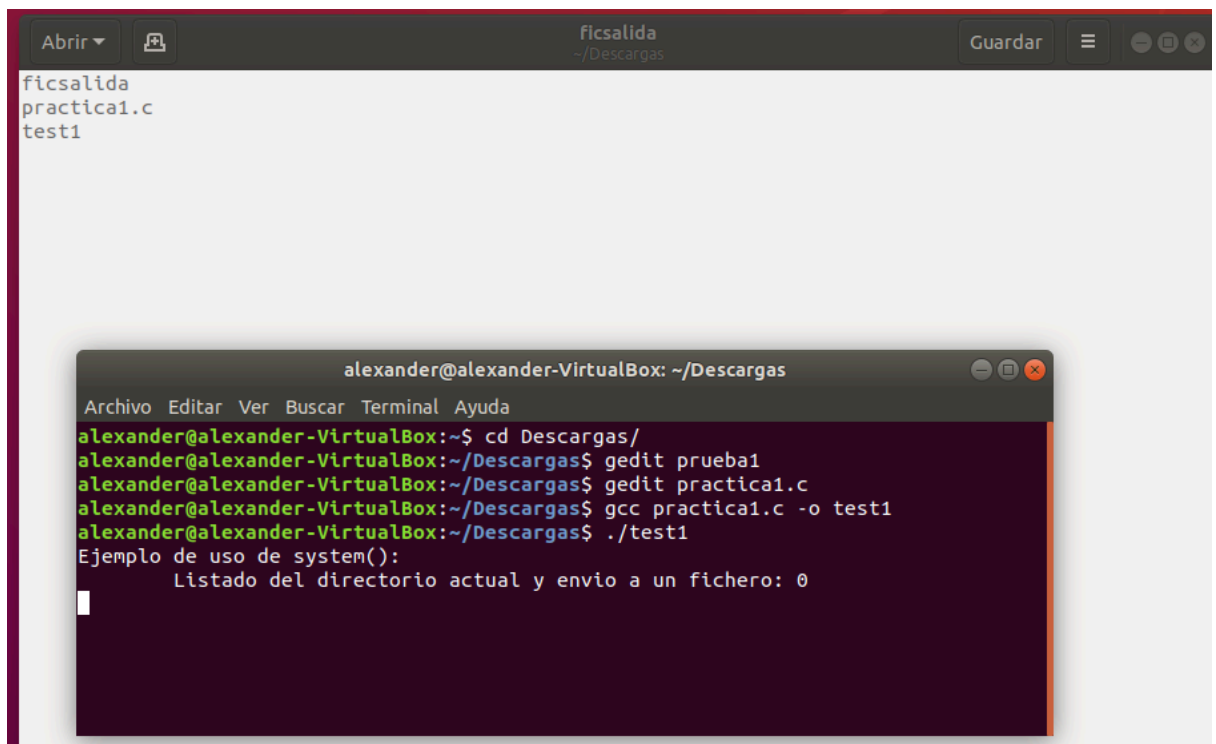


EJERCICIOS REALIZADOS

1. Listar Contenido

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    printf("Ejemplo de uso de system():");
    printf("\n\tListado del directorio actual y envio a un fichero:");
    printf("%d", system("ls > ficsalida"));
    printf("\n\tAbrimos con el gedit el fichero...");
    printf("%d", system("gedit ficsalida"));
    printf("\n\tEste comando es erróneo: %d", system("ged"));
    printf("\nFin de programa....\n");
}
```



The screenshot shows a code editor window titled 'ficsalida' with the following content:

```
ficsalida
practica1.c
test1
```

Below the code editor is a terminal window titled 'alexander@alexander-VirtualBox: ~/Descargas'. The terminal shows the following commands and output:

```
alexander@alexander-VirtualBox:~$ cd Descargas/
alexander@alexander-VirtualBox:~/Descargas$ gedit prueba1
alexander@alexander-VirtualBox:~/Descargas$ gedit practica1.c
alexander@alexander-VirtualBox:~/Descargas$ gcc practica1.c -o test1
alexander@alexander-VirtualBox:~/Descargas$ ./test1
Ejemplo de uso de system():
Listado del directorio actual y envio a un fichero: 0
```

2. Proceso Hijo e Proceso Padre

```
#include <stdio.h>
#include <unistd.h>

void main(void)
{
    pid_t id_pactual, id_padre;

    id_pactual = getpid();
    id_padre = getppid();

    printf("PID de este proceso: %d\n", id_pactual);
    printf("PID del proceso padre: %d\n", id_padre);
}
```

```
alexander@alexander-VirtualBox:~/Descargas$ ./test2
PID de este proceso: 1657
PID del proceso padre: 1438
alexander@alexander-VirtualBox:~/Descargas$
```

3. Otro Hijo y Padre

```
alexander@alexander-VirtualBox:~/Descargas$ ./test2
Soy el proceso padre:
    Mi PID es 2629, El PID de mi padre es: 2328.
    Mi hijo: 2630 terminó.
alexander@alexander-VirtualBox:~/Descargas$ Soy el proceso hijo
    Mi PID es 2630, El PID de mi padre es: 1131.
```

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

void main() {
    pid_t pid, Hijo_pid;
    pid = fork();

    if (pid == -1) // Ha ocurrido un error
    {
        printf("No se ha podido crear el proceso hijo...");
        exit(-1);
    }

    if (pid == 0) // Nos encontramos en Proceso hijo
    {
```

```

        printf("Soy el proceso hijo \n\tMi PID es %d, El PID de mi
padre es: %d.\n",
            getpid(), getppid());
    }
    else // Nos encontramos en Proceso padre
    {
        Hijo_pid = wait(NULL); // espera la finalización del proceso
hijo
        printf("Soy el proceso padre:\n\tMi PID es %d, El PID de mi
padre es: %d.\n\tMi hijo: %d terminó.\n",
            getpid(), getppid(), pid);
    }

    exit(0);
}

```

4. Abuelo, Hijo y Nieto

```

alexander@alexander-VirtualBox:~/Descargas$ ./test3
        Soy el proceso NIETO 2626; Mi padre es = 2625
Soy el proceso HIJO 2625, Mi padre es: 2624.
        Mi hijo: 2626 terminó.
Soy el abuelo: 2624, MI HIJO 2625 terminó.

```

```

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h> // For wait() system call

// ABUELO-HIJO-NIETO
int main() {
    pid_t pid, Hijo_pid, pid2, Hijo2_pid;

    pid = fork(); // Soy el Abuelo, creo al Hijo

    if (pid == -1) { // Ha ocurrido un error
        printf("No se ha podido crear el proceso hijo...\n");
        exit(-1);
    }

    if (pid == 0) { // Nos encontramos en Proceso Hijo

```

```

pid2 = fork(); // Soy el Hijo, creo al Nieto

switch (pid2) {
    case -1: // Error
        printf("No se ha podido crear el proceso hijo en el
HIJO...\n");
        exit(-1);
        break;

    case 0: // Proceso Nieto
        printf("\t\tSoy el proceso NIETO %d; Mi padre es =
%d\n", getpid(), getppid());
        break;

    default: // Proceso Hijo (padre del Nieto)
        Hijo2_pid = wait(NULL); // Espera la terminación del
Nieto
        printf("\tSoy el proceso HIJO %d, Mi padre es: %d.\n",
getpid(), getppid());
        printf("\tMi hijo: %d terminó.\n", Hijo2_pid);
        break;
}
} else { // Nos encontramos en Proceso Abuelo
    Hijo_pid = wait(NULL); // Espera la finalización del proceso
Hijo
    printf("Soy el proceso ABUELO: %d, Mi HIJO: %d
terminó.\n", getpid(), pid);
}

exit(0);
}

```

5. Otro de Padre e Hijo

```

Variable en Proceso Padre: 11alexander@alexander-VirtualBox:~/Descargas$
Valor inicial de la variable: 6
Variable en Proceso Hijo: 1

```

```

#include <stdio.h>
#include <unistd.h>

```

```

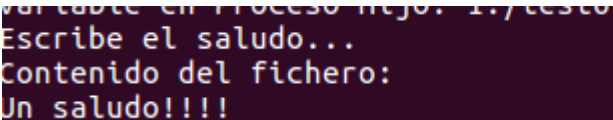
#include <stdlib.h>

void main(){
    pid_t pid;
    int num = 6;
    pid = fork();

    if(pid == -1){
        printf("Error al crear proceso");
        exit(-1);
    }
    else{
        if(pid == 0){
            num -= 5;
            printf("Variable en Proceso Hijo: %d", num);
        } else{
            num += 5;
            printf("Variable en Proceso Padre: %d", num);
        }
    }
}

```

6. Open Write



```

Variable en Proceso Hijo: 1./texto
Escribe el saludo...
Contenido del fichero:
Un saludo!!!!

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(void)
{
    char saludo[] = "Un saludo!!!!\n";
    char archivo[] = "texto.txt";
    char buffer[10];
    int fd, bytesleidos;

    fd = open(archivo, 1);

```

```

    if( fd == -1 )
    {
        printf("ERROR AL CREAR EL FICHER...\n");
        exit(-1);
    }

    printf("Escribe el saludo...\n");
    write(fd, saludo, strlen(saludo));
    close(fd);

    fd = open(archivo,0);
    printf("Contenido del fichero: \n");

    bytesleidos = read(fd, buffer, 1);
    while (bytesleidos != 0){
        printf("%s", buffer);
        bytesleidos = read(fd, buffer, 1);
    }
    close(fd);
}

```

7. Pipe Prueba

```

gcc: no se encontró la orden
alexander@alexander-VirtualBox:~/Descargas$ gcc practica7.c -o test7
practica7.c: In function 'main':
practica7.c:24:13: warning: implicit declaration of function 'wait' [-Wimplicit-
function-declaration]
    wait(NULL); // espera que finalice proceso hijo
    ^
alexander@alexander-VirtualBox:~/Descargas$ ./test7
El HIJO escribe en el pipe...
El PADRE lee del pipe...
Mensaje leído: Hola papi

```

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int fd[2];
    char buffer[30];
    pid_t pid;

```

```
pipe(fd); // se crea el pipe

pid = fork(); // se crea el proceso hijo

switch(pid) {
    case -1: // ERROR
        printf("NO SE HA PODIDO CREAR HIJO...\n");
        exit(-1);
        break;
    case 0: // HIJO
        printf("El HIJO escribe en el pipe...\n");
        write(fd[1], "Hola papi", 10);
        break;
    default: // PADRE
        wait(NULL); // espera que finalice proceso hijo
        printf("El PADRE lee del pipe...\n");
        read(fd[0], buffer, 10);
        printf("\tMensaje leído: %s\n", buffer);
        break;
}
}
```

8. Pipe Prueba 2

```
alexander@alexander-VirtualBox:~/Descargas$ ./test8
El HIJO ENVIA ALGO AL pipe...
El PADRE recibe algo del pipe: Buenos dia
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main (void)
{
    int fd[2];
    pid_t pid;
    char saludoPadre[]="Buenos dias hijo.\0";
    char buffer[80];

    pipe(fd); //creo pipe
    pid=fork(); //creo proceso

    switch (pid) {

        case -1 : //ERROR
            printf("NO SE HA PODIDO CREAR HIJO.");
            exit (-1);

        case 0 : //HIJO RECIBE
            close(fd[1]); //cierra el descriptor de entrada
            read(fd[0], buffer, sizeof (buffer)); //leo el pipe
            printf("\tEl HIJO recibe algo del pipe: %s\n",buffer);
            break;

        default : //PADRE ENVIA
            close (fd[0]);
            write (fd[1], saludoPadre, strlen (saludoPadre)); //escribo
en pipe
            printf("El PADRE ENVIA MENSAJE AL HIJO...\n");
            wait (NULL); //espero al proceso hijo
            break;
    }
    return 0;
}

```


9. Pipe Prueba 3

```
alexander@alexander-VirtualBox:~/Descargas$ ./test9
El HIJO ENVIA MENSAJE AL PADRE...
El PADRE recibe algo del pipe: Hola padre.
alexander@alexander-VirtualBox:~/Descargas$ █
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    int fd[2];
    pid_t pid;
    char saludoHijo[] = "Hola padre.\0";
    char buffer[80];

    pipe(fd);
    pid = fork();

    switch (pid) {
        case -1:
            printf("NO SE HA PODIDO CREAR HIJO.\n");
            exit(-1);

        case 0:
            close(fd[0]);
            write(fd[1], saludoHijo, strlen(saludoHijo));
            printf("El HIJO ENVIA MENSAJE AL PADRE...\n");
            close(fd[1]);
            break;

        default:
            close(fd[1]);
            read(fd[0], buffer, sizeof(buffer));
            printf("El PADRE recibe algo del pipe: %s\n", buffer);
            close(fd[0]);
            wait(NULL);
            break;
    }
}
```

```
    return 0;
}
```

10. PA's y Demás

```
alexander@alexander-VirtualBox:~$ ./miprimotest
    Soy el proceso pa11 2161; Mi padre es: 2160
        Soy el proceso pa121 2163; Mi padre es: 2162
            Soy el proceso pa12 2162; Mi padre es: 2160
                Soy el proceso pa1 2160; Mi padre es: 2159
Soy el proceso pa2 2164; Mi padre es: 2159
Soy el proceso padre: 2159, Mis HIJOS: 2160 y 2164 terminaron
```

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

void main(void)
{
    pid_t pa1, pa2, pa11, pa12, pa121;

    pa1 = fork(); // Crea el primer proceso hijo

    if (pa1 == -1) { // error al crear el proceso hijo
        printf("No se ha podido crear el proceso hijo pa1...\n");
        exit(-1);
    }

    if (pa1 == 0) { // Estamos en el primer hijo (pa1)
        pa11 = fork(); // Crea un hijo (pa11) dentro de pa1
        if (pa11 == 0) {
            // Este es el proceso pa11
            printf("\tSoy el proceso pa11 %d; Mi padre es: %d\n", getpid(),
getppid());
        } else {
            wait(NULL); // Espera a pa11
            pa12 = fork(); // Crea otro hijo (pa12) dentro de pa1

            if (pa12 == -1) {
                printf("No se ha podido crear el proceso hijo pa12 en el
HIJO\n");
                exit(-1);
            }
        }
    }
}
```

```

    }

    if (pa12 == 0) { // Estamos en el proceso pa12
        pa121 = fork(); // Crea un hijo (pa121) dentro de pa12
        if (pa121 == 0) {
            printf("\t\tSoy el proceso pa121 %d; Mi padre es: %d\n",
getpid(), getppid());
        } else {
            wait(NULL); // Espera a pa121
            printf("\t\tSoy el proceso pa12 %d; Mi padre es: %d\n",
getpid(), getppid());
        }
    } else {
        wait(NULL); // Espera a pa12
        printf("\tSoy el proceso pa1 %d; Mi padre es: %d\n", getpid(),
getppid());
    }
}
} else { // Estamos en el proceso padre
    wait(NULL); // Esperamos finalización del proceso hijo pa1
    pa2 = fork(); // Crea el segundo proceso hijo pa2

    if (pa2 == -1) { // error al crear el proceso hijo pa2
        printf("No se ha podido crear el proceso hijo pa2...\n");
        exit(-1);
    }

    if (pa2 == 0) { // Estamos en el segundo hijo pa2
        printf("Soy el proceso pa2 %d; Mi padre es: %d\n", getpid(),
getppid());
    } else { // Proceso padre
        wait(NULL); // Esperamos finalización del proceso hijo pa2
        printf("Soy el proceso padre: %d, Mis HIJOS: %d y %d
terminaron \n", getpid(), pa1, pa2);
    }
}

exit(0);
}

```