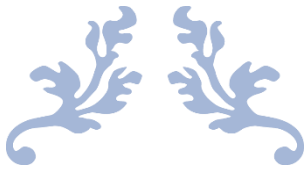


PRÁCTICA 2:
GESTIÓN DE PROCESOS

Julián Blanco González

8 DE DICIEMBRE DE 2023

GRADO EN INGENIERÍA INFORMÁTICA

Grupo Lunes/ Convocatoria: Diciembre

Contenido

1. Memoria Explicativa de la Práctica

1.1. Introducción de la práctica

La práctica consiste en la comunicación de procesos a través de diferentes mecanismos, como tuberías, señales o memoria compartida.

Hay una matriz de 15 filas * 10000 columnas, donde cada posición es un número aleatorio entre 0 y 30000. Para gestionar la práctica, hay 3 niveles jerárquicos:

- **Proceso Padre:** Nivel 1 (1 proceso)
- **Proceso Hijo Nivel 2:** Nivel 2 (3 procesos)
- **Proceso Hijo Nivel 3:** Nivel 3 (5 procesos)

A continuación, voy a hacer un breve resumen de lo que realiza cada proceso:

1.1.1. Proceso Padre

Tareas proceso padre:

- Crear todos los datos que se van a utilizar.
- Crear los 3 procesos de nivel 2 y asignar sus tareas.
- Recuperar de memoria compartida el total de primos de cada hijo de nivel 2 mediante la recepción de la señal SIGUSR1.
- Esperar a que terminen los 3 hijos terminen, notificarle de que pueden terminar mediante el envío de la señal SIGINT y sumar el total de primos de los 3 hijos.
- Crear un fichero con el formato N1_pid.primos, que recoja esta información:
    - Comienzo
    - Procesos que crea
    - Envío y recepción de notificaciones
    - Cómputo final de los números primos encontrados.

1.1.2. Procesos Hijos de Nivel 2

Tareas hijo de Nivel 2:

- Cada uno tratará 5000 números, es decir, el primero de las filas de 0 a 4, el segundo de 5 a 9 y el tercero de 10 a 14.
- Crea 5 procesos hijos de nivel 3 para que traten una de las 5 filas asignadas, es decir, computando el número de primos totales en cada fila.
- Esperará a que terminen sus 5 procesos hijos y recogerán el total de números primos encontrados y lo guardarán en memoria compartida. Se enviará la señal SIGUSR1 para notificar al padre que esta el total de primos en memoria compartida y pueda leerlos.
- Crear un fichero con el formato N2_pid.primos, que recoja esta información:
    - Inicio de ejecución
    - Identificación de que procesos está creando
    - Resultado total enviado por sus hijos

- Para que el padre calcule el total, se enviará la comunicación a través de tubería, y el hijo terminará una vez que el proceso padre lea de la tubería y envíe la señal SIGINT al hijo correspondiente para que termine

1.1.3. Procesos Hijos de Nivel 3

Tareas hijo nivel 3:

- Computará el total de números primos encontrado en la fila que le haya asignado el hijo de nivel 2 (1000 números por fila).
- Enviar el total de números primos encontrados a través de exit(X) (X: número total de primos).
- Crear un fichero con el nombre N3_pid.primos con el siguiente formato

  nv:id_proceso:num_primo

1.2. Solución aportada

Para manejar los procesos, he utilizado la estructura switch – case, donde:

- Case -1: Error al crear el proceso
- Case 0: Código del proceso creado
- Default: Código del proceso creador (padre).

Para poder recuperar el pid de cada hijo de nivel 2 en la función de la señal SIGUSR1 y poder luego enviar a cada hijo la señal SIGINT para terminarlos, he utilizado una estructura:

```
struct infoHijoNivel2
{
    int pid;
    int totalPrimos;
};
```

Al ser un programa de procesos pequeños, para ver la sincronización, he utilizado la función de usleep() justo antes del envío de las señales, para poder ver correctamente en la salida por pantalla la sincronización de los procesos.

1.3. Retos que han surgido

Conseguir poner la instrucción de wait(&estado) en el sitio correcto, para hacer que los hijos (nivel 2 o nivel 3) realicen sus tareas y el programa espere a que terminen

Conseguir hacer la comunicación del padre al hijo con la señal SIGINT para que los hijos de nivel 2 terminen. Lo he hecho dentro del manejador de señal SIGUSR1.

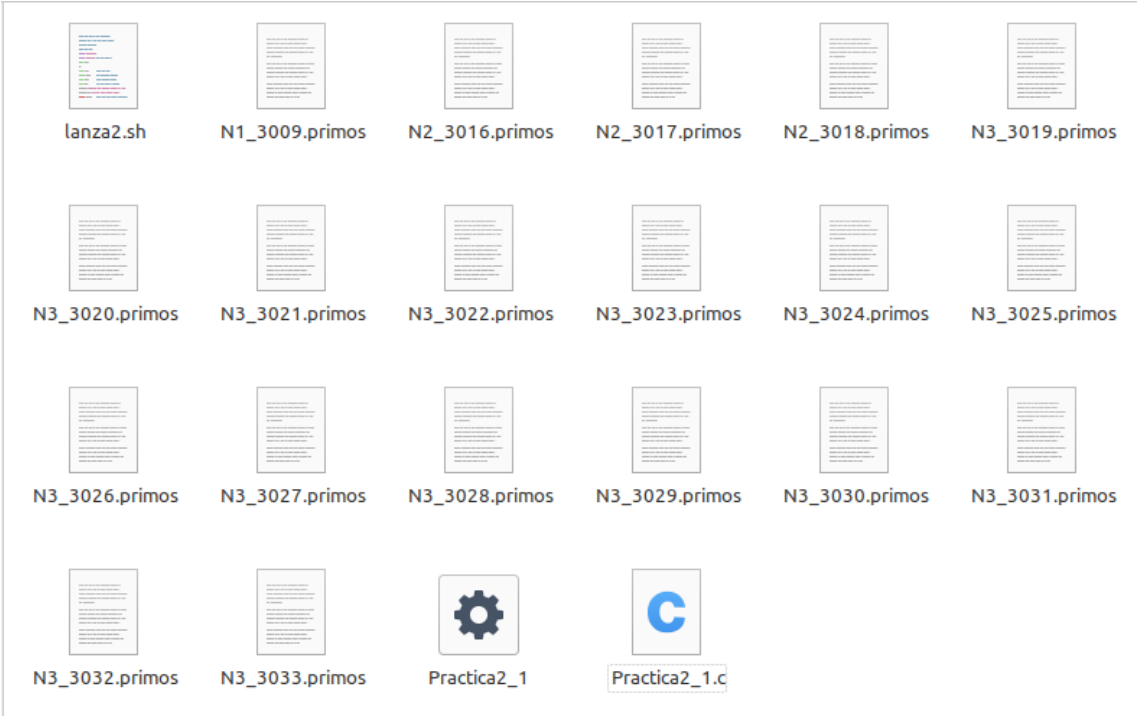
Conseguir hacer la función de devolver el número total de primos de una fila, y para ello, he añadido la librería: #include <stdbool.h>, para poder tratar con valores booleanos.

Conseguir crear la memoria compartida y acceder a ella mediante un puntero, en este caso, mi memoria compartida es una estructura y mi puntero también es una estructura.

2. Manual de Usuario

2.1. Carpeta del programa

Al descargar la carpeta, estarán estos ficheros:



.primos: extensión de los ficheros resultado de realizar la ejecución de la práctica. Dependiendo del número que sigue a la N, se ve el nivel de cada uno.

lanza2.sh: ejecutable shell de la práctica

Practica2_1.c: código de lenguaje c con el contenido de la práctica.

Práctica2_1: compilación del código de c, es decir, el otro ejecutable.

2.2. Como ejecutar la aplicación

Para poder ejecutar hay 2 posibilidades:

2.2.1. Ejecutar el archivo lanza2.sh

Lanza2.sh tiene este contenido:

```
# Limpiar la pantalla
clear

# Compilar el archivo Practica2.c
gcc Practica2_1.c -o Practica2_1

# Verificar si la compilación fue exitosa antes de intentar ejecutar el program
if [ $? -eq 0 ]; then
    # Ejecutar el programa Practica2
    ./Practica2_1
else
    # Imprimir un mensaje de error si la compilación falla
    echo "Error: La compilación ha fallado."
fi
```

Entonces, lo único que hay que hacer en consola es ejecutarlo utilizando ./ o bash:

```
julian@julian-VirtualBox:~/Escritorio/3-Practica2$ ./lanza2.sh
```

2.2.2. Compilar el archivo .c y ejecutarlo

La otra posibilidad, es compilar por consola y ejecutarlo después

Compilar por consola:

```
julian@julian-VirtualBox:~/Escritorio/3-Practica2$ gcc Practica2_1.c -o Practica2_1
```

Ejecución:

```
julian@julian-VirtualBox:~/Escritorio/3-Practica2$ ./Practica2_1
```

3. Manual del Programador

3.1. Función que recoge el número total de primos de una fila de la matriz

```
int calcularNumerosPrimos(int fila, int matriz[filas][columnas])
{
    int contadorPrimos = 0;
    for (int j = 0; j < columnas; j++)
    {
        int num = matriz[fila][j];
        bool esPrimo = true;
        if (num <= 1)
        {
            esPrimo = false;
        }
        else
        {
            for (int i = 2; i * i <= num; i++)
            {
                if (num % i == 0)
                {
                    esPrimo = false;
                    break;
                }
            }
        }
        if (esPrimo)
        {
            contadorPrimos++;
            escribirPrimoEnArchivo(3, getpid(), num);
        }
    }
    return contadorPrimos;
}
```

Le paso por parámetro la matriz y la fila de la matriz que quiero que busque los números primos. Recorro todas sus columnas, recojo el número de la columna en una variable y me creo un bool para ver si es primo o no. Luego, si el número es 1 o 0, es primo y retorno false, y si no, aplico la fórmula para saber si un número es primo o no. Cuando lo sea, incremento la variable contador del inicio de la función y escribo en el archivo de nivel 3. Una vez que recorre todas las columnas, devuelvo el número de primos encontrados (valor de la variable contador).

3.2. Envío de total de primos en una fila a hijo 2

```
    case 0:
        printf("\n\t\tProceso Nieto %d (PID: %d) | Proceso Padre (PID: %d)
        // filaMatriz(k, matriz);
        fopen(nombreArchivo_hijo2, "a");
        fprintf(archivo_hijo2, "Crea el proceso de Nivel 3: %d\n", getpid()
        fclose(archivo_hijo2);
        int primosEnFila = calcularNumerosPrimos(k, matriz);
        printf("\t\tNúmero total de primos en la fila %d: %d\n", k, primosE

        exit(primosEnFila);
        break;
    default:

        break;
    }
}
// esperar a que terminen los 5 hijos de nivel 3
for (int i = 0; i < 5; i++)
{
    wait(&estado);
    if (WIFEXITED(estado) >= 0)
    {
        printf("\tNúmeros primos encontrados: %d \n", WEXITSTATUS(estado));
        totalPrimosAux = totalPrimosAux + WEXITSTATUS(estado);
    }
}
```

Una vez recuperado el total de números primos de la fila que toque, los envío en la llamada exit. Para recuperar este valor, en el hijo de nivel 2, primero espero a que terminen los hijos de nivel 3, y luego, con la llamada WIFEXITED, se ve que el proceso no ha terminado con un exit(0) y con WEXIXSTATUS recupero la parte alta de este valor (8 bits), en este caso, el total de primos encontrados en la fila correspondiente al hijo 3.

3.3. Creación y manipulación de memoria compartida

3.3.1. Variables y creación de memoria compartida

```
// Memoria compartida
key_t llave;                            // Clave para obtener con ftok
long int shmid;                         // ID de memoria a obtener con shmget
struct infoHijoNivel2 *infoHijolNivel2; // Puntero de memoria compartida
```

```
// Crear la memoria compartida
llave = ftok("/bin/cat", 121);
shmid = shmget(llave, 3 * sizeof(struct infoHijoNivel2), 0777 | IPC_CREAT); // R
if (shmid == -1)
{
    printf("Error en la creacion de la memoria compartida.\n");
    exit(0);
}
infoHijolNivel2 = (struct infoHijoNivel2 *)shmat(shmid, NULL, 0);
```

En shmid, reservo el espacio para 3 estructuras (los 3 hijos de nivel 2 con su total de primos).

3.3.2. Inicializar la memoria compartida

```
// Inicializar la memoria compartida
for (int i = 0; i < 3; i++)
{
    infoHijolNivel2[i].pid = 0;
    infoHijolNivel2[i].totalPrimos = 0;
}
```

3.3.3. Escribir en memoria compartida con el puntero

```
infoHijolNivel2[l].pid = getpid();
```

Escribo en el atributo de pid, el pid del hijo de nivel 2.

```
infoHijolNivel2[l].totalPrimos = totalPrimosAux;
```

Una vez que tengo el total de primos de los 5 hijos de nivel 3 sumados, los añado a la memoria compartida de la estructura que le toque.

3.3.4. Acceder desde el padre a la memoria compartida.

```
write(tuberia[1], &infoHijolNivel2[l], sizeof(infoHijolNivel2[l]));
usleep(500);
kill(getppid(), SIGUSR1); // Llamada a SIGURS1 para comunicarse con
```

Para poder acceder desde el padre, se utiliza una tubería y una señal. Una vez añadido el total de número primos de cada hijo de nivel 2 en memoria compartida, escribo en la tubería ese número. Llamo a SIGUSR1 utilizando la sentencia kill con el pid del padre.

```
void manejador_senial_SIGUSR1(int senial)
{
    struct infoHijoNivel2 info; // puntero
    pid_t pid = getpid();

    // Leer la estructura desde la tubería
    read(tuberia[0], &info, sizeof(info));
    printf("------> Proceso PADRE (PID: %d
    // Envía la señal SIGINT al hijo2 (PID
    kill(info.pid, SIGINT);
    usleep(500);
}
```

En la función, me creo un puntero para poder acceder a memoria compartida. Para ello, leo de la tubería con el puntero y recupero la estructura. Una vez que recupero el total de primos en el padre, envío una señal al hijo de nivel 2 para indicarle que puede terminar, mediante la señal SIGINT.

3.3.5. Liberar la memoria compartida

```
// Liberar memomria compartida
shmdt(infoHijolNivel2);
shmctl(shmid, IPC_RMID, 0);
```

3.4. Código para borrar los archivos ".primos" generados anteriormente

```
// Borrar los archivos de la anterior ejecución
system("rm -f N1_*.primos");
system("rm -f N2_*.primos");
system("rm -f N3_*.primos");
```

Llamo a la llamada del sistema "system" y con rm borro los archivos especificados. Con -f, cojo los archivos que tengan el patrón de letras y números que pongo arriba.

4. Juego de Pruebas

La matriz es de 15 x 15 y me he creado 2 funciones auxiliares para hacer este juego de pruebas.

4.1. Función de inicialización de la matriz.

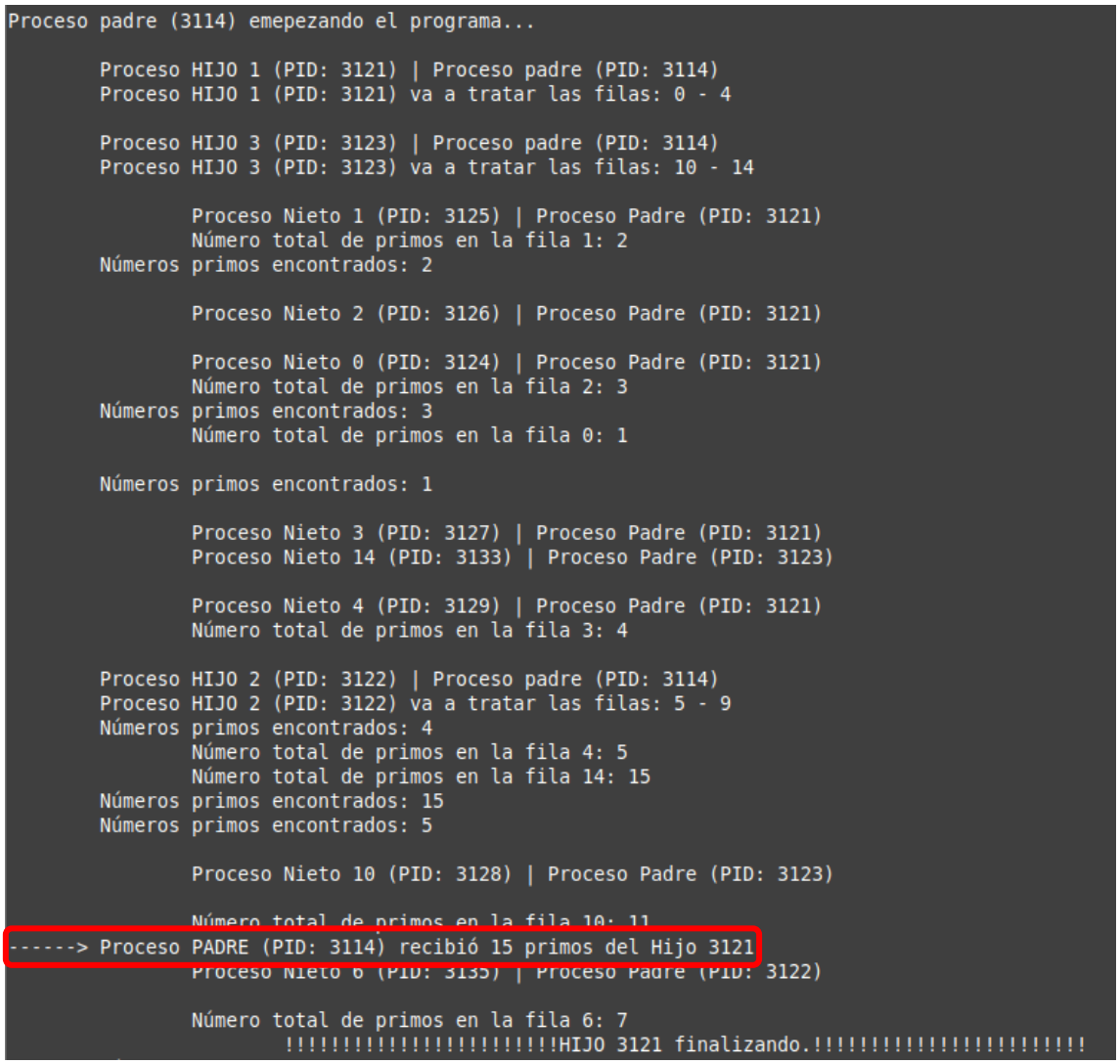
```
// Función para verificar si un número es primo
int esPrimo(int num)
{
    if (num < 2)
    {
        return 0; // 0 y 1 no son primos
    }
    for (int i = 2; i * i <= num; i++)
    {
        if (num % i == 0)
        {
            return 0; // No es primo si es divis
        }
    }
    return 1; // Es primo
}
```

```
void inicializarMatriz2(int matriz[filas][columnas])
{
    for (int i = 0; i < filas; i++)
    {
        int numPrimos = 2; // Comenzamos desde el segundo número primo
        for (int j = 0; j < i + 1; j++)
        {
            while (!esPrimo(numPrimos))
            {
                numPrimos++;
            }
            matriz[i][j] = numPrimos;
            numPrimos++;
        }
        // Rellenar el resto de la fila con ceros
        for (int j = i + 1; j < filas; j++)
        {
            matriz[i][j] = 0;
        }
    }
}
```

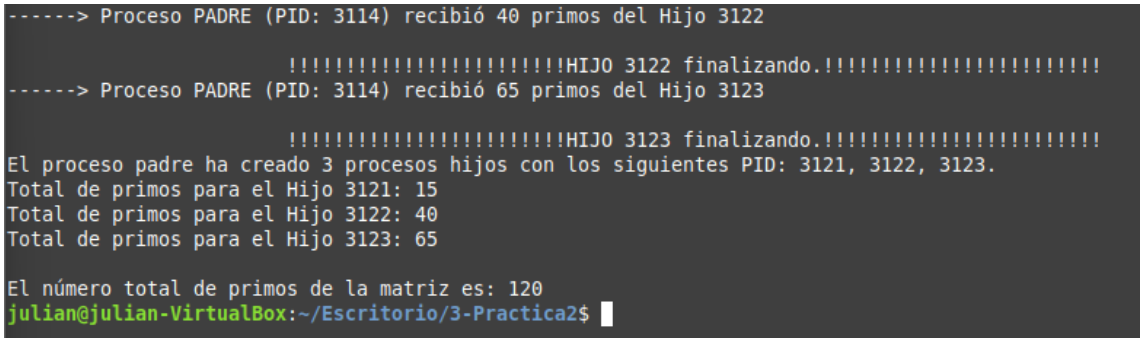
La primera es para indicar si un número es primo o no. Y la segunda, es para ir rellenando la matriz, donde la primera fila tendrá el primer número primo, la segunda, tendrá los dos primeros números primos… El resultado queda así:

```
Matriz-----------------------------------------------
Fila  0: |  2 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  1: |  2 |  3 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  2: |  2 |  3 |  5 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  3: |  2 |  3 |  5 |  7 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  4: |  2 |  3 |  5 |  7 | 11 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  5: |  2 |  3 |  5 |  7 | 11 | 13 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  6: |  2 |  3 |  5 |  7 | 11 | 13 | 17 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  7: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  8: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 |  0 |  0 |  0 |  0 |  0 |  0 |
Fila  9: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 |  0 |  0 |  0 |  0 |  0 |
Fila 10: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 |  0 |  0 |  0 |  0 |
Fila 11: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 |  0 |  0 |  0 |
Fila 12: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 |  0 |  0 |
Fila 13: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 |  0 |
Fila 14: |  2 |  3 |  5 |  7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 | 37 | 41 | 43 | 47 |
```

4.2. Capturas de la ejecución del programa



Se puede ver que los procesos están sincronizados y no ocurren de manera secuencial. Los mensajes con 1 tabulación pertenecen a los hijos de nivel 2, y los que tienen 2 tabulaciones, a los hijos de nivel 3. Abajo del todo, una vez recibido el total de primos de los 5 hijos de nivel 3, el padre, mediante la señal y la tubería, recupera de memoria compartida el total de números, y posteriormente finaliza por la señal SIGINT que envía el padre a ese hijo.



Una vez recuperados en el padre, los 3 totales de primos computados por los hijos de nivel 2, éste los suma, y computa el total de números primos encontrados.

4.3. Captura de los ficheros generados

4.3.1. Fichero Padre

```
COMIENZA EL PROGRAMA EL PADRE (PID: 3114)...
Padre (PID: 3114) crea el hijo de Nivel 2: (PID: 3121)
Padre (PID: 3114) crea el hijo de Nivel 2: (PID: 3123)
Padre (PID: 3114) crea el hijo de Nivel 2: (PID: 3122)
Señal SIGUSR1 del hijo (PID: 3121) al Padre (PID: 3114) para comunicar que puede leer de la tubería
Señal SIGUSR1 del hijo (PID: 3122) al Padre (PID: 3114) para comunicar que puede leer de la tubería
Señal SIGUSR1 del hijo (PID: 3123) al Padre (PID: 3114) para comunicar que puede leer de la tubería
TOTAL DE PRIMOS ENCONTRADOS EN LA MATRIZ: 120
```

4.3.2. Ficheros Hijo Nivel 2

```
Inicio de ejecución del hijo 1 de Nivel 2 (PID: 3121)
Crea el proceso de Nivel 3: 3125
Crea el proceso de Nivel 3: 3126
Crea el proceso de Nivel 3: 3124
Crea el proceso de Nivel 3: 3127
Crea el proceso de Nivel 3: 3129
Total de números primos encontrados: 15
```

4.3.3. Ficheros Hijo Nivel 3

```
3:3128:2
3:3128:3
3:3128:5
3:3128:7
3:3128:11
3:3128:13
3:3128:17
3:3128:19
3:3128:23
3:3128:29
3:3128:31
```

5. Bibliografía

Para hacer la práctica, he utilizado en su mayoría el PDF de la asignatura Gestión de Procesos_2023. También me ha sido de ayuda, los ejercicios proporcionados, para entender como programar la teoría de la clase. Y he consultado en estos sitios webs, algunos conceptos para la práctica

Números primos en C

https://programador-apli.blogspot.com/2012/04/comprobar-si-un-numero-es-primo-en.html

Trabajar con ficheros en C

https://www.infor.uva.es/~belar/Informatica/Curso%202008-2009/Ficheros%20en%20C%202008-2009%202txp.pdf