double res[10][10];
11
12 void CrearMatriz()
13 {
14     srand(time(NULL));
15     for(int i=0; i<10; i++)
16     {
17         for(int j=0; j<10; j++)
18         {
19             mat1[i][j]=rand()%11;
20             mat2[j][i]=rand()%11;
21         }
22     }
23 }
24
25 void SumarMatriz()
26 {
27     for(int i=0; i<10; i++)
28     {
29         for(int j=0; j<10; j++)
30         {
31             res[i][j]=mat1[i][j]+mat2[i][j];
32         }
33     }
34 }
35
36 void crearArchivo(char* ruta)
37 {
38     int i, j;
39     char num[15];
40
41     HANDLE h = CreateFile(ruta,                //ruta del archivo
42                           GENERIC_WRITE,        //abrir para escribir
43                           0,                    //no compartir
44                           NULL,                // seguridad por default
45                           CREATE_ALWAYS,       //crear siempre
46                           FILE_ATTRIBUTE_TEMPORARY, //archivo normal
47                           NULL);               //sin tributos
48
49     if (h == INVALID_HANDLE_VALUE)
50     {
51         perror(ruta);
52         exit(EXIT_FAILURE);
53     }
54     else
55     {
56         DWORD bytesEscritos = 0;
57
58         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
59         {
60             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo
61             {
62                 sprintf(num, "%.3f\t", res[i][j]);
63                 /*Funcion WriteFile recibe los parametros a continuacion y devuelve un
64                  ↳ true si no existieron errores*/
65                 BOOL escribir = WriteFile(
66                     h,                                // abrir handle del archivo
67                     num,                              // informacion a escribir
68                     (DWORD)strlen(num),               // tamaño de bytes a escribir

```

```

68             &bytesEscritos,           // tamaño de bytes escrit
69             NULL);                     // no overlapped structure
70
71         if(!escribir)
72         {
73             perror(ruta);
74             exit(EXIT_FAILURE);
75         }
76     }
77     BOOL espacio = WriteFile(
78         h,                               // abrir handle del archivo
79         "\n",                             // informacion a escribir
80         (DWORD)strlen("\n"),              // tamaño de bytes a escribir
81         &bytesEscritos,                   // tamaño de bytes escrit
82         NULL);                           // no overlapped structure
83     if(!espacio)
84     {
85         perror(ruta);
86         exit(EXIT_FAILURE);
87     }
88 }
89 //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna un
90 //↪ valor cero si han habido errores
91 if(CloseHandle(h) == 0)
92 {
93     perror(ruta);
94     exit(EXIT_FAILURE);
95 }
96 }
97
98 void imprimir(double m[10][10])
99 {
100     int i, j;
101     for(i = 0; i < 10; i++)
102     {
103         for(j = 0; j < 10; j++)
104             printf("%.3f\t", m[i][j]);
105         printf("\n");
106     }
107     printf("\n");
108 }
109
110 int main(int argc, char *argv[])
111 {
112     CrearMatriz();
113     imprimir(mat1);
114     imprimir(mat2);
115     SumarMatriz();
116     char* dir = (char*)calloc(2000, sizeof(char));
117     dir =
118     "C:\\Users\\YaKerTaker\\Google Drive\\5to
119     ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\suma.txt";
120     crearArchivo(dir);
121     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
122     printf("ARCHIVO SUMA ESCRITO\n");
123     printf("%s", dir);
124     printf("\n");
125
126     STARTUPINFO si;
127     PROCESS_INFORMATION pi;
128     int i;
129     ZeroMemory(&si, sizeof(si));
130     si.cb=sizeof(si);
131     ZeroMemory(&pi, sizeof(pi));
132     if(argc!=2)
133     {
134         //printf(/*Usar: %s */"Soy el proceso hijo \n"/*, argv[0]*/);

```

```

135         return 0;
136     }
137     //Creacion proceso hijo
138
139     if(!CreateProcess(NULL, argv[0],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
140     {
141         printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
142         return 0;
143     }
144     //Proceso Padre
145     WaitForSingleObject(pi.hProcess, INFINITE);
146
147     //Terminacion controlada del proceso e hilo asociado de ejecucion
148     CloseHandle(pi.hProcess);
149     CloseHandle(pi.hThread);
150 }

```

#### o Resta de matrices

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  double mat1[10][10];
9  double mat2[10][10];
10 double res[10][10];
11
12 void CrearMatriz()
13 {
14     srand(time(NULL));
15     for(int i=0;i<10;i++)
16     {
17         for(int j=0;j<10;j++)
18         {
19             mat1[i][j]=rand()%11;
20             mat2[j][i]=rand()%11;
21         }
22     }
23 }
24 void RestarMatriz()
25 {
26     for(int i=0;i<10;i++)
27     {
28         for(int j=0;j<10;j++)
29         {
30             res[i][j]=mat1[i][j]-mat2[i][j];
31         }
32     }
33 }
34
35 void crearArchivo(char* ruta)
36 {
37     int i,j;
38     char num[15];
39
40     HANDLE h = CreateFile(ruta,                                //ruta del archivo
41                           GENERIC_WRITE,                       //abrir para escribir
42                           0,                                    //no compartir
43                           NULL,                                // seguridad por default
44                           CREATE_ALWAYS,                       //crear siempre
45                           FILE_ATTRIBUTE_TEMPORARY,             //archivo normal
46                           NULL);                               //sin tributos
47
48     if (h == INVALID_HANDLE_VALUE)
49     {
50         perror(ruta);
51         exit(EXIT_FAILURE);

```



```

52     }
53     else
54     {
55         DWORD bytesEscritos = 0;
56
57         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
58         {
59             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo
60             {
61                 sprintf(num, "%.3f\t", res[i][j]);
62                 /*Funcion WriteFile recibe los parametros a continuacion y devuelve un
63                  ↳ true si no existieron errores*/
64                 BOOL escribir = WriteFile(
65                     h,                                     // abrir handle del archivo
66                     num,                                   // informacion a escribir
67                     (DWORD)strlen(num),                   // tamaño de bytes a escribir
68                     &bytesEscritos,                       // tamaño de bytes escrit
69                     NULL);                                // no overlapped structure
70
71                 if(!escribir)
72                 {
73                     perror(ruta);
74                     exit(EXIT_FAILURE);
75                 }
76                 BOOL espacio = WriteFile(
77                     h,                                     // abrir handle del archivo
78                     "\n",                                 // informacion a escribir
79                     (DWORD)strlen("\n"),                 // tamaño de bytes a escribir
80                     &bytesEscritos,                       // tamaño de bytes escrit
81                     NULL);                                // no overlapped structure
82                 if(!espacio)
83                 {
84                     perror(ruta);
85                     exit(EXIT_FAILURE);
86                 }
87             }
88             //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna un
89             ↳ valor cero si han habido errores
90             if(CloseHandle(h) == 0)
91             {
92                 perror(ruta);
93                 exit(EXIT_FAILURE);
94             }
95         }
96     }
97     void imprimir(double m[10][10])
98     {
99         int i, j;
100        for(i = 0; i < 10; i++)
101        {
102            for(j = 0; j < 10; j++)
103                printf("%.3f\t", m[i][j]);
104            printf("\n");
105        }
106        printf("\n");
107    }
108
109    int main(int argc, char *argv[])
110    {
111        CrearMatriz();
112        imprimir(mat1);
113        imprimir(mat2);
114        RestarMatriz();
115        char* dir = (char*)calloc(2000, sizeof(char));
116        dir =
117        "C:\\Users\\YaKerTaker\\Google Drive\\5to
118        ↳ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\resta.txt";

```

```

118     crearArchivo(dir);
119     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
120
121     printf("ARCHIVO RESTA ESCRITO\n");
122     printf("%s", dir);
123     printf("\n");
124
125     STARTUPINFO si;
126     PROCESS_INFORMATION pi;
127     int i;
128     ZeroMemory(&si, sizeof(si));
129     si.cb=sizeof(si);
130     ZeroMemory(&pi, sizeof(pi));
131     if(argc!=2)
132     {
133         //printf(/*Usar: %s */"Soy el proceso hijo \n"/*, argv[0]*/);
134         return 0;
135     }
136     //Creacion proceso hijo
137
138     if(!CreateProcess(NULL, argv[0],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
139     {
140         printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
141         return 0;
142     }
143     //Proceso Padre
144     WaitForSingleObject(pi.hProcess, INFINITE);
145
146     //Terminacion controlada del proceso e hilo asociado de ejecucion
147     CloseHandle(pi.hProcess);
148     CloseHandle(pi.hThread);
149 }

```

#### o Multiplicación de matrices

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  double mat1[10][10];
9  double mat2[10][10];
10 double res[10][10];
11
12 void CrearMatriz()
13 {
14     srand(time(NULL));
15     for(int i=0;i<10;i++)
16     {
17         for(int j=0;j<10;j++)
18         {
19             mat1[i][j]=rand()%11;
20             mat2[j][i]=rand()%11;
21         }
22     }
23 }
24
25 void MultiplicarMatriz()
26 {
27     int i, j, k, aux;
28     for(i = 0; i < 10; i++)
29     {
30         for(j = 0; j < 10; j++)
31         {
32             aux = 0;
33             for(k = 0; k < 10; k++)
34                 aux = mat1[i][k] * mat2[k][j] + aux;
35             res[i][j] = aux;

```

```

36     }
37 }
38 }
39
40 void crearArchivo(char* ruta)
41 {
42     int i,j;
43     char num[15];
44
45     HANDLE h = CreateFile(ruta,                //ruta del archivo
46                           GENERIC_WRITE,        //abrir para escribir
47                           0,                    //no compartir
48                           NULL,                // seguridad por default
49                           CREATE_ALWAYS,       //crear siempre
50                           FILE_ATTRIBUTE_TEMPORARY, //archivo normal
51                           NULL);               //sin tributos
52
53     if (h == INVALID_HANDLE_VALUE)
54     {
55         perror(ruta);
56         exit(EXIT_FAILURE);
57     }
58     else
59     {
60         DWORD bytesEscritos = 0;
61
62         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
63         {
64             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo
65             {
66                 sprintf(num, "%.3f\t", res[i][j]);
67                 /*Funcion WriteFile recibe los parametros a continuacion y devuelve un
68                 ↪ true si no existieron errores*/
69                 BOOL escribir = WriteFile(
70                     h,                                // abrir handle del archivo
71                     num,                              // informacion a escribir
72                     (DWORD)strlen(num),               // tamaño de bytes a escribir
73                     &bytesEscritos,                  // tamaño de bytes escrit
74                     NULL);                            // no overlapped structure
75
76                 if(!escribir)
77                 {
78                     perror(ruta);
79                     exit(EXIT_FAILURE);
80                 }
81                 BOOL espacio = WriteFile(
82                     h,                                // abrir handle del archivo
83                     "\n",                            // informacion a escribir
84                     (DWORD)strlen("\n"),              // tamaño de bytes a escribir
85                     &bytesEscritos,                  // tamaño de bytes escrit
86                     NULL);                            // no overlapped structure
87
88                 if(!espacio)
89                 {
90                     perror(ruta);
91                     exit(EXIT_FAILURE);
92                 }
93             }
94             //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna un
95             ↪ valor cero si han habido errores
96             if(CloseHandle(h) == 0)
97             {
98                 perror(ruta);
99                 exit(EXIT_FAILURE);
100             }
101         }
102     }
103
104 void imprimir(double m[10][10])

```

```

103 {
104     int i, j;
105     for(i = 0; i < 10; i++)
106     {
107         for(j = 0; j < 10; j++)
108             printf("%.3f\t", m[i][j]);
109         printf("\n");
110     }
111     printf("\n");
112 }
113
114 int main(int argc, char *argv[])
115 {
116     CrearMatriz();
117     imprimir(mat1);
118     imprimir(mat2);
119     MultiplicarMatriz();
120     char* dir = (char*)calloc(2000, sizeof(char));
121     dir =
122     "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↳ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\multiplicacion.txt";
123     crearArchivo(dir);
124     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
125
126     printf("ARCHIVO MULTIPLICACION ESCRITO\n");
127     printf("%s", dir);
128     printf("\n");
129
130     STARTUPINFO si;
131     PROCESS_INFORMATION pi;
132     int i;
133     ZeroMemory(&si, sizeof(si));
134     si.cb=sizeof(si);
135     ZeroMemory(&pi, sizeof(pi));
136     if(argc!=2)
137     {
138         //printf(/*Usar: %s */"Soy el proceso hijo \n"/*, argv[0]*/);
139         return 0;
140     }
141     //Creacion proceso hijo
142
143     if(!CreateProcess(NULL, argv[0], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
144     {
145         printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
146         return 0;
147     }
148     //Proceso Padre
149     WaitForSingleObject(pi.hProcess, INFINITE);
150
151     //Terminacion controlada del proceso e hilo asociado de ejecucion
152     CloseHandle(pi.hProcess);
153     CloseHandle(pi.hThread);
154 }

```

#### o Transpuestas de matrices

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  double mat1[10][10];
9  double mat2[10][10];
10 double res1[10][10];
11 double res2[10][10];
12
13 void CrearMatriz()
14 {

```

```

15     srand(time(NULL));
16     for(int i=0; i<10; i++)
17     {
18         for(int j=0; j<10; j++)
19         {
20             mat1[i][j]=rand()%11;
21             mat2[j][i]=rand()%11;
22         }
23     }
24 }
25
26 void TranspuestaMatriz()
27 {
28
29     int i, j;
30     for(i = 0; i < 10; i++)
31     {
32         for(j = 0; j < 10; j++)
33         {
34             res1[i][j] = mat1[j][i];
35             res2[i][j] = mat2[j][i];
36         }
37     }
38 }
39
40 void crearArchivo(char* ruta, double m[10][10])
41 {
42     int i,j;
43     char num[15];
44
45     HANDLE h = CreateFile(ruta,                //ruta del archivo
46                           GENERIC_WRITE,        //abrir para escribir
47                           0,                    //no compartir
48                           NULL,                // seguridad por default
49                           CREATE_ALWAYS,        //crear siempre
50                           FILE_ATTRIBUTE_TEMPORARY, //archivo normal
51                           NULL);                //sin tributos
52
53     if (h == INVALID_HANDLE_VALUE)
54     {
55         perror(ruta);
56         exit(EXIT_FAILURE);
57     }
58     else
59     {
60         DWORD bytesEscritos = 0;
61
62         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
63         {
64             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo
65             {
66                 sprintf(num, "%.3f\t", m[i][j]);
67                 /*Funcion WriteFile recibe los parametros a continuacion y devuelve un
68                 ↪ true si no existieron errores*/
69                 BOOL escribir = WriteFile(
69                     h,                // abrir handle del archivo
70                     num,              // informacion a escribir
71                     (DWORD)strlen(num), // tamaño de bytes a escribir
72                     &bytesEscritos,    // tamaño de bytes escrit
73                     NULL);            // no overlapped structure
74
75                 if(!escribir)
76                 {
77                     perror(ruta);
78                     exit(EXIT_FAILURE);
79                 }
80             }
81             BOOL espacio = WriteFile(
82                 h,                // abrir handle del archivo

```

```

83         "\n",                                     // informacion a escribir
84         (DWORD)strlen("\n"),                       // tamaño de bytes a escribir
85         &bytesEscritos,                             // tamaño de bytes escrit
86         NULL);                                     // no overlapped structure
87     if(!espacio)
88     {
89         perror(ruta);
90         exit(EXIT_FAILURE);
91     }
92 }
93 //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna un
94 //↪ valor cero si han habido errores
95 if(CloseHandle(h) == 0)
96 {
97     perror(ruta);
98     exit(EXIT_FAILURE);
99 }
100 }
101
102 void imprimir(double m[10][10])
103 {
104     int i, j;
105     for(i = 0; i < 10; i++)
106     {
107         for(j = 0; j < 10; j++)
108             printf("%.3f\t", m[i][j]);
109         printf("\n");
110     }
111     printf("\n");
112 }
113
114 int main(int argc, char *argv[])
115 {
116     CrearMatriz();
117     imprimir(mat1);
118     imprimir(mat2);
119     TranspuestaMatriz();
120     char* dir1 = (char*)calloc(2000, sizeof(char));
121     dir1 =
122     "C:\\Users\\YaKerTaker\\Google Drive\\5to
123     ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\transpuesta1.txt";
124     crearArchivo(dir1, res1);
125
126     char* dir2 = (char*)calloc(2000, sizeof(char));
127     dir2 =
128     "C:\\Users\\YaKerTaker\\Google Drive\\5to
129     ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\transpuesta2.txt";
130     crearArchivo(dir2, res2);
131     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
132
133     printf("ARCHIVOS TRANSPUESTAS ESCRITO\n");
134     printf("%s", dir1);
135     printf("\n");
136     printf("%s", dir2);
137     printf("\n");
138
139     STARTUPINFO si;
140     PROCESS_INFORMATION pi;
141     int i;
142     ZeroMemory(&si, sizeof(si));
143     si.cb=sizeof(si);
144     ZeroMemory(&pi, sizeof(pi));
145     if(argc!=2)
146     {
147         //printf(/*Usar: %s */"Soy el proceso hijo \n"/*, argv[0]*/);
148         return 0;
149     }
150 }
151 //Creacion proceso hijo

```

```

149
150         if(!CreateProcess(NULL, argv[0],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
151         {
152             printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
153             return 0;
154         }
155         //Proceso Padre
156         WaitForSingleObject(pi.hProcess, INFINITE);
157
158         //Terminacion controlada del proceso e hilo asociado de ejecucion
159         CloseHandle(pi.hProcess);
160         CloseHandle(pi.hThread);
161     }

```

#### o Inversas de matrices

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <stdbool.h>
8  #include <math.h>
9
10 // Declaracion de funciones
11 void imprimir(double **m, int n);
12 void llenar(double **m, int n);
13 int inversa(double **matriz, double **resultado, int n);
14 int potencia(int base, int pot);
15 double determinante(double **matriz, int n);
16 void crearArchivo(char *ruta, double **m);
17
18 int main(int argc, char *argv[])
19 {
20     int i, n;
21     double **matriz1, **matriz2, **inversa1, **inversa2;
22     time_t t;
23     srand((unsigned) time(&t));
24     n = 10;
25     // Inicializa las matrices.
26     matriz1 = (double**)calloc(n,sizeof(double*));
27     for (i = 0; i < n; i++)
28         matriz1[i] = (double*)calloc(n,sizeof(double));
29
30     matriz2 = (double**)calloc(n,sizeof(double*));
31     for (i = 0; i < n; i++)
32         matriz2[i] = (double*)calloc(n,sizeof(double));
33
34     inversa1 = (double**)calloc(n,sizeof(double*));
35     for (i = 0; i < n; i++)
36         inversa1[i] = (double*)calloc(n,sizeof(double));
37
38     inversa2 = (double**)calloc(n,sizeof(double*));
39     for (i = 0; i < n; i++)
40         inversa2[i] = (double*)calloc(n,sizeof(double));
41
42     // CREAR DIRECTORIO
43     char* dir1 = (char*)calloc(2000,sizeof(char));
44     dir1 =
45     "C:\\Users\\YaKerTaker\\Google Drive\\5to
46     ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\inversa1.txt";
47
48     char* dir2 = (char*)calloc(2000,sizeof(char));
49     dir2 =
50     "C:\\Users\\YaKerTaker\\Google Drive\\5to
51     ↪ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados\\inversa2.txt";
52
53     // Llena matriz 1 y matriz 2
54     llenar(matriz1, n);

```

```

53     llenar(matriz2, n);
54
55     printf("MATRIZ 1\n"); imprimir(matriz1, n);
56     printf("MATRIZ 2\n"); imprimir(matriz2, n);
57
58     inversa(matriz1, inversa1, n);
59     inversa(matriz2, inversa2, n);
60     crearArchivo(dir1, inversa1);
61     crearArchivo(dir2, inversa2);
62
63     printf("ARCHIVOS INVERSAS ESCRITO\n");
64     printf("%s", dir1);
65     printf("\n");
66     printf("%s", dir2);
67     printf("\n");
68
69     //LOS ARCHIVOS NO DEBEN EXISTIR, SINO ARROJARA ERROR Y NO ESCRIBIRA NADA
70
71     STARTUPINFO si;
72     PROCESS_INFORMATION pi;
73     i = 0;
74     ZeroMemory(&si, sizeof(si));
75     si.cb=sizeof(si);
76     ZeroMemory(&pi, sizeof(pi));
77     if(argc!=2)
78     {
79         //printf(/*Usar: %s */"Soy el proceso hijo \n"/*, argv[0]*/);
80         return 0;
81     }
82     //Creacion proceso hijo
83
84     if(!CreateProcess(NULL, argv[0],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi))
85     {
86         printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
87         return 0;
88     }
89     //Proceso Padre
90     WaitForSingleObject(pi.hProcess, INFINITE);
91
92     //Terminacion controlada del proceso e hilo asociado de ejecucion
93     CloseHandle(pi.hProcess);
94     CloseHandle(pi.hThread);
95     return 0;
96 }
97
98 void crearArchivo(char *ruta, double **m)
99 {
100     int i,j;
101     char num[20];
102     HANDLE h = CreateFile(ruta,                                //ruta del archivo
103                           GENERIC_WRITE,                      //abrir para escribir
104                           0,                                    //no compartir
105                           NULL,                                // seguridad por default
106                           CREATE_ALWAYS,                      //crear siempre
107                           FILE_ATTRIBUTE_TEMPORARY,           //archivo normal
108                           NULL);                                //sin tributos
109
110     if (h == INVALID_HANDLE_VALUE)
111     {
112         perror(ruta);
113         exit(EXIT_FAILURE);
114     }
115     else
116     {
117         DWORD bytesEscritos = 0;
118
119         for(i=0 ; i<10 ; i++) // Escribimos 5 veces el texto en el archivo
120         {
121             for(j=0 ; j<10 ; j++) // Escribimos 5 veces el texto en el archivo

```



```

122     {
123         sprintf(num, "%.3f\t", m[i][j]);
124         /*Funcion WriteFile recibe los parametros a continuacion y devuelve
           ↳ un true si no existieron errores*/
125         BOOL escribir = WriteFile(
126             h,                                     // abrir handle del archivo
127             num,                                   // informacion a escribir
128             (DWORD)strlen(num),                    // tamaño de bytes a
           ↳ escribir
129             &bytesEscritos,                        // tamaño de bytes escrit
130             NULL);                                // no overlapped structure
131
132         if(!escribir)
133         {
134             perror(ruta);
135             exit(EXIT_FAILURE);
136         }
137     }
138     BOOL espacio = WriteFile(
139         h,                                     // abrir handle del archivo
140         "\n",                                   // informacion a escribir
141         (DWORD)strlen("\n"),                    // tamaño de bytes a
           ↳ escribir
142         &bytesEscritos,                        // tamaño de bytes escrit
143         NULL);                                // no overlapped structure
144     if(!espacio)
145     {
146         perror(ruta);
147         exit(EXIT_FAILURE);
148     }
149 }
150 //Llamada al sistema CloseHandle recibe un descriptor de archivo y retorna
           ↳ un valor cero si han habido errores
151 if(CloseHandle(h) == 0)
152 {
153     perror(ruta);
154     exit(EXIT_FAILURE);
155 }
156 }
157 }
158
159 void imprimir(double **m, int n)
160 {
161     int i, j;
162     for(i = 0; i < n; i++)
163     {
164         for(j = 0; j < n; j++)
165             printf("%.3f\t", m[i][j]);
166         printf("\n");
167     }
168     printf("\n");
169 }
170
171 // Llena con numeros random
172 void llenar(double **m, int n)
173 {
174     int i, j;
175     for (i = 0; i < n; i++)
176     {
177         for (j = 0; j < n; j++)
178         {
179             m[i][j] = (rand()%11);
180         }
181     }
182 }
183
184 bool esCero(double x)
185 {
186     return fabs(x) < 1e-8;

```

```

187 }
188
189 double determinante(double **m, int n)
190 {
191     double det = 0, aux = 0;
192     int c;
193     //Si el orden es de 2, multiplica cruzadon directamente
194     if(n==2)
195         return m[0][0]*m[1][1] - m[1][0]*m[0][1];
196     else
197     {
198         for(int j=0; j<n; j++)
199         {
200             //Crea arreglo dinamico temporal
201             double **menor = (double **)malloc(sizeof(double)*(n-1));
202             //Redimensiona
203             for(int i=0; i<(n-1); i++)
204                 menor[i] = (double *)malloc(sizeof(double)*(n-1));
205             for(int k=1; k<n; k++)
206             {
207                 c = 0;
208                 for(int l=0; l<n; l++)
209                 {
210                     if(l!=j)
211                     {
212                         /*Parte matriz principal en matrices de 3
213                         y multiplica cruzado*/
214                         menor[k-1][c] = m[k][l];
215                         c++;
216                     }
217                 }
218             }
219             //Recurividad, repite la funcion
220             aux = potencia(-1, 2+j)*m[0][j]*determinante(menor, n-1);
221             det += aux;
222
223             for(int x = 0; x<(n-1); x++)
224                 free(menor[x]); //Libera espacio en memoria
225             free(menor);
226         }
227         return det; //Devuelve resultado
228     }
229 }
230
231 // Usando definicion de la adjunta
232 int inversa(double **A, double **resultado, int n)
233 {
234     int tieneInversa;
235     if(determinante(A, n) == 0)
236     {
237         tieneInversa=0;
238         printf("La matriz no tiene inversa. Determinante = 0\n\n");
239     }
240     else{
241         tieneInversa=1;
242         int i, j, k, l;
243         double *tmp;
244         tmp = (double*)malloc(sizeof(double)*n);
245
246         for(i = 0; i < n; ++i)
247             resultado[i][i] = 1;
248         i = 0; j = 0;
249         while(i < n && j < n)
250         {
251             if(esCero(A[i][j]))
252             {
253                 for(k = i + 1; k < n; ++k)
254                 {
255                     if(!esCero(A[k][j]))

```

```

256         {
257             tmp = A[i];
258             A[i] = A[k];
259             A[k] = tmp;
260             tmp = resultado[i];
261             resultado[i] = resultado[k];
262             resultado[k] = tmp;
263             break;
264         }
265     }
266 }
267 if(!esCero(A[i][j]))
268 {
269     for(l = 0; l < n; ++l)
270         resultado[i][l] /= A[i][j];
271     for(l = n - 1; l >= j; --l)
272         A[i][l] /= A[i][j];
273     for(k = 0; k < n; ++k)
274     {
275         if(i == k) continue;
276         for(l = 0; l < n; ++l)
277             resultado[k][l] -= resultado[i][l] * A[k][j];
278         for(l = n; l >= j; --l)
279             A[k][l] -= A[i][l] * A[k][j];
280     }
281     ++i;
282 }
283 ++j;
284 }
285 }
286 return tieneInversa;
287 }
288
289 int potencia(int base, int pot)
290 {
291     int i, resultado = 1;
292     for(i = 0; i < pot; i++)
293         resultado = base * resultado;
294
295     return resultado;
296 }

```

#### o Leer archivos con resultados

```

1  #include <windows.h>
2  #include <time.h>
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <dirent.h>
8  #include <sys/stat.h>
9
10 // Declaracion de funciones
11 void imprimirArchivo(char *directorio, char *nombre);
12
13 int main(int argc, char const *argv[])
14 {
15     char* path = (char*)calloc(2000, sizeof(char));
16     strcpy(path, "C:\\Users\\YaKerTaker\\Google Drive\\5to
    ↳ SEMESTRE\\Sistemas-Operativos\\Practica4\\Windows\\8\\Resultados");
17     printf(" ----- \n");
18     printf(" ----- RESULTADOS ----- \n");
19     printf(" ----- \n");
20
21     printf("SUMA\n"); imprimirArchivo(path, "suma.txt");
22     printf("\nRESTA\n"); imprimirArchivo(path, "resta.txt");
23     printf("\nMULTIPLICAR\n"); imprimirArchivo(path, "multiplicacion.txt");
24     printf("\nTRANSPUESTA MATRIZ 1\n"); imprimirArchivo(path, "transpuesta1.txt");
25     printf("\nTRANSPUESTA MATRIZ 2\n"); imprimirArchivo(path, "transpuesta2.txt");

```


```
26     printf("\nINVERSA MATRIZ 1\n"); imprimirArchivo(path, "inversal.txt");
27     printf("\nINVERSA MATRIZ 2\n"); imprimirArchivo(path, "inversa2.txt");
28
29     return 0;
30 }
31
32 void imprimirArchivo(char *directorio, char *nombre)
33 {
34     char *name = (char *)calloc(150, sizeof(char));
35     strcpy(name, nombre);
36     char *dir = (char *)calloc(150, sizeof(char));
37     // Contatenamos la ruta original con el nombre del archivo
38     strcat(strcat(strcpy(dir, directorio), "\\"), name);
39
40     HANDLE file;
41     DWORD BytesEscritos = 0;
42     char *contenido = (char*)calloc(1000000, sizeof(char));
43     file = CreateFile(
44         dir,
45         GENERIC_WRITE | GENERIC_READ,
46         FILE_SHARE_READ,
47         NULL,
48         OPEN_EXISTING,
49         FILE_ATTRIBUTE_NORMAL,
50         NULL);
51
52     if(file == INVALID_HANDLE_VALUE)
53     {
54         printf("Error 1\n");
55         perror(dir);
56         exit(EXIT_FAILURE);
57     }
58     else
59     {
60         if(ReadFile(file, contenido, 1000000, &BytesEscritos, NULL))
61         {
62             printf("%s", contenido);
63         }
64         free(contenido);
65
66         if(CloseHandle(file) == 0)
67         {
68             perror(dir);
69             exit(EXIT_FAILURE);
70         }
71     }
72
73     free(name);
74     free(dir);
75 }
```

## 2.3. Pantallas de ejecución de los programas desarrollados

### 2.3.1. Sección Linux:

- ✓ **Punto 3:** Creación de procesos por copia exacta de código

- **Primer código**

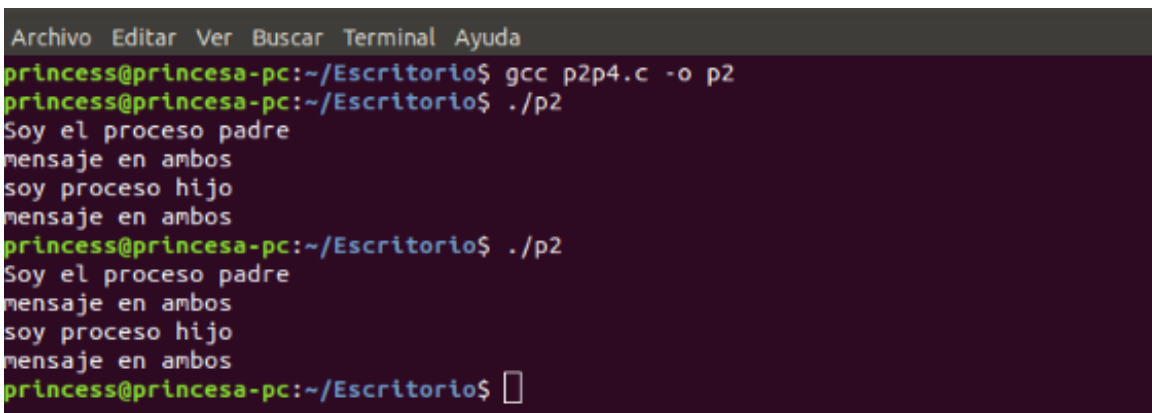


```
Archivo Editar Ver Buscar Terminal Ayuda
princess@princesa-pc:~$ pwd
/home/princess
princess@princesa-pc:~$ cd /home/princess/Escritorio
princess@princesa-pc:~/Escritorio$ gcc p1p4.c -o p
princess@princesa-pc:~/Escritorio$ ./p
Soy el proceso padre
soy proceso hijo
princess@princesa-pc:~/Escritorio$ ./p
Soy el proceso padre
soy proceso hijo
princess@princesa-pc:~/Escritorio$
```

Figura 1: Ejecución del primer código

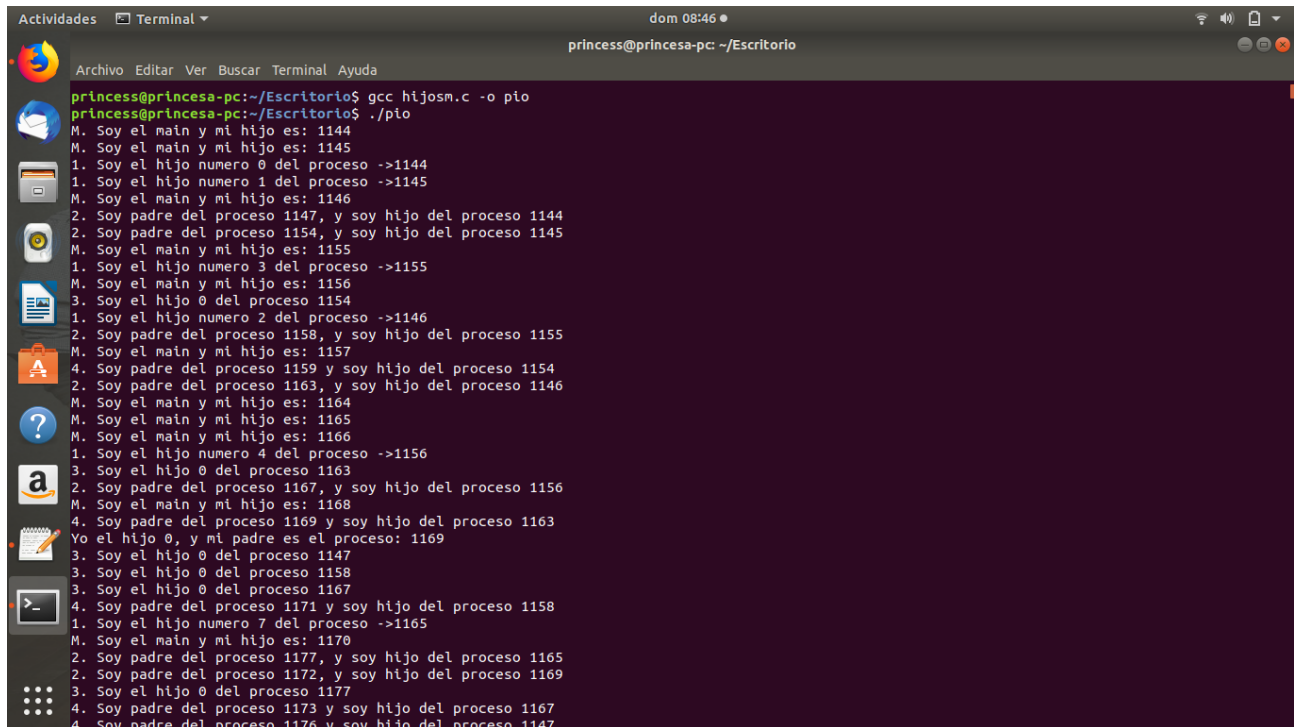
- **Segundo código**

En el funcionamiento del segundo código, en esta caso, pasa el primer proceso e imprime el ultimo mensaje, posteriormente el siguiente proceso, e igualmente imprime el mensaje, y termina el programa.



```
Archivo Editar Ver Buscar Terminal Ayuda
princess@princesa-pc:~/Escritorio$ gcc p2p4.c -o p2
princess@princesa-pc:~/Escritorio$ ./p2
Soy el proceso padre
mensaje en ambos
soy proceso hijo
mensaje en ambos
princess@princesa-pc:~/Escritorio$ ./p2
Soy el proceso padre
mensaje en ambos
soy proceso hijo
mensaje en ambos
princess@princesa-pc:~/Escritorio$
```

Figura 2: Ejecución del segundo código

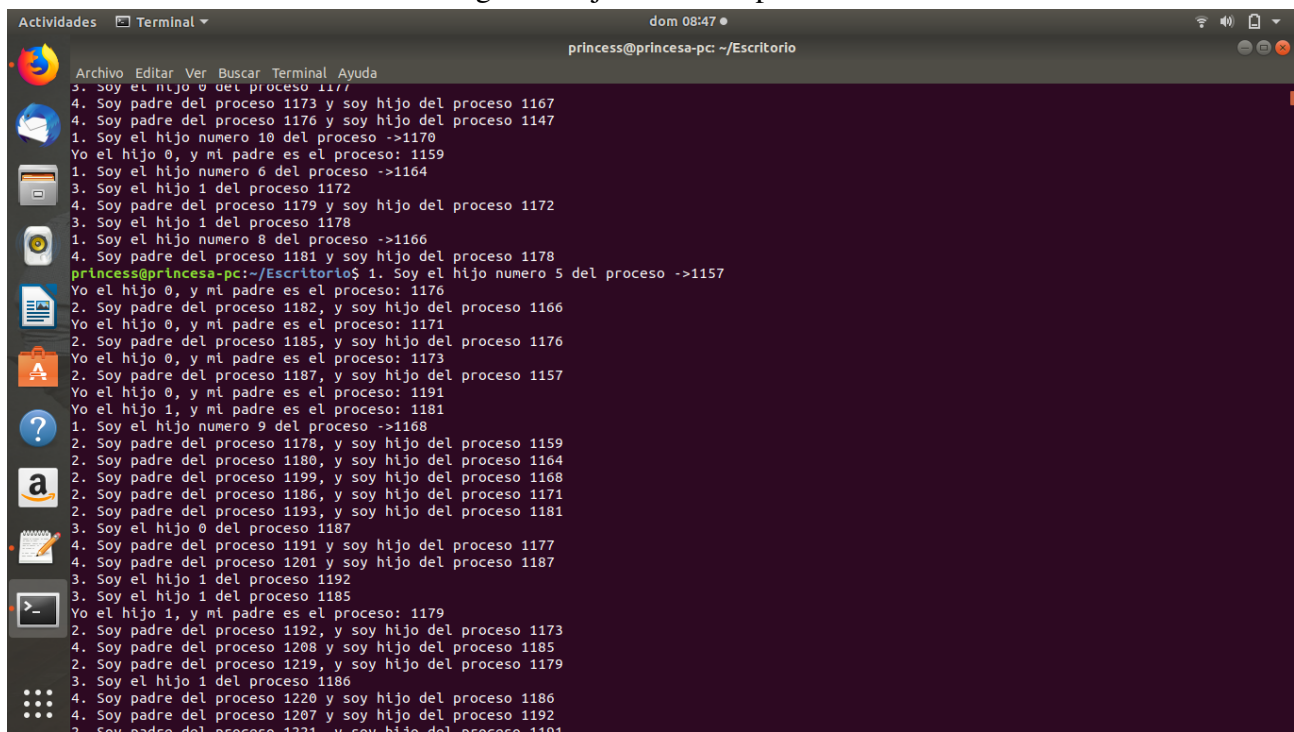


```

princess@princesa-pc: ~/Escritorio
princess@princesa-pc:~/Escritorio$ gcc hijosm.c -o pio
princess@princesa-pc:~/Escritorio$ ./pio
M. Soy el main y mi hijo es: 1144
M. Soy el main y mi hijo es: 1145
1. Soy el hijo numero 0 del proceso ->1144
1. Soy el hijo numero 1 del proceso ->1145
M. Soy el main y mi hijo es: 1146
2. Soy padre del proceso 1147, y soy hijo del proceso 1144
2. Soy padre del proceso 1154, y soy hijo del proceso 1145
M. Soy el main y mi hijo es: 1155
1. Soy el hijo numero 3 del proceso ->1155
M. Soy el main y mi hijo es: 1156
3. Soy el hijo 0 del proceso 1154
1. Soy el hijo numero 2 del proceso ->1146
2. Soy padre del proceso 1158, y soy hijo del proceso 1155
M. Soy el main y mi hijo es: 1157
4. Soy padre del proceso 1159 y soy hijo del proceso 1154
2. Soy padre del proceso 1163, y soy hijo del proceso 1146
M. Soy el main y mi hijo es: 1164
M. Soy el main y mi hijo es: 1165
M. Soy el main y mi hijo es: 1166
1. Soy el hijo numero 4 del proceso ->1156
3. Soy el hijo 0 del proceso 1163
2. Soy padre del proceso 1167, y soy hijo del proceso 1156
M. Soy el main y mi hijo es: 1168
4. Soy padre del proceso 1169 y soy hijo del proceso 1163
Yo el hijo 0, y mi padre es el proceso: 1169
3. Soy el hijo 0 del proceso 1147
3. Soy el hijo 0 del proceso 1158
3. Soy el hijo 0 del proceso 1167
4. Soy padre del proceso 1171 y soy hijo del proceso 1158
1. Soy el hijo numero 7 del proceso ->1165
M. Soy el main y mi hijo es: 1170
2. Soy padre del proceso 1177, y soy hijo del proceso 1165
2. Soy padre del proceso 1172, y soy hijo del proceso 1169
3. Soy el hijo 0 del proceso 1177
4. Soy padre del proceso 1173 y soy hijo del proceso 1167
4. Soy padre del proceso 1176 y soy hijo del proceso 1147

```

Figura 5: Ejecución del punto 4



```

princess@princesa-pc: ~/Escritorio
princess@princesa-pc:~/Escritorio$ gcc hijosm.c -o pio
princess@princesa-pc:~/Escritorio$ ./pio
M. Soy el main y mi hijo es: 1144
M. Soy el main y mi hijo es: 1145
1. Soy el hijo numero 0 del proceso ->1144
1. Soy el hijo numero 1 del proceso ->1145
M. Soy el main y mi hijo es: 1146
2. Soy padre del proceso 1147, y soy hijo del proceso 1144
2. Soy padre del proceso 1154, y soy hijo del proceso 1145
M. Soy el main y mi hijo es: 1155
1. Soy el hijo numero 3 del proceso ->1155
M. Soy el main y mi hijo es: 1156
3. Soy el hijo 0 del proceso 1154
1. Soy el hijo numero 2 del proceso ->1146
2. Soy padre del proceso 1158, y soy hijo del proceso 1155
M. Soy el main y mi hijo es: 1157
4. Soy padre del proceso 1159 y soy hijo del proceso 1154
2. Soy padre del proceso 1163, y soy hijo del proceso 1146
M. Soy el main y mi hijo es: 1164
M. Soy el main y mi hijo es: 1165
M. Soy el main y mi hijo es: 1166
1. Soy el hijo numero 4 del proceso ->1156
3. Soy el hijo 0 del proceso 1163
2. Soy padre del proceso 1167, y soy hijo del proceso 1156
M. Soy el main y mi hijo es: 1168
4. Soy padre del proceso 1169 y soy hijo del proceso 1163
Yo el hijo 0, y mi padre es el proceso: 1169
3. Soy el hijo 0 del proceso 1147
3. Soy el hijo 0 del proceso 1158
3. Soy el hijo 0 del proceso 1167
4. Soy padre del proceso 1171 y soy hijo del proceso 1158
1. Soy el hijo numero 7 del proceso ->1165
M. Soy el main y mi hijo es: 1170
2. Soy padre del proceso 1177, y soy hijo del proceso 1165
2. Soy padre del proceso 1172, y soy hijo del proceso 1169
3. Soy el hijo 0 del proceso 1177
4. Soy padre del proceso 1173 y soy hijo del proceso 1167
4. Soy padre del proceso 1176 y soy hijo del proceso 1147
1. Soy el hijo numero 10 del proceso ->1170
Yo el hijo 0, y mi padre es el proceso: 1159
1. Soy el hijo numero 6 del proceso ->1164
3. Soy el hijo 1 del proceso 1172
4. Soy padre del proceso 1179 y soy hijo del proceso 1172
3. Soy el hijo 1 del proceso 1178
1. Soy el hijo numero 8 del proceso ->1166
4. Soy padre del proceso 1181 y soy hijo del proceso 1178
princess@princesa-pc:~/Escritorio$ 1. Soy el hijo numero 5 del proceso ->1157
Yo el hijo 0, y mi padre es el proceso: 1176
2. Soy padre del proceso 1182, y soy hijo del proceso 1166
Yo el hijo 0, y mi padre es el proceso: 1171
2. Soy padre del proceso 1185, y soy hijo del proceso 1176
Yo el hijo 0, y mi padre es el proceso: 1173
2. Soy padre del proceso 1187, y soy hijo del proceso 1157
Yo el hijo 0, y mi padre es el proceso: 1191
Yo el hijo 1, y mi padre es el proceso: 1181
1. Soy el hijo numero 9 del proceso ->1168
2. Soy padre del proceso 1178, y soy hijo del proceso 1159
2. Soy padre del proceso 1180, y soy hijo del proceso 1164
2. Soy padre del proceso 1199, y soy hijo del proceso 1168
2. Soy padre del proceso 1186, y soy hijo del proceso 1171
2. Soy padre del proceso 1193, y soy hijo del proceso 1181
3. Soy el hijo 0 del proceso 1187
4. Soy padre del proceso 1191 y soy hijo del proceso 1177
4. Soy padre del proceso 1201 y soy hijo del proceso 1187
3. Soy el hijo 1 del proceso 1192
3. Soy el hijo 1 del proceso 1185
Yo el hijo 1, y mi padre es el proceso: 1179
2. Soy padre del proceso 1192, y soy hijo del proceso 1173
4. Soy padre del proceso 1208 y soy hijo del proceso 1185
2. Soy padre del proceso 1219, y soy hijo del proceso 1179
3. Soy el hijo 1 del proceso 1186
4. Soy padre del proceso 1220 y soy hijo del proceso 1186
4. Soy padre del proceso 1207 y soy hijo del proceso 1192
2. Soy padre del proceso 1221 y soy hijo del proceso 1181

```

Figura 6: Ejecución del punto 4

```

dom 08:47
princess@princesa-pc: ~/Escritorio

Archivo Editar Ver Buscar Terminal Ayuda
1. Soy el hijo numero 5 del proceso ->1338
M. Soy el main y mi hijo es: 1378
Yo el hijo 0, y mi padre es el proceso: 1372
2. Soy padre del proceso 1380, y soy hijo del proceso 1372
M. Soy el main y mi hijo es: 1382
Yo el hijo 1, y mi padre es el proceso: 1377
M. Soy el main y mi hijo es: 1383
1. Soy el hijo numero 8 del proceso ->1373
2. Soy padre del proceso 1385, y soy hijo del proceso 1373
1. Soy el hijo numero 9 del proceso ->1374
3. Soy el hijo 1 del proceso 1380
3. Soy el hijo 1 del proceso 1381
4. Soy padre del proceso 1387 y soy hijo del proceso 1380
4. Soy padre del proceso 1388 y soy hijo del proceso 1381
2. Soy padre del proceso 1389, y soy hijo del proceso 1374
1. Soy el hijo numero 8 del proceso ->1382
2. Soy padre del proceso 1390, y soy hijo del proceso 1382
Yo el hijo 1, y mi padre es el proceso: 1387
1. Soy el hijo numero 9 del proceso ->1383
3. Soy el hijo 0 del proceso 1389
2. Soy padre del proceso 1392, y soy hijo del proceso 1383
4. Soy padre del proceso 1393 y soy hijo del proceso 1389
1. Soy el hijo numero 7 del proceso ->1378
3. Soy el hijo 0 del proceso 1385
4. Soy padre del proceso 1394 y soy hijo del proceso 1385
2. Soy padre del proceso 1395, y soy hijo del proceso 1378
1. Soy el hijo numero 9 del proceso ->1391
3. Soy el hijo 0 del proceso 1390
4. Soy padre del proceso 1396 y soy hijo del proceso 1390
2. Soy padre del proceso 1397, y soy hijo del proceso 1391
Yo el hijo 0, y mi padre es el proceso: 1394
3. Soy el hijo 0 del proceso 1392
2. Soy padre del proceso 1398, y soy hijo del proceso 1394
4. Soy padre del proceso 1399 y soy hijo del proceso 1392
Yo el hijo 0, y mi padre es el proceso: 1396
Yo el hijo 0, y mi padre es el proceso: 1393
3. Soy el hijo 0 del proceso 1395
M. Soy el main y mi hijo es: 1400

```

Figura 7: Ejecución del punto 4

```

dom 08:48
princess@princesa-pc: ~/Escritorio

Archivo Editar Ver Buscar Terminal Ayuda
1. Soy el hijo numero 9 del proceso ->4622
3. Soy el hijo 0 del proceso 4624
4. Soy padre del proceso 4628 y soy hijo del proceso 4626
4. Soy padre del proceso 4629 y soy hijo del proceso 4624
Yo el hijo 0, y mi padre es el proceso: 4625
3. Soy el hijo 0 del proceso 4627
Yo el hijo 0, y mi padre es el proceso: 4628
4. Soy padre del proceso 4630 y soy hijo del proceso 4627
2. Soy padre del proceso 4631, y soy hijo del proceso 4622
M. Soy el main y mi hijo es: 4632
M. Soy el main y mi hijo es: 4633
Yo el hijo 0, y mi padre es el proceso: 4630
2. Soy padre del proceso 4634, y soy hijo del proceso 4630
1. Soy el hijo numero 10 del proceso ->4623
Yo el hijo 0, y mi padre es el proceso: 4629
3. Soy el hijo 0 del proceso 4631
M. Soy el main y mi hijo es: 4635
4. Soy padre del proceso 4636 y soy hijo del proceso 4631
3. Soy el hijo 1 del proceso 4634
1. Soy el hijo numero 10 del proceso ->4633
4. Soy padre del proceso 4637 y soy hijo del proceso 4634
1. Soy el hijo numero 10 del proceso ->4635
1. Soy el hijo numero 10 del proceso ->4632
Yo el hijo 1, y mi padre es el proceso: 4637
Yo el hijo 0, y mi padre es el proceso: 4636
M. Soy el main y mi hijo es: 4638
M. Soy el main y mi hijo es: 4639
M. Soy el main y mi hijo es: 4640
1. Soy el hijo numero 9 del proceso ->4638
1. Soy el hijo numero 10 del proceso ->4639
2. Soy padre del proceso 4641, y soy hijo del proceso 4638
1. Soy el hijo numero 10 del proceso ->4640
3. Soy el hijo 0 del proceso 4641
4. Soy padre del proceso 4642 y soy hijo del proceso 4641
Yo el hijo 0, y mi padre es el proceso: 4642
M. Soy el main y mi hijo es: 4643
1. Soy el hijo numero 10 del proceso ->4643
princess@princesa-pc:~/Escritorio$

```

Figura 8: Ejecución del punto 4

Se crean los procesos con un numero de identificador diferente cada vez que se ejecuta el programa. El administrador de procesos es quien va marcando la prioridad para el orden de la construcción de los procesos.

```

MATRIZ 2
8.000 5.000 6.000 8.000 10.000 5.000 10.000 10.000 3.000 2.000
2.000 10.000 1.000 10.000 3.000 8.000 0.000 6.000 9.000 5.000
8.000 10.000 3.000 5.000 1.000 3.000 2.000 9.000 7.000 7.000
10.000 4.000 10.000 6.000 10.000 7.000 9.000 9.000 6.000 2.000
9.000 7.000 10.000 0.000 4.000 1.000 6.000 5.000 5.000 2.000
8.000 2.000 0.000 9.000 7.000 1.000 1.000 8.000 10.000 8.000
2.000 8.000 10.000 1.000 1.000 9.000 6.000 10.000 6.000 1.000
1.000 2.000 6.000 10.000 0.000 0.000 0.000 4.000 3.000 5.000
7.000 0.000 5.000 7.000 9.000 2.000 6.000 8.000 8.000 4.000
5.000 10.000 1.000 2.000 9.000 2.000 9.000 4.000 1.000 4.000

SUMA
RESTA
MULTIPLICAR
TRANSPUESTA MATRIZ 1
TRANSPUESTA MATRIZ 2
INVERSA MATRIZ 1
INVERSA MATRIZ 2
-----
----- RESULTADOS -----
-----
SUMA
9.000 5.000 12.000 11.000 10.000 8.000 16.000 14.000 8.000 4.000

```

Figura 11: Realiza operaciones de matrices de forma secuencial y genera archivos de resultados

```

-----
SUMA
9.000 5.000 12.000 11.000 10.000 8.000 16.000 14.000 8.000 4.000
7.000 14.000 8.000 18.000 5.000 13.000 9.000 13.000 19.000 13.000
12.000 13.000 11.000 6.000 2.000 6.000 12.000 12.000 11.000 8.000
13.000 7.000 11.000 13.000 16.000 7.000 19.000 19.000 8.000 4.000
11.000 14.000 16.000 7.000 7.000 9.000 7.000 6.000 7.000 11.000
17.000 9.000 10.000 16.000 13.000 10.000 9.000 11.000 12.000 18.000
6.000 13.000 13.000 5.000 2.000 16.000 10.000 19.000 11.000 5.000
1.000 7.000 6.000 15.000 1.000 3.000 2.000 4.000 6.000 8.000
14.000 1.000 13.000 11.000 15.000 5.000 8.000 11.000 14.000 8.000
6.000 19.000 10.000 6.000 11.000 10.000 18.000 8.000 5.000 5.000

RESTA
-7.000 -5.000 0.000 -5.000 -10.000 -2.000 -4.000 -6.000 2.000 0.000
3.000 -6.000 6.000 -2.000 -1.000 -3.000 9.000 1.000 1.000 3.000
-4.000 -7.000 5.000 -4.000 0.000 0.000 8.000 -6.000 -3.000 -6.000
-7.000 -1.000 -9.000 1.000 -4.000 -7.000 1.000 1.000 -4.000 0.000
-7.000 0.000 -4.000 7.000 -1.000 7.000 -5.000 -4.000 -3.000 7.000
1.000 5.000 10.000 -2.000 -1.000 8.000 7.000 -5.000 -8.000 2.000
2.000 -3.000 -7.000 3.000 0.000 -2.000 -2.000 -1.000 -1.000 3.000
-1.000 3.000 -6.000 -5.000 1.000 3.000 2.000 -4.000 0.000 -2.000
0.000 1.000 3.000 -3.000 -3.000 1.000 -4.000 -5.000 -2.000 0.000
-4.000 -1.000 8.000 2.000 -7.000 6.000 0.000 0.000 3.000 -3.000

```

Figura 12: Lee resultados de suma y resta

```

MULTIPLICAR
171.000 159.000 165.000 168.000 136.000 115.000 136.000 239.000 183.000 128.000
377.000 357.000 345.000 373.000 363.000 258.000 339.000 489.000 371.000 255.000
201.000 243.000 210.000 205.000 147.000 179.000 167.000 316.000 238.000 152.000
216.000 245.000 326.000 229.000 180.000 195.000 221.000 314.000 223.000 130.000
301.000 305.000 172.000 273.000 285.000 175.000 220.000 324.000 282.000 215.000
445.000 473.000 339.000 387.000 394.000 291.000 368.000 535.000 405.000 292.000
243.000 227.000 211.000 321.000 236.000 159.000 185.000 326.000 272.000 205.000
133.000 129.000 103.000 136.000 146.000 109.000 111.000 160.000 149.000 87.000
309.000 251.000 239.000 239.000 258.000 142.000 245.000 334.000 243.000 176.000
275.000 321.000 237.000 318.000 204.000 233.000 171.000 397.000 360.000 235.000

TRANSPUESTA MATRIZ 1
1.000 5.000 4.000 3.000 2.000 9.000 4.000 0.000 7.000 1.000
0.000 4.000 3.000 3.000 7.000 7.000 5.000 5.000 1.000 9.000
6.000 7.000 8.000 1.000 6.000 10.000 3.000 0.000 8.000 9.000
3.000 8.000 1.000 7.000 7.000 7.000 4.000 5.000 4.000 4.000
0.000 2.000 1.000 6.000 3.000 6.000 1.000 1.000 6.000 2.000
3.000 5.000 3.000 0.000 8.000 9.000 7.000 3.000 3.000 8.000
6.000 9.000 10.000 10.000 1.000 8.000 4.000 2.000 2.000 9.000
4.000 7.000 3.000 10.000 1.000 3.000 9.000 0.000 3.000 4.000
5.000 10.000 4.000 2.000 2.000 2.000 5.000 3.000 6.000 4.000
2.000 8.000 1.000 2.000 9.000 10.000 4.000 3.000 4.000 1.000

```

Figura 13: Lee resultados de multiplicación y transpuestas



```

1.568 -4.767 6.680 0.842 4.079 -2.848 2.002 3.631 0.995 -4.475
3.462 -8.972 12.116 1.537 7.424 -5.096 3.642 6.691 1.870 -8.323
7.024 -17.599 23.310 2.990 14.073 -9.651 7.168 13.528 3.583 -16.280
-6.823 17.492 -23.545 -2.921 -14.191 9.751 -7.252 -13.361 -3.486 16.340
-3.585 9.840 -13.595 -1.764 -8.296 5.755 -4.023 -7.442 -2.053 9.320
-2.565 6.776 -9.067 -1.138 -5.571 3.850 -2.820 -5.103 -1.450 6.291
1.000 -2.724 3.722 0.521 2.335 -1.625 1.200 1.933 0.554 -2.558
-3.150 8.076 -10.711 -1.401 -6.517 4.390 -3.283 -6.030 -1.569 7.450
-3.600 9.013 -11.783 -1.491 -6.997 4.869 -3.681 -6.935 -1.878 8.180

INVERSA MATRIZ 2
-0.072 0.014 0.247 0.156 -0.065 -0.229 -0.166 -0.048 0.153 -0.054
0.170 0.047 -0.192 -0.161 0.184 0.222 0.067 -0.010 -0.266 -0.000
0.045 -0.050 -0.242 0.001 0.133 0.281 0.119 0.082 -0.306 0.007
0.018 0.088 0.144 -0.007 -0.048 -0.265 -0.137 0.029 0.233 -0.048
0.283 -0.062 -0.637 -0.139 0.285 0.823 0.262 0.016 -0.789 0.036
-0.224 0.005 0.193 0.302 -0.229 -0.200 -0.055 -0.006 0.115 0.039
-0.379 0.075 0.711 0.157 -0.369 -1.041 -0.315 -0.027 1.088 0.064
0.349 -0.101 -0.336 -0.246 0.173 0.557 0.248 -0.015 -0.514 -0.066
-0.137 0.119 0.187 -0.062 -0.018 -0.353 -0.103 -0.072 0.487 -0.032
-0.341 -0.127 0.185 0.280 -0.252 -0.083 0.001 0.113 0.096 0.170

Tiempo ejecucion: 0.8645 s

```

Figura 14: Lee resultados de inversas y muestra el tiempo de ejecución.

	inversa_2.txt	mul.txt	suma.txt	tran1.txt
1	1.000	5.000	4.000	3.000
2	0.000	4.000	3.000	7.000
3	6.000	7.000	8.000	1.000
4	3.000	8.000	1.000	7.000
5	0.000	2.000	1.000	6.000
6	3.000	5.000	3.000	0.000
7	6.000	9.000	10.000	1.000
8	4.000	7.000	3.000	10.000
9	5.000	10.000	4.000	2.000
10	2.000	8.000	1.000	2.000

Figura 15: Archivos de resultados generados

- Aplicación con seis procesos por copia exacta de código

```

enrike@enrike:~/Escritorio/P4-S0/5$ gcc 5.c tiempo.c -o 5
enrike@enrike:~/Escritorio/P4-S0/5$ ./5
Ingrese el nuevo directorio: resultados2
MATRIZ 1
0.000 6.000 10.000 8.000 2.000 7.000 5.000 2.000 8.000 1.000
7.000 10.000 6.000 10.000 5.000 3.000 8.000 6.000 0.000 7.000
2.000 3.000 3.000 9.000 0.000 4.000 5.000 7.000 3.000 6.000
3.000 3.000 2.000 0.000 10.000 4.000 8.000 2.000 7.000 3.000
3.000 1.000 2.000 9.000 9.000 7.000 1.000 6.000 0.000 10.000
0.000 2.000 2.000 1.000 10.000 1.000 6.000 4.000 8.000 7.000
9.000 10.000 0.000 9.000 10.000 8.000 0.000 7.000 10.000 7.000
10.000 2.000 6.000 10.000 9.000 4.000 4.000 9.000 9.000 4.000
8.000 9.000 5.000 0.000 9.000 4.000 10.000 4.000 6.000 5.000
0.000 2.000 4.000 9.000 0.000 4.000 6.000 1.000 0.000 3.000

MATRIZ 2
6.000 9.000 5.000 2.000 8.000 2.000 4.000 2.000 0.000 0.000
4.000 8.000 10.000 9.000 6.000 6.000 0.000 5.000 10.000 5.000
0.000 8.000 7.000 2.000 7.000 8.000 6.000 0.000 9.000 5.000
4.000 2.000 3.000 7.000 4.000 9.000 9.000 7.000 9.000 9.000
7.000 1.000 7.000 4.000 10.000 0.000 10.000 9.000 4.000 7.000
3.000 4.000 5.000 10.000 4.000 10.000 5.000 0.000 10.000 1.000
5.000 1.000 4.000 6.000 9.000 6.000 2.000 7.000 2.000 10.000
4.000 8.000 0.000 0.000 10.000 8.000 9.000 8.000 6.000 2.000

```

Figura 16: Genera matrices aleatorias

```

4.000 8.000 0.000 0.000 10.000 8.000 9.000 8.000 6.000 2.000
4.000 7.000 4.000 7.000 5.000 9.000 6.000 10.000 9.000 6.000
1.000 1.000 7.000 3.000 5.000 3.000 9.000 7.000 0.000 10.000

----- TRANSPUESTA
----- INVERSA
----- MULTIPLICACION
----- RESTA
----- SUMA
----- RESULTADOS -----
SUMA
6.000 15.000 15.000 10.000 10.000 9.000 9.000 4.000 8.000 1.000
11.000 18.000 16.000 19.000 11.000 9.000 8.000 11.000 10.000 12.000
2.000 11.000 10.000 11.000 7.000 12.000 11.000 7.000 12.000 11.000
7.000 5.000 5.000 7.000 14.000 13.000 17.000 9.000 16.000 12.000
10.000 2.000 9.000 13.000 19.000 7.000 11.000 15.000 4.000 17.000
3.000 6.000 7.000 11.000 14.000 11.000 11.000 4.000 18.000 8.000
14.000 11.000 4.000 15.000 19.000 14.000 2.000 14.000 12.000 17.000
14.000 10.000 6.000 10.000 19.000 12.000 13.000 17.000 15.000 6.000
12.000 16.000 9.000 7.000 14.000 13.000 16.000 14.000 15.000 11.000
1.000 3.000 11.000 12.000 5.000 7.000 15.000 8.000 0.000 13.000

```

Figura 17: Operaciones con procesos y genera archivos de resultados. Lee resultados de suma

```

RESTA
-6.000 -3.000 5.000 6.000 -6.000 5.000 1.000 0.000 8.000 1.000
3.000 2.000 -4.000 1.000 -1.000 -3.000 8.000 1.000 -10.000 2.000
2.000 -5.000 -4.000 7.000 -7.000 -4.000 -1.000 7.000 -6.000 1.000
-1.000 1.000 -1.000 -7.000 6.000 -5.000 -1.000 -5.000 -2.000 -6.000
-4.000 0.000 -5.000 5.000 -1.000 7.000 -9.000 -3.000 -4.000 3.000
-3.000 -2.000 -3.000 -9.000 6.000 -9.000 1.000 4.000 -2.000 6.000
4.000 9.000 -4.000 3.000 1.000 2.000 -2.000 0.000 8.000 -3.000
6.000 -6.000 6.000 10.000 -1.000 -4.000 -5.000 1.000 3.000 2.000
4.000 2.000 1.000 -7.000 4.000 -5.000 4.000 -6.000 -3.000 -1.000
-1.000 1.000 -3.000 6.000 -5.000 1.000 -3.000 -6.000 0.000 -7.000

MULTIPLICAR
157.000 252.000 262.000 297.000 296.000 379.000 272.000 242.000 394.000 285.000
237.000 291.000 338.000 305.000 427.000 359.000 352.000 332.000 346.000 370.000
143.000 188.000 182.000 209.000 267.000 298.000 272.000 245.000 257.000 257.000
191.000 169.000 230.000 223.000 314.000 216.000 247.000 274.000 219.000 255.000
181.000 165.000 238.000 224.000 317.000 263.000 376.000 280.000 253.000 288.000
170.000 149.000 217.000 192.000 303.000 208.000 285.000 310.000 205.000 286.000
299.000 354.000 371.000 382.000 455.000 406.000 443.000 426.000 433.000 353.000
279.000 342.000 305.000 295.000 471.000 399.000 455.000 399.000 383.000 349.000
254.000 298.000 347.000 300.000 444.000 311.000 309.000 339.000 309.000 331.000
93.000 99.000 140.000 174.000 171.000 218.000 173.000 144.000 195.000 207.000

```

Figura 18: Lee resultados de resta y multiplicación

```

2.000 5.000 0.000 7.000 9.000 0.000 7.000 8.000 10.000 7.000
0.000 10.000 9.000 9.000 4.000 10.000 2.000 6.000 9.000 0.000
0.000 5.000 5.000 9.000 7.000 1.000 10.000 2.000 6.000 10.000

INVERSA MATRIZ 1
-0.081 -0.142 -0.081 -0.214 -0.004 0.008 0.031 0.082 0.213 0.192
0.044 0.134 0.015 0.082 -0.034 -0.018 0.034 -0.072 -0.087 -0.120
0.081 -0.021 -0.067 -0.145 0.052 0.038 -0.064 0.040 0.092 -0.018
-0.021 0.054 -0.050 0.023 -0.039 0.022 0.034 0.028 -0.102 0.093
0.027 0.139 -0.058 0.195 0.028 -0.024 -0.021 -0.002 -0.166 -0.124
0.041 -0.075 0.080 0.107 0.079 -0.138 0.001 -0.048 0.034 -0.045
-0.042 0.008 0.057 0.103 -0.041 -0.024 -0.026 -0.008 -0.005 0.051
0.064 0.184 0.232 0.315 0.017 -0.133 -0.077 -0.027 -0.224 -0.391
-0.012 -0.123 -0.018 -0.142 -0.060 0.098 0.055 0.031 0.087 0.125
-0.065 -0.200 -0.068 -0.386 0.043 0.173 0.036 0.002 0.278 0.247

INVERSA MATRIZ 2
-0.093 3.625 -2.429 2.858 -1.574 -0.440 -1.044 2.926 -4.987 1.427
0.251 -2.583 1.738 -1.862 1.080 0.201 0.711 -2.159 3.572 -1.101
-0.256 4.005 -2.620 2.847 -1.717 -0.379 -1.170 3.245 -5.396 1.744
0.267 -4.308 2.791 -3.217 1.877 0.532 1.251 -3.562 5.885 -1.782
0.057 -2.000 1.347 -1.590 0.918 0.258 0.652 -1.571 2.672 -0.852
-0.252 4.030 -2.701 2.967 -1.850 -0.363 -1.124 3.375 -5.520 1.757
0.085 -0.997 0.625 -0.663 0.435 0.130 0.209 -0.780 1.281 -0.357
-0.126 1.007 -0.710 0.625 -0.416 -0.102 -0.296 0.857 -1.242 0.460
-0.040 -0.317 0.289 -0.213 0.219 -0.003 0.071 -0.291 0.471 -0.243
0.142 -1.632 1.137 -1.063 0.684 0.063 0.507 -1.404 2.189 -0.720

Tiempo ejecucion: 0.7959 s

```

Figura 19: Lee resultados de inversas y muestra el tiempo de ejecución.

	inversa_1.txt	inversa_2.txt	tran1.txt	suma.txt
1	-0.093 3.625 -2.429 2.858 -1.574 -0.440 -1.044 2.926 -4.987 1.427			
2	0.251 -2.583 1.738 -1.862 1.080 0.201 0.711 -2.159 3.572 -1.101			
3	-0.256 4.005 -2.620 2.847 -1.717 -0.379 -1.170 3.245 -5.396 1.744			
4	0.267 -4.308 2.791 -3.217 1.877 0.532 1.251 -3.562 5.885 -1.782			
5	0.057 -2.000 1.347 -1.590 0.918 0.258 0.652 -1.571 2.672 -0.852			
6	-0.252 4.030 -2.701 2.967 -1.850 -0.363 -1.124 3.375 -5.520 1.757			
7	0.085 -0.997 0.625 -0.663 0.435 0.130 0.209 -0.780 1.281 -0.357			
8	-0.126 1.007 -0.710 0.625 -0.416 -0.102 -0.296 0.857 -1.242 0.460			
9	-0.040 -0.317 0.289 -0.213 0.219 -0.003 0.071 -0.291 0.471 -0.243			
10	0.142 -1.632 1.137 -1.063 0.684 0.063 0.507 -1.404 2.189 -0.720			

Figura 20: Archivos de resultados generados

### ✓ Punto 6: Creación de procesos por copia exacta de código

```

enrike@enrike:~/Escritorio/P4-SO/6$ gcc hola.c -o hola
enrike@enrike:~/Escritorio/P4-SO/6$ ./hola Mundo
Hola MundoMundo
enrike@enrike:~/Escritorio/P4-SO/6$ gcc 6.c -o 6
enrike@enrike:~/Escritorio/P4-SO/6$ ./6
Soy el hijo ejecutando: /home/enrike/Escritorio/P4-SO/6/hola
Hola MundoDesde el Hijo
Soy el Padre
enrike@enrike:~/Escritorio/P4-SO/6$

```

Figura 21: El programa que posee al proceso padre (en este caso el main del código) recibe como argumento la ruta del ejecutable del código que contendrá al proceso hijo (en este caso un "Hola Mundo").

```

6.000 8.000 9.000 1.000 7.000 8.000 0.000 6.000 1.000 3.000
9.000 5.000 4.000 9.000 2.000 7.000 6.000 2.000 4.000 3.000
10.000 4.000 3.000 10.000 9.000 5.000 8.000 1.000 0.000 5.000

INVERSA MATRIZ 1
0.146 -0.141 -0.175 -0.127 0.103 -0.001 0.094 0.019 0.007 0.142
-0.105 0.070 0.146 0.013 0.007 0.003 -0.029 0.114 0.006 -0.186
0.153 0.096 -0.034 -0.098 -0.096 0.001 -0.065 -0.142 0.053 0.154
0.070 -0.031 0.041 0.054 -0.019 -0.051 0.014 -0.018 -0.099 0.065
0.029 0.019 -0.092 -0.148 0.173 0.043 -0.132 -0.051 0.125 0.085
-0.136 0.045 0.086 -0.003 0.042 -0.008 -0.070 -0.062 0.135 -0.050
-0.040 -0.138 -0.035 0.130 -0.030 0.003 0.055 0.072 -0.046 -0.002
0.037 -0.055 -0.052 0.111 -0.170 0.075 0.199 0.064 -0.209 -0.001
-0.312 0.192 0.246 0.114 -0.029 -0.121 -0.048 0.169 0.029 -0.270
0.064 0.010 -0.058 -0.016 0.089 0.045 -0.067 -0.100 0.092 -0.016

INVERSA MATRIZ 2
-0.139 -0.207 0.137 0.105 0.065 -0.103 0.078 -0.047 -0.052 0.140
0.001 0.140 -0.038 -0.165 0.046 0.045 -0.036 0.048 0.303 -0.264
-0.004 -0.090 0.022 -0.095 0.123 -0.024 0.030 0.070 0.157 -0.157
0.042 0.028 -0.061 -0.101 -0.018 0.035 0.069 0.044 0.085 -0.057
0.149 0.088 -0.010 0.214 -0.287 -0.041 -0.001 0.011 -0.595 0.440
0.170 0.146 -0.089 0.230 -0.322 0.019 -0.125 -0.006 -0.430 0.349
-0.023 0.057 -0.039 -0.078 0.131 0.025 -0.063 -0.050 0.262 -0.171
-0.199 -0.118 -0.005 -0.177 0.308 0.041 0.122 0.050 0.431 -0.330
0.108 0.045 0.051 0.193 -0.202 -0.040 -0.003 -0.055 -0.363 0.238
-0.165 -0.074 0.037 -0.275 0.362 0.121 -0.081 -0.018 0.593 -0.463

Soy el padre de los hijitos
Soy el primer padre del hijo
enrike@enrike:~/Escritorio/P4-S0/7$

```

Figura 25: Proceso de inversas de matrices

```

enrike@enrike:~/Escritorio/P4-S0/7$ ls -l
total 84
-rwxr-xr-x 1 enrike enrike 12696 oct 21 12:06 7
-rwxrwxrwx 1 enrike enrike 1571 oct 20 19:11 7.c
-rwxr-xr-x 1 enrike enrike 12904 oct 21 12:05 expresion
-rwxrwxrwx 1 enrike enrike 2487 oct 20 19:11 expresion.c
-rwxr-xr-x 1 enrike enrike 12976 oct 21 12:06 inversa
-rw-r-xr-- 1 enrike enrike 4300 oct 20 19:11 inversa.c
-rwxr-xr-x 1 enrike enrike 12840 oct 21 12:05 permisos
-rwxrwxrwx 1 enrike enrike 2317 oct 20 19:11 permisos.c

```

Figura 26: Verificación del cambio de permisos en el archivo "inversa.c"

Un funcionamiento 100 % concurrente es imposible en aplicaciones que utilizan creación de códigos por sustitución de código y por copia exacta de código.

Por ejemplo, en esta aplicación, se requiere que se ingrese por medio del teclado muchas entradas, que se manejarán como variables dentro de la ejecución de cada proceso.

Si intentáramos que la aplicación sea concurrente, las entradas exclusivas y correspondientes a cada proceso se mezclarían y los procesos no sabrían identificar cual es su entrada correcta, por lo que arrojarían múltiples errores.

Para evitar esto, colocamos la llamada al sistema **wait()** por cada proceso que este realizando nuestra aplicación, para darle tiempo al usuario de que ingrese la entrada correcta, así como al mismo proceso para que realice las operaciones que tenga que realizar, y no existan colisiones entre las variables de éstos.



```

ARCHIVO SUMA ESCRITO
/home/enrike/Escritorio/P4-S0/8/Resultados/suma.txt
-----

Soy el hijo 2 ejecutando: /home/enrike/Escritorio/P4-S0/8/resta
MATRIZ 1
9.000  0.000  9.000  6.000  8.000  4.000  9.000  8.000  2.000  3.000
8.000  10.000  5.000  9.000  6.000  3.000  8.000  4.000  9.000  4.000
4.000  9.000  10.000  4.000  0.000  9.000  7.000  3.000  2.000  9.000
9.000  0.000  7.000  5.000  5.000  4.000  8.000  3.000  10.000  8.000
4.000  5.000  7.000  9.000  1.000  0.000  2.000  9.000  4.000  9.000
0.000  9.000  6.000  8.000  0.000  4.000  4.000  7.000  7.000  7.000
5.000  5.000  5.000  1.000  9.000  8.000  3.000  6.000  1.000  0.000
3.000  5.000  5.000  8.000  2.000  6.000  8.000  2.000  2.000  10.000
0.000  0.000  6.000  4.000  9.000  7.000  8.000  2.000  3.000  4.000
7.000  7.000  8.000  0.000  6.000  6.000  8.000  8.000  10.000  9.000

MATRIZ 2
8.000  0.000  2.000  1.000  8.000  2.000  7.000  3.000  4.000  8.000
2.000  2.000  6.000  9.000  5.000  4.000  5.000  2.000  5.000  6.000
7.000  10.000  0.000  4.000  10.000  7.000  8.000  8.000  4.000  7.000
4.000  10.000  7.000  4.000  0.000  2.000  4.000  6.000  5.000  6.000
3.000  5.000  7.000  7.000  3.000  1.000  10.000  6.000  3.000  4.000
0.000  10.000  1.000  0.000  1.000  1.000  7.000  9.000  7.000  9.000
3.000  9.000  9.000  8.000  1.000  9.000  8.000  5.000  4.000  2.000
10.000  5.000  6.000  6.000  2.000  7.000  7.000  10.000  3.000  10.000
1.000  1.000  8.000  2.000  1.000  9.000  3.000  7.000  6.000  8.000
5.000  9.000  5.000  1.000  5.000  6.000  0.000  2.000  9.000  2.000

ARCHIVO RESTA ESCRITO
/home/enrike/Escritorio/P4-S0/8/Resultados/resta.txt

```

Figura 29: Proceso Hijo 2: Ejecuta la resta y genera archivo de resultados

```

Soy el hijo 3 ejecutando: /home/enrike/Escritorio/P4-S0/8/multiplicacion
MATRIZ 1
9.000  0.000  9.000  6.000  8.000  4.000  9.000  8.000  2.000  3.000
8.000  10.000  5.000  9.000  6.000  3.000  8.000  4.000  9.000  4.000
4.000  9.000  10.000  4.000  0.000  9.000  7.000  3.000  2.000  9.000
9.000  0.000  7.000  5.000  5.000  4.000  8.000  3.000  10.000  8.000
4.000  5.000  7.000  9.000  1.000  0.000  2.000  9.000  4.000  9.000
0.000  9.000  6.000  8.000  0.000  4.000  4.000  7.000  7.000  7.000
5.000  5.000  5.000  1.000  9.000  8.000  3.000  6.000  1.000  0.000
3.000  5.000  5.000  8.000  2.000  6.000  8.000  2.000  2.000  10.000
0.000  0.000  6.000  4.000  9.000  7.000  8.000  2.000  3.000  4.000
7.000  7.000  8.000  0.000  6.000  6.000  8.000  8.000  10.000  9.000

MATRIZ 2
8.000  0.000  2.000  1.000  8.000  2.000  7.000  3.000  4.000  8.000
2.000  2.000  6.000  9.000  5.000  4.000  5.000  2.000  5.000  6.000
7.000  10.000  0.000  4.000  10.000  7.000  8.000  8.000  4.000  7.000
4.000  10.000  7.000  4.000  0.000  2.000  4.000  6.000  5.000  6.000
3.000  5.000  7.000  7.000  3.000  1.000  10.000  6.000  3.000  4.000
0.000  10.000  1.000  0.000  1.000  1.000  7.000  9.000  7.000  9.000
3.000  9.000  9.000  8.000  1.000  9.000  8.000  5.000  4.000  2.000
10.000  5.000  6.000  6.000  2.000  7.000  7.000  10.000  3.000  10.000
1.000  1.000  8.000  2.000  1.000  9.000  3.000  7.000  6.000  8.000
5.000  9.000  5.000  1.000  5.000  6.000  0.000  2.000  9.000  2.000

ARCHIVO MULTIPLICACION ESCRITO
/home/enrike/Escritorio/P4-S0/8/Resultados/multiplicacion.txt
-----

Soy el hijo 4 ejecutando: /home/enrike/Escritorio/P4-S0/8/transpuesta

```

Figura 30: Proceso Hijo 3: Ejecuta la multiplicación y genera archivo de resultados

```

MATRIZ 1
9.000 0.000 9.000 6.000 8.000 4.000 9.000 8.000 2.000 3.000
8.000 10.000 5.000 9.000 6.000 3.000 8.000 4.000 9.000 4.000
4.000 9.000 10.000 4.000 0.000 9.000 7.000 3.000 2.000 9.000
9.000 0.000 7.000 5.000 5.000 4.000 8.000 3.000 10.000 8.000
4.000 5.000 7.000 9.000 1.000 0.000 2.000 9.000 4.000 9.000
0.000 9.000 6.000 8.000 0.000 4.000 4.000 7.000 7.000 7.000
5.000 5.000 5.000 1.000 9.000 8.000 3.000 6.000 1.000 0.000
3.000 5.000 5.000 8.000 2.000 6.000 8.000 2.000 2.000 10.000
0.000 0.000 6.000 4.000 9.000 7.000 8.000 2.000 3.000 4.000
7.000 7.000 8.000 0.000 6.000 6.000 8.000 8.000 10.000 9.000

MATRIZ 2
8.000 0.000 2.000 1.000 8.000 2.000 7.000 3.000 4.000 8.000
2.000 2.000 6.000 9.000 5.000 4.000 5.000 2.000 5.000 6.000
7.000 10.000 0.000 4.000 10.000 7.000 8.000 8.000 4.000 7.000
4.000 10.000 7.000 4.000 0.000 2.000 4.000 6.000 5.000 6.000
3.000 5.000 7.000 7.000 3.000 1.000 10.000 6.000 3.000 4.000
0.000 10.000 1.000 0.000 1.000 1.000 7.000 9.000 7.000 9.000
3.000 9.000 9.000 8.000 1.000 9.000 8.000 5.000 4.000 2.000
10.000 5.000 6.000 6.000 2.000 7.000 7.000 10.000 3.000 10.000
1.000 1.000 8.000 2.000 1.000 9.000 3.000 7.000 6.000 8.000
5.000 9.000 5.000 1.000 5.000 6.000 0.000 2.000 9.000 2.000

ARCHIVOS TRANSPUESTAS ESCRITO
/home/enrike/Escritorio/P4-S0/8/Resultados/tran1.txt
/home/enrike/Escritorio/P4-S0/8/Resultados/tran2.txt
-----

Soy el hijo 4 ejecutando: /home/enrike/Escritorio/P4-S0/8/inversa
MATRIZ 1
7.000 6.000 7.000 6.000 6.000 3.000 6.000 10.000 1.000 5.000

```

Figura 31: Proceso Hijo 4: Ejecuta las transpuestas y genera dos archivos de resultados

```

5.000 8.000 9.000 7.000 7.000 2.000 5.000 6.000 3.000 8.000
4.000 9.000 5.000 8.000 8.000 8.000 6.000 1.000 7.000 3.000
4.000 10.000 9.000 5.000 8.000 5.000 7.000 8.000 10.000 9.000
6.000 4.000 4.000 2.000 9.000 0.000 4.000 1.000 4.000 7.000
7.000 8.000 4.000 1.000 4.000 10.000 8.000 8.000 9.000 2.000

MATRIZ 2
1.000 2.000 10.000 8.000 6.000 8.000 3.000 2.000 3.000 0.000
9.000 7.000 2.000 0.000 9.000 10.000 9.000 1.000 0.000 3.000
6.000 8.000 9.000 8.000 7.000 2.000 7.000 2.000 0.000 5.000
5.000 10.000 6.000 2.000 7.000 10.000 10.000 8.000 1.000 1.000
8.000 8.000 8.000 9.000 8.000 5.000 6.000 6.000 4.000 6.000
7.000 8.000 1.000 6.000 6.000 7.000 8.000 0.000 9.000 6.000
4.000 1.000 5.000 10.000 4.000 2.000 9.000 1.000 8.000 8.000
2.000 4.000 3.000 9.000 2.000 9.000 3.000 8.000 2.000 5.000
1.000 10.000 2.000 1.000 5.000 8.000 8.000 0.000 7.000 6.000
7.000 0.000 6.000 10.000 8.000 10.000 10.000 6.000 0.000 8.000

ARCHIVOS INVERSAS ESCRITO
/home/enrike/Escritorio/P4-S0/8/Resultados/inv_1.txt
/home/enrike/Escritorio/P4-S0/8/Resultados/inv_2.txt
-----

Soy el padre ejecutando: /home/enrike/Escritorio/P4-S0/8/leerArchivos
-----
----- RESULTADOS -----
-----

SUMA
17.000 0.000 11.000 7.000 16.000 6.000 16.000 11.000 6.000 11.000
10.000 12.000 11.000 18.000 11.000 7.000 13.000 6.000 14.000 10.000
11.000 19.000 10.000 8.000 10.000 16.000 15.000 11.000 6.000 16.000
13.000 10.000 14.000 9.000 5.000 6.000 12.000 9.000 15.000 14.000

```

Figura 32: Proceso Hijo 5: Ejecuta las inversas y genera dos archivos de resultados

```

1.000 1.000 14.000 6.000 10.000 16.000 11.000 9.000 9.000 12.000
12.000 16.000 13.000 1.000 11.000 12.000 8.000 10.000 19.000 11.000

RESTA
1.000 0.000 7.000 5.000 0.000 2.000 2.000 5.000 -2.000 -5.000
6.000 8.000 -1.000 0.000 1.000 -1.000 3.000 2.000 4.000 -2.000
-3.000 -1.000 10.000 0.000 -10.000 2.000 -1.000 -5.000 -2.000 2.000
5.000 -10.000 0.000 1.000 5.000 2.000 4.000 -3.000 5.000 2.000
1.000 0.000 0.000 2.000 -2.000 -1.000 -8.000 3.000 1.000 5.000
0.000 -1.000 5.000 8.000 -1.000 3.000 -3.000 -2.000 0.000 -2.000
2.000 -4.000 -4.000 -7.000 8.000 -1.000 -5.000 1.000 -3.000 -2.000
-7.000 0.000 -1.000 2.000 0.000 -1.000 1.000 -8.000 -1.000 0.000
-1.000 -1.000 -2.000 2.000 8.000 -2.000 5.000 -5.000 -3.000 -4.000
2.000 -2.000 3.000 -1.000 1.000 0.000 8.000 6.000 1.000 7.000

MULTIPLICAR
307.000 380.000 280.000 252.000 232.000 278.000 401.000 364.000 253.000 359.000
266.000 357.000 372.000 306.000 230.000 323.000 382.000 352.000 320.000 400.000
234.000 409.000 241.000 228.000 246.000 287.000 315.000 312.000 314.000 339.000
260.000 354.000 302.000 202.000 225.000 317.000 332.000 335.000 305.000 349.000
275.000 323.000 257.000 207.000 199.000 267.000 246.000 284.000 257.000 313.000
216.000 339.000 283.000 232.000 169.000 288.000 255.000 303.000 279.000 328.000
186.000 253.000 189.000 199.000 166.000 162.000 319.000 279.000 189.000 293.000
203.000 384.000 262.000 204.000 175.000 249.000 264.000 267.000 285.000 271.000
152.000 336.000 226.000 189.000 129.000 203.000 290.000 278.000 212.000 233.000
303.000 387.000 349.000 285.000 274.000 382.000 400.000 397.000 352.000 426.000

TRANSPUESTA MATRIZ 1
9.000 8.000 4.000 9.000 4.000 0.000 5.000 3.000 0.000 7.000
0.000 10.000 9.000 0.000 5.000 9.000 5.000 5.000 0.000 7.000
9.000 5.000 10.000 7.000 7.000 6.000 5.000 5.000 6.000 8.000
6.000 9.000 4.000 5.000 9.000 8.000 1.000 8.000 4.000 0.000

```

Figura 33: Proceso Sub-Padre (6): Lee archivos de resultados y los despliega en la consola

```

TRANSPUESTA MATRIZ 2
8.000 2.000 7.000 4.000 3.000 0.000 3.000 10.000 1.000 5.000
0.000 2.000 10.000 10.000 5.000 10.000 9.000 5.000 1.000 9.000
2.000 6.000 0.000 7.000 7.000 1.000 9.000 6.000 8.000 5.000
1.000 9.000 4.000 4.000 7.000 0.000 8.000 6.000 2.000 1.000
8.000 5.000 10.000 0.000 3.000 1.000 1.000 2.000 1.000 5.000
2.000 4.000 7.000 2.000 1.000 1.000 9.000 7.000 9.000 6.000
7.000 5.000 8.000 4.000 10.000 7.000 8.000 7.000 3.000 0.000
3.000 2.000 8.000 6.000 6.000 9.000 5.000 10.000 7.000 2.000
4.000 5.000 4.000 5.000 3.000 7.000 4.000 3.000 6.000 9.000
8.000 6.000 7.000 6.000 4.000 9.000 2.000 10.000 8.000 2.000

INVERSA MATRIZ 1
0.162 0.115 -0.101 -0.097 0.114 -0.266 -0.122 0.049 0.161 0.032
-0.446 -0.145 0.130 0.241 -0.094 0.742 0.125 -0.440 -0.164 0.158
0.181 0.088 -0.146 -0.145 -0.035 -0.191 -0.101 0.211 0.131 -0.064
0.285 0.106 -0.049 -0.090 0.116 -0.462 0.032 0.232 -0.014 -0.178
0.018 -0.082 0.004 0.051 -0.082 0.013 0.105 -0.042 0.094 -0.038
-0.157 -0.051 0.002 -0.056 0.015 0.245 -0.038 -0.118 -0.031 0.201
-0.034 -0.038 0.149 0.088 -0.126 0.138 0.136 -0.085 -0.134 -0.060
0.116 -0.021 -0.000 0.021 0.012 -0.123 -0.019 0.074 -0.035 -0.027
0.312 0.146 -0.122 -0.124 0.106 -0.687 -0.063 0.419 0.126 -0.169
-0.267 -0.049 0.089 0.005 0.058 0.359 -0.092 -0.126 -0.082 0.154

INVERSA MATRIZ 2
-0.001 0.626 -0.081 -0.138 0.068 -0.426 0.496 0.349 -0.138 -0.509
-0.023 -0.069 0.101 0.022 -0.044 0.090 -0.092 0.015 0.016 -0.005
0.069 0.637 -0.083 -0.151 0.090 -0.628 0.575 0.349 -0.009 -0.552
0.007 -0.406 0.153 0.086 -0.147 0.432 -0.336 -0.155 -0.059 0.308
-0.007 -0.971 0.040 0.186 0.088 0.784 -0.888 -0.691 0.131 0.884
0.055 0.340 -0.052 -0.097 -0.039 -0.239 0.225 0.238 -0.006 -0.244
-0.029 -0.080 0.066 0.120 -0.151 0.081 0.025 -0.056 -0.035 0.062

```

Figura 34: Lee archivos de resultados de matrices transpuestas e inversas



1	17.000	0.000	11.000	7.000	16.000	6.000	16.000	11.000	6.000	11.000
2	10.000	12.000	11.000	18.000	11.000	7.000	13.000	6.000	14.000	10.000
3	11.000	19.000	10.000	8.000	10.000	16.000	15.000	11.000	6.000	16.000
4	13.000	10.000	14.000	9.000	5.000	6.000	12.000	9.000	15.000	14.000
5	7.000	10.000	14.000	16.000	4.000	1.000	12.000	15.000	7.000	13.000
6	0.000	19.000	7.000	8.000	1.000	5.000	11.000	16.000	14.000	16.000
7	8.000	14.000	14.000	9.000	10.000	17.000	11.000	11.000	5.000	2.000
8	13.000	10.000	11.000	14.000	4.000	13.000	15.000	12.000	5.000	20.000
9	1.000	1.000	14.000	6.000	10.000	16.000	11.000	9.000	9.000	12.000
10	12.000	16.000	13.000	1.000	11.000	12.000	8.000	10.000	19.000	11.000

Figura 35: Archivos generados durante la ejecución de cada proceso

### 2.3.2. Sección Windows

✓ **Punto 3:** Programa de creación de un nuevo proceso.

```
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\3>gcc 3.c -o 3
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\3>3
Usar: 3 Nombre_programa_hijo
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\3>
```

✓ **Punto 4:** Programa que contendrá al proceso hijo.

```
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\4>gcc 4.c -o 4
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\4>4
Soy el hijo
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\4>
```

✓ **Punto 5:** Programa que contendrá al proceso hijo, con un nuevo argumento.

```
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\5>gcc 5_hijo.c -o hijo
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\5>gcc 5.c -o 5
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\5>5 hijo
Soy el proceso padre
Soy el hijo
C:\Users\YaKerTaker\Google Drive\5º SEMESTRE\Sistemas-Operativos\Practica4\Windows\5>
```



- ✓ **Punto 8:** Operaciones con matrices de 10x10 con creación de seis procesos por sustitución de código y de forma secuencial. Medición de ambos tiempos.

- **Aplicación secuencial**

```
C:\Users\YaKerTaker\Google Drive\Sto SEMESTRE\Sistemas-Operativos\Practica4\Windows\8>gcc 8_1.c -o o
C:\Users\YaKerTaker\Google Drive\Sto SEMESTRE\Sistemas-Operativos\Practica4\Windows\8>o
MATRIZ 1
1.000 6.000 3.000 10.000 8.000 0.000 3.000 7.000 7.000 0.000
10.000 2.000 3.000 4.000 6.000 0.000 6.000 1.000 4.000 9.000
8.000 6.000 6.000 3.000 10.000 6.000 6.000 8.000 4.000 0.000
6.000 3.000 10.000 9.000 1.000 3.000 3.000 0.000 1.000 10.000
3.000 3.000 0.000 3.000 10.000 8.000 5.000 8.000 10.000 8.000
1.000 5.000 8.000 0.000 6.000 6.000 2.000 3.000 6.000 7.000
6.000 9.000 1.000 7.000 7.000 0.000 6.000 0.000 1.000 6.000
8.000 8.000 7.000 2.000 5.000 4.000 2.000 8.000 6.000 6.000
0.000 7.000 5.000 9.000 4.000 9.000 10.000 0.000 9.000 1.000
4.000 0.000 2.000 3.000 3.000 1.000 1.000 4.000 7.000 3.000

MATRIZ 2
10.000 8.000 4.000 7.000 4.000 9.000 6.000 9.000 1.000 10.000
5.000 4.000 10.000 9.000 8.000 7.000 8.000 7.000 9.000 5.000
5.000 2.000 7.000 5.000 9.000 2.000 6.000 0.000 0.000 1.000
7.000 2.000 10.000 2.000 6.000 5.000 8.000 1.000 7.000 10.000
0.000 3.000 6.000 0.000 6.000 8.000 1.000 0.000 2.000 9.000
7.000 6.000 3.000 10.000 6.000 7.000 3.000 9.000 2.000 8.000
4.000 3.000 1.000 0.000 6.000 7.000 3.000 1.000 10.000 2.000
8.000 3.000 2.000 0.000 5.000 7.000 6.000 0.000 9.000 8.000
8.000 9.000 8.000 3.000 0.000 0.000 1.000 4.000 8.000 5.000
7.000 1.000 3.000 8.000 5.000 0.000 0.000 2.000 0.000 6.000

SUMA
RESTA
MULTIPLICAR
TRANSPUESTA MATRIZ 1
TRANSPUESTA MATRIZ 2
INVERSA MATRIZ 1
INVERSA MATRIZ 2
```

Figura 37: Genera matrices aleatorias, realiza operaciones de forma secuencial y genera archivos de resultados

```
-----
----- RESULTADOS -----
-----
SUMA
11.000 14.000 7.000 17.000 12.000 9.000 9.000 16.000 8.000 10.000
15.000 6.000 13.000 13.000 14.000 7.000 14.000 8.000 13.000 14.000
13.000 8.000 13.000 8.000 19.000 8.000 12.000 8.000 4.000 1.000
13.000 5.000 20.000 11.000 7.000 8.000 11.000 1.000 8.000 20.000
3.000 6.000 6.000 3.000 16.000 16.000 6.000 8.000 12.000 17.000
8.000 11.000 11.000 10.000 12.000 13.000 5.000 12.000 8.000 15.000
10.000 12.000 2.000 7.000 13.000 7.000 9.000 1.000 11.000 8.000
16.000 11.000 9.000 2.000 10.000 11.000 8.000 8.000 15.000 14.000
8.000 16.000 13.000 12.000 4.000 9.000 11.000 4.000 17.000 6.000
11.000 1.000 5.000 11.000 8.000 1.000 1.000 6.000 7.000 9.000

RESTA
-9.000 -2.000 -1.000 3.000 4.000 -9.000 -3.000 -2.000 6.000 -10.000
5.000 -2.000 -7.000 -5.000 -2.000 -7.000 -2.000 -6.000 -5.000 4.000
3.000 4.000 -1.000 -2.000 1.000 4.000 0.000 8.000 4.000 -1.000
-1.000 1.000 0.000 7.000 -5.000 -2.000 -5.000 -1.000 -6.000 0.000
3.000 0.000 -6.000 3.000 4.000 0.000 4.000 8.000 8.000 -1.000
-6.000 -1.000 5.000 -10.000 0.000 -1.000 -1.000 -6.000 4.000 -1.000
2.000 6.000 0.000 7.000 1.000 -7.000 3.000 -1.000 -9.000 4.000
0.000 5.000 5.000 2.000 0.000 -3.000 -4.000 8.000 -3.000 -2.000
-8.000 -2.000 -3.000 6.000 4.000 9.000 9.000 -4.000 1.000 -4.000
-3.000 -1.000 -1.000 -5.000 -2.000 1.000 1.000 2.000 7.000 -3.000

MULTIPLICAR
249.000 175.000 306.000 117.000 240.000 241.000 218.000 92.000 290.000 312.000
280.000 186.000 224.000 195.000 229.000 227.000 160.000 148.000 169.000 301.000
323.000 250.000 296.000 218.000 324.000 361.000 254.000 193.000 279.000 380.000
299.000 147.000 270.000 250.000 284.000 190.000 212.000 138.000 142.000 279.000
342.000 257.000 281.000 228.000 272.000 290.000 173.000 184.000 289.000 401.000
246.000 174.000 241.000 226.000 250.000 185.000 148.000 138.000 165.000 245.000
233.000 154.000 265.000 193.000 255.000 252.000 196.000 146.000 218.000 292.000
359.000 243.000 307.000 273.000 307.000 290.000 247.000 204.000 252.000 358.000
```

Figura 38: Lee resultados de suma, resta y multiplicación

```

305.000 234.000 331.000 231.000 298.000 269.000 228.000 187.000 324.000 309.000
191.000 138.000 155.000 99.000 117.000 121.000 100.000 83.000 135.000 194.000

TRANSPUESTA MATRIZ 1
1.000 10.000 8.000 6.000 3.000 1.000 6.000 8.000 0.000 4.000
6.000 2.000 6.000 3.000 3.000 5.000 9.000 8.000 7.000 0.000
3.000 3.000 6.000 10.000 0.000 8.000 1.000 7.000 5.000 2.000
10.000 4.000 3.000 9.000 3.000 0.000 7.000 2.000 9.000 3.000
8.000 6.000 10.000 1.000 10.000 6.000 7.000 5.000 4.000 3.000
0.000 0.000 6.000 3.000 8.000 6.000 0.000 4.000 9.000 1.000
3.000 6.000 6.000 3.000 5.000 2.000 6.000 2.000 10.000 1.000
7.000 1.000 8.000 0.000 8.000 3.000 0.000 8.000 0.000 4.000
7.000 4.000 4.000 1.000 10.000 6.000 1.000 6.000 9.000 7.000
0.000 9.000 0.000 10.000 8.000 7.000 6.000 6.000 1.000 3.000

TRANSPUESTA MATRIZ 2
10.000 5.000 5.000 7.000 0.000 7.000 4.000 8.000 8.000 7.000
8.000 4.000 2.000 2.000 3.000 6.000 3.000 3.000 9.000 1.000
4.000 10.000 7.000 10.000 6.000 3.000 1.000 2.000 8.000 3.000
7.000 9.000 5.000 2.000 0.000 10.000 0.000 0.000 3.000 8.000
4.000 8.000 9.000 6.000 6.000 6.000 6.000 5.000 0.000 5.000
9.000 7.000 2.000 5.000 8.000 7.000 7.000 7.000 0.000 0.000
6.000 8.000 6.000 8.000 1.000 3.000 3.000 6.000 1.000 0.000
9.000 7.000 0.000 1.000 0.000 9.000 1.000 0.000 4.000 2.000
1.000 9.000 0.000 7.000 2.000 2.000 10.000 9.000 8.000 0.000
10.000 5.000 1.000 10.000 9.000 8.000 2.000 8.000 5.000 6.000

INVERSA MATRIZ 1
-0.143 -0.124 0.090 0.022 -0.040 -0.034 0.143 -0.042 -0.023 0.291
-0.023 -0.067 -0.035 -0.037 -0.027 0.012 0.097 0.080 0.016 0.014
0.067 0.088 0.010 0.008 -0.067 0.075 -0.104 0.006 0.008 -0.093
-0.015 -0.124 0.030 0.083 0.017 -0.051 0.085 -0.070 -0.014 0.141
-0.037 -0.097 0.104 -0.001 -0.014 0.111 0.134 -0.179 -0.071 0.183
-0.175 -0.244 0.102 0.077 0.052 -0.015 0.152 -0.088 -0.007 0.246
0.190 0.383 -0.101 -0.074 0.032 -0.073 -0.309 0.156 0.104 -0.546
0.165 0.182 -0.071 -0.005 0.083 -0.108 -0.237 0.180 0.010 -0.387

```

Figura 39: Lee resultados de matrices transpuestas

```

4.000 8.000 9.000 6.000 6.000 6.000 6.000 5.000 0.000 5.000
9.000 7.000 2.000 5.000 8.000 7.000 7.000 7.000 0.000 0.000
6.000 8.000 6.000 8.000 1.000 3.000 3.000 6.000 1.000 0.000
9.000 7.000 0.000 1.000 0.000 9.000 1.000 0.000 4.000 2.000
1.000 9.000 0.000 7.000 2.000 2.000 10.000 9.000 8.000 0.000
10.000 5.000 1.000 10.000 9.000 8.000 2.000 8.000 5.000 6.000

INVERSA MATRIZ 1
-0.143 -0.124 0.090 0.022 -0.040 -0.034 0.143 -0.042 -0.023 0.291
-0.023 -0.067 -0.035 -0.037 -0.027 0.012 0.097 0.080 0.016 0.014
0.067 0.088 0.010 0.008 -0.067 0.075 -0.104 0.006 0.008 -0.093
-0.015 -0.124 0.030 0.083 0.017 -0.051 0.085 -0.070 -0.014 0.141
-0.037 -0.097 0.104 -0.001 -0.014 0.111 0.134 -0.179 -0.071 0.183
-0.175 -0.244 0.102 0.077 0.052 -0.015 0.152 -0.088 -0.007 0.246
0.190 0.383 -0.101 -0.074 0.032 -0.073 -0.309 0.156 0.104 -0.546
0.165 0.182 -0.071 -0.005 0.083 -0.108 -0.237 0.180 0.010 -0.387
-0.029 -0.022 -0.034 -0.056 -0.046 0.047 0.036 -0.008 0.033 0.197
0.041 0.088 -0.087 0.019 0.065 -0.002 -0.057 0.056 -0.012 -0.160

INVERSA MATRIZ 2
0.269 -0.012 -0.056 0.049 -0.051 -0.325 0.233 -0.203 -0.033 0.219
-0.181 -0.017 0.144 -0.190 0.063 0.234 -0.253 0.291 0.123 -0.204
0.197 0.089 -0.055 0.024 0.045 -0.314 0.122 -0.203 -0.003 0.150
0.027 0.314 -0.008 -0.408 0.110 -0.228 -0.304 0.352 0.014 0.135
-0.235 -0.292 0.114 0.293 -0.080 0.451 0.168 -0.175 -0.004 -0.172
0.366 0.268 -0.102 -0.279 0.132 -0.578 0.062 -0.002 -0.067 0.260
-0.152 0.016 0.100 -0.029 -0.047 0.215 -0.250 0.260 0.004 -0.211
-0.060 -0.291 -0.047 0.473 -0.174 0.356 0.329 -0.443 -0.038 -0.140
-0.096 0.034 -0.038 -0.005 -0.017 0.064 -0.005 0.065 0.025 -0.019
-0.201 -0.107 0.001 0.105 0.005 0.307 -0.136 0.114 0.017 -0.121

Tiempo ejecucion: 8.4970 s
C:\Users\YaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8>

```

Figura 40: Lee resultados de matrices inversas. Tiempo de ejecución

```

Multiplicacion de Matrices, ID del Proceso (2524)
4.000 0.000 2.000 4.000 1.000 3.000 2.000 6.000 1.000 6.000
3.000 6.000 5.000 10.000 9.000 10.000 5.000 5.000 1.000 1.000
10.000 10.000 9.000 5.000 9.000 5.000 2.000 7.000 6.000 10.000
4.000 10.000 5.000 7.000 4.000 3.000 7.000 8.000 10.000 7.000
1.000 8.000 0.000 7.000 6.000 4.000 4.000 9.000 6.000 3.000
6.000 7.000 1.000 3.000 4.000 10.000 5.000 7.000 7.000 4.000
10.000 7.000 6.000 9.000 6.000 8.000 6.000 9.000 4.000 8.000
9.000 10.000 4.000 6.000 9.000 4.000 10.000 6.000 0.000 4.000
6.000 0.000 3.000 6.000 5.000 8.000 10.000 6.000 7.000 7.000
8.000 2.000 2.000 5.000 1.000 5.000 7.000 3.000 1.000 3.000

6.000 6.000 9.000 9.000 9.000 9.000 10.000 1.000 5.000 6.000
4.000 1.000 5.000 9.000 0.000 0.000 4.000 10.000 2.000 5.000
7.000 10.000 4.000 6.000 6.000 10.000 1.000 10.000 1.000 0.000
8.000 3.000 8.000 10.000 3.000 9.000 9.000 8.000 7.000 4.000
0.000 0.000 3.000 10.000 7.000 3.000 1.000 2.000 10.000 7.000
4.000 5.000 6.000 2.000 9.000 5.000 7.000 2.000 4.000 0.000
0.000 9.000 7.000 9.000 6.000 2.000 3.000 7.000 8.000 1.000
7.000 7.000 6.000 3.000 3.000 8.000 6.000 4.000 4.000 2.000
6.000 2.000 5.000 5.000 8.000 7.000 0.000 3.000 7.000 2.000
0.000 5.000 3.000 6.000 0.000 9.000 6.000 4.000 5.000 5.000

ARCHIVO MULTIPLICACION ESCRITO
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8\Resultados\multiplicacion.txt

Transpuestas de Matrices, ID del Proceso (11320)
4.000 0.000 2.000 4.000 1.000 3.000 2.000 6.000 1.000 6.000
3.000 6.000 5.000 10.000 9.000 10.000 5.000 5.000 1.000 1.000

```

Figura 44: Proceso Hijo 3 y 4: Ejecuta la multiplicación, transpuestas y genera archivos de resultados

```

6.000 6.000 9.000 9.000 9.000 9.000 10.000 1.000 5.000 6.000
4.000 1.000 5.000 9.000 0.000 0.000 4.000 10.000 2.000 5.000
7.000 10.000 4.000 6.000 6.000 10.000 1.000 10.000 1.000 0.000
8.000 3.000 8.000 10.000 3.000 9.000 9.000 8.000 7.000 4.000
0.000 0.000 3.000 10.000 7.000 3.000 1.000 2.000 10.000 7.000
4.000 5.000 6.000 2.000 9.000 5.000 7.000 2.000 4.000 0.000
0.000 9.000 7.000 9.000 6.000 2.000 3.000 7.000 8.000 1.000
7.000 7.000 6.000 3.000 3.000 8.000 6.000 4.000 4.000 2.000
6.000 2.000 5.000 5.000 8.000 7.000 0.000 3.000 7.000 2.000
0.000 5.000 3.000 6.000 0.000 9.000 6.000 4.000 5.000 5.000

ARCHIVOS TRANSPUESTAS ESCRITO
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8\Resultados\transpuesta1.txt
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8\Resultados\transpuesta2.txt

Inversas de Matrices, ID del Proceso (10544)
MATRIZ 1
4.000 6.000 0.000 4.000 2.000 7.000 4.000 8.000 1.000 0.000
3.000 4.000 2.000 0.000 6.000 7.000 1.000 6.000 6.000 0.000
3.000 6.000 6.000 1.000 5.000 10.000 10.000 3.000 9.000 0.000
10.000 5.000 5.000 9.000 5.000 7.000 1.000 2.000 1.000 5.000
10.000 9.000 10.000 5.000 9.000 4.000 5.000 8.000 9.000 3.000
5.000 6.000 2.000 7.000 7.000 6.000 6.000 5.000 10.000 3.000
4.000 9.000 10.000 9.000 5.000 6.000 7.000 10.000 4.000 10.000
3.000 2.000 7.000 9.000 8.000 3.000 10.000 5.000 7.000 6.000
1.000 9.000 8.000 0.000 0.000 6.000 7.000 3.000 6.000 7.000
4.000 9.000 4.000 6.000 9.000 3.000 6.000 8.000 3.000 0.000

```

Figura 45: Proceso Hijo 5: Ejecuta las inversas y genera dos archivos de resultados

```

6.000 5.000 0.000 2.000 3.000 1.000 6.000 7.000 5.000 10.000
8.000 4.000 10.000 8.000 6.000 4.000 7.000 7.000 7.000 5.000
8.000 6.000 2.000 5.000 2.000 0.000 5.000 4.000 1.000 7.000
5.000 0.000 7.000 1.000 3.000 2.000 1.000 2.000 3.000 5.000

ARCHIVOS INVERSAS ESCRITO
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8\Resultados\inversa1.txt
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8\Resultados\inversa2.txt

Impresion de las Matrices, ID del Proceso (11180)
----- RESULTADOS -----
SUMA
10.000 6.000 11.000 13.000 10.000 12.000 12.000 7.000 6.000 12.000
7.000 7.000 10.000 19.000 9.000 10.000 9.000 15.000 3.000 6.000
17.000 20.000 13.000 11.000 15.000 15.000 3.000 17.000 7.000 10.000
12.000 13.000 13.000 17.000 7.000 12.000 16.000 16.000 17.000 11.000
1.000 8.000 3.000 17.000 13.000 7.000 5.000 11.000 16.000 10.000
10.000 12.000 7.000 5.000 13.000 15.000 12.000 9.000 11.000 4.000
10.000 16.000 13.000 18.000 12.000 10.000 9.000 16.000 12.000 9.000
16.000 17.000 10.000 9.000 12.000 12.000 16.000 10.000 4.000 6.000
12.000 2.000 8.000 11.000 13.000 15.000 10.000 9.000 14.000 9.000
8.000 7.000 5.000 11.000 1.000 14.000 13.000 7.000 6.000 8.000

RESTA
-2.000 -6.000 -7.000 -5.000 -8.000 -6.000 -8.000 5.000 -4.000 0.000
-1.000 5.000 0.000 1.000 9.000 10.000 1.000 -5.000 -1.000 -4.000
3.000 0.000 5.000 -1.000 3.000 -5.000 1.000 -3.000 5.000 10.000
-4.000 7.000 -3.000 -3.000 1.000 -6.000 -2.000 0.000 3.000 3.000

```

Figura 46: Proceso Hijo 6: Lee archivos de resultados y los despliega en la consola

```

RESTA
-2.000 -6.000 -7.000 -5.000 -8.000 -6.000 -8.000 5.000 -4.000 0.000
-1.000 5.000 0.000 1.000 9.000 10.000 1.000 -5.000 -1.000 -4.000
3.000 0.000 5.000 -1.000 3.000 -5.000 1.000 -3.000 5.000 10.000
-4.000 7.000 -3.000 -3.000 1.000 -6.000 -2.000 0.000 3.000 3.000
1.000 8.000 -3.000 -3.000 -1.000 1.000 3.000 7.000 -4.000 -4.000
2.000 2.000 -5.000 1.000 -5.000 5.000 -2.000 5.000 3.000 4.000
10.000 -2.000 -1.000 0.000 0.000 6.000 3.000 2.000 -4.000 7.000
2.000 3.000 -2.000 3.000 6.000 -4.000 4.000 2.000 -4.000 2.000
0.000 -2.000 -2.000 1.000 -3.000 1.000 10.000 3.000 0.000 5.000
8.000 -3.000 -1.000 -1.000 1.000 -4.000 1.000 -1.000 -4.000 -2.000

MULTIPLICAR
130.000 163.000 170.000 181.000 132.000 223.000 178.000 129.000 149.000 93.000
238.000 241.000 317.000 392.000 293.000 310.000 279.000 293.000 304.000 173.000
308.000 329.000 389.000 513.000 348.000 469.000 346.000 368.000 360.000 271.000
283.000 294.000 360.000 451.000 288.000 387.000 284.000 363.000 339.000 208.000
209.000 181.000 268.000 330.000 207.000 259.000 223.000 251.000 271.000 165.000
226.000 240.000 313.000 338.000 294.000 304.000 271.000 238.000 283.000 164.000
321.000 359.000 427.000 504.000 362.000 473.000 397.000 362.000 377.000 245.000
228.000 294.000 364.000 485.000 300.000 326.000 315.000 333.000 341.000 233.000
221.000 305.000 339.000 383.000 331.000 373.000 286.000 253.000 345.000 166.000
157.000 211.000 244.000 267.000 210.000 237.000 228.000 176.000 201.000 115.000

TRANSPUESTA MATRIZ 1
4.000 3.000 10.000 4.000 1.000 6.000 10.000 9.000 6.000 8.000
0.000 6.000 10.000 10.000 8.000 7.000 7.000 10.000 0.000 2.000
2.000 5.000 9.000 5.000 0.000 1.000 6.000 4.000 3.000 2.000
4.000 10.000 5.000 7.000 7.000 3.000 9.000 6.000 6.000 5.000

```

Figura 47: Lee archivos de resultados de resta, multiplicación y transpuestas de matrices

```

0.000 10.000 6.000 9.000 1.000 2.000 3.000 4.000 10.000 4.000
8.000 8.000 10.000 7.000 1.000 6.000 2.000 9.000 6.000 6.000
5.000 4.000 7.000 5.000 3.000 2.000 5.000 0.000 7.000 10.000
6.000 6.000 7.000 7.000 9.000 9.000 10.000 5.000 4.000 8.000
0.000 3.000 10.000 1.000 10.000 4.000 5.000 10.000 10.000 9.000
7.000 9.000 1.000 9.000 7.000 0.000 8.000 4.000 0.000 0.000
5.000 0.000 0.000 9.000 1.000 3.000 8.000 4.000 10.000 9.000

INVERSA MATRIZ 1
0.226 -0.071 -0.369 0.288 -0.375 0.049 0.203 0.155 -0.173 -0.211
0.049 -0.035 -0.026 -0.033 -0.055 0.052 -0.040 0.045 0.067 -0.039
0.780 -0.462 -1.086 0.840 -0.876 0.023 0.170 0.085 -0.350 -0.852
-0.311 0.249 0.504 -0.465 0.542 -0.059 -0.078 -0.521 0.174 0.421
-0.228 0.057 0.321 -0.135 0.125 0.075 0.013 -0.253 0.109 0.096
0.112 -0.130 -0.190 0.167 -0.128 -0.053 -0.003 0.185 -0.038 -0.051
-0.699 0.504 0.850 -0.666 0.751 -0.023 -0.091 -0.905 0.271 0.750
0.603 -0.386 -0.755 0.577 -0.713 0.010 0.090 0.872 -0.226 -0.685
-0.092 0.004 0.088 -0.145 0.210 -0.031 -0.080 0.024 0.027 0.149
-0.671 0.494 0.086 -0.708 0.883 -0.044 -0.217 -0.944 0.258 0.786

INVERSA MATRIZ 2
0.526 0.044 -0.454 0.302 0.063 -0.403 -0.511 -0.207 0.086 0.682
0.132 0.009 -0.084 0.240 -0.034 -0.122 -0.256 -0.097 0.090 0.165
-0.404 -0.006 0.330 -0.375 0.020 0.384 0.676 0.195 -0.103 -0.777
-0.591 -0.016 0.469 -0.313 -0.100 0.452 0.572 0.331 -0.115 -0.749
-0.237 -0.034 0.168 0.057 -0.031 0.031 -0.133 0.091 0.112 -0.006
-0.420 -0.005 0.406 -0.366 -0.016 0.482 0.575 0.260 -0.096 -0.836
0.163 -0.061 -0.255 0.139 0.045 -0.104 -0.129 -0.132 0.001 0.298
-0.128 0.003 0.188 -0.309 -0.040 0.224 0.412 0.084 -0.079 -0.365
1.066 -0.045 -0.819 0.608 0.116 -0.923 -1.077 -0.496 0.132 1.591
0.071 0.088 -0.079 0.114 0.019 -0.073 -0.162 -0.101 -0.020 0.148

Tiempo ejecucion: 8.2770 s
C:\Users\VaKerTaker\Google Drive\5to SEMESTRE\Sistemas-Operativos\Practica4\Windows\8>

```

Figura 48: Lee archivos de resultados de matrices inversas

	inversa1.txt	inversa2.txt	multiplicacion.txt	resta.txt	suma.txt	transpuesta1.txt	transpuesta2.txt
1	0.185	0.029	-0.167	0.069	0.097	-0.092	-0.268
2	-0.055	-0.056	0.004	0.007	-0.026	0.073	0.020
3	-0.156	-0.096	0.227	-0.042	0.054	-0.011	0.275
4	-0.140	-0.185	0.218	-0.037	-0.021	0.173	0.321
5	-0.031	0.180	-0.132	0.072	-0.069	-0.095	-0.182
6	-0.027	0.055	0.098	0.042	-0.071	-0.009	0.076
7	0.161	-0.002	-0.102	0.015	0.002	-0.101	-0.235
8	0.103	0.036	-0.040	-0.075	0.054	-0.033	0.040
9	-0.072	-0.058	0.080	-0.074	0.048	0.156	0.121
10	0.102	0.168	-0.274	0.074	-0.048	-0.079	-0.254

Figura 49: Archivos generados durante la ejecución de cada proceso

### 3. Observaciones

- ✓ Cuando estamos trabajando con árboles de procesos, es de suma importancia identificar si la creación de nuevos procesos hijos será de forma horizontal o vertical, pues de esto dependerá en que parte del código se colocará el **exit(0)** de cada proceso.
- ✓ En el sistema operativo Linux, para crear n procesos en una misma orientación (horizontal, vertical), podemos hacer uso de la función **fork()** dentro de un ciclo. Para saber en qué proceso estamos, utilizamos el contador del mismo ciclo.
- ✓ Para la realización de las aplicaciones con matrices, en su mayoría utilizamos arreglos bidimensionales dinámicos, para no estar limitados a un tamaño fijo de las matrices.
- ✓ En las funciones de escrituras de archivos, utilizamos la función **sprintf()** para pasar el contenido de las variables en las matrices de tipo double a otra de tipo cadena, y así escribir archivos de texto plano.
- ✓ Para las funciones de escritura y lectura de archivos, utilizamos las llamadas al sistema previamente revisadas en la Práctica 2. Para Linux: **creat()**, **open()**, **write()**, **read()**. Para Windows: **CreateFile()**, **WriteFile()**, **ReadFile()**.
- ✓ Para el manejo de directorios de los archivos, así como las rutas de los ejecutables de los procesos hijos en las aplicaciones de creación de procesos por sustitución de códigos, nosotros definimos éstas dentro de los programas como una variable de tipo cadena. Estos directorios debe existir antes de ejecutar la aplicación, sino habrá errores en la lectura/escritura de archivos (En las aplicaciones de Linux, puedes crear el directorio donde se guardaran los resultados, desde el programa).

- ✓ Para la medición de tiempos, se utilizaron métodos distintos para cada sistema operativo. En el caso de Linux, utilizamos una librería **tiempo.h** creada por nosotros, así como un código objeto **tiempo.c**, que contiene funciones para medir el tiempo en segundos automáticamente. En el caso de Windows, utilizamos una variable de tipo **clock\_t** para medir el tiempo de inicio y de finalización, que posteriormente convertimos a segundos.
- ✓ La ejecución de las matrices inversas en Windows tarda mucho más que en Linux, aproximadamente unos 6 segundos en los que la terminal con la aplicación queda detenida, realizando los cálculos necesarios.
- ✓ Para las aplicaciones de creación de procesos por sustitución de código, fue necesario agregar la llamada al sistema **wait(0)** en Linux y **WaitForSingleObject(pi.hProcess, INFINITE)** para esperar que cada proceso se ejecute y termine, y así evitar que se mezclen sus ejecuciones.
- ✓ Algunas aplicaciones deben compilarse con atributos extras en el comando GCC. La instrucción para compilar éstas aplicaciones se encuentra al inicio de su respectivo código (medición de tiempos en Linux, enviar procesos hijos como parámetros a códigos padre, uso de la función **pow()**, etc.).
- ✓ En Windows solo se maneja la creación de procesos por sustitución de código, por lo que a cada nivel de un árbol de procesos (o creación de nuevos procesos en general) se tendrá que crear un código extra que contenga las instrucciones a ejecutar de los procesos hijos. Esto se puede apreciar mejor en los códigos del punto 7 de la sección de Windows.

## 4. Análisis Crítico

La diferencia de la creación de procesos en los sistemas operativos Windows y Linux, es notable por los métodos que existen en ambos:

- Linux : Sustitución de código, copia exacta de código
- Windows: Sustitución de código

La manera de manejar dichos procesos en cada sistema operativo, cambia en la sintaxis que se usa. En Linux se puede usar **fork()** o **execv()** y en Windows Se usa **CreateProcess()** cabe destacar que siempre deben ser cerrados en windows.

Al ejecutar el programa de las matrices, hicimos una comparativa de los tiempos en los que se realiza teniendo los siguientes resultados: