```c
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   simulating_pipe.c                                  :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: caliaga- <marvin@42.fr>                    +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2024/02/28 14:12:19 by caliaga-          #+#    #+#             */
/*   Updated: 2024/02/28 14:12:22 by caliaga-         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

/* → Simulating the pipe "|" operator in C ==> https://www.youtube.com/watch?v=6xbLgZpOBi8
→ fork,Pipe y EOF (EndOfFile), Linux,C ==> https://www.youtube.com/watch?v=a0G85clhv6o
→ execv(...): llamada al sistema, familia "exec". Linux, C ==> https://www.youtube.com/
watch?v=yIe_9lqng2I
→ Comunicar dos procesos "ls -l" y "wc" con un pipe y redireccionar la salida a un archivo
de texto. ==> https://www.youtube.com/watch?v=l-UhKLdh4aY ==>
→ Comunicar 3 procesos (ls, grep y wc) usando 2 pipes, fork(), dup2(), execlp(). Programar
en C, Linux ==> https://www.youtube.com/watch?v=8LdO09Ep9RY */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h> // System calls - family exec
#include <sys/wait.h>
#define READ    0    /* index pipe extremo escritura */
#define WRITE   1    /* index pipe extremo lectura */

void    create_pipe(int *pip, int fd[2])
{
    *pip = pipe(fd); //  Creamos el «pipe».
    if (*pip == -1)  //  Y controlamos errores del mismo.
        exit (1);
    //  Recibimos la declaración de un array de dos enteros para los «file descriptors»
    //  fd[0] - read (para leer) O_RDONLY
    //  fd[1] - write (pasa escribir) O_WRONLY
    //  «pipe(int fd)» es una función de envoltorio «wrap» de una estructura «struct»
    //  de dos enteros que van a alojar los «file descriptors» del «pipe».
    //  int fd[2];    int pip;    create_pipe(&pip, fd);
    //  Creamos el pipe en el procedimeinto padre.
};

void cretate_initial_child(int *pid, int *fd)
{
    char *argVec[]={"/usr/bin/ls", NULL}; // /usr/bin
    char *ex_env[]={NULL};  // Parámetro ENVIROMENT de execve()

    *pid = fork();  //  Creamos el proceso hijo.
    if (*pid < 0)  //  Control de errores del proceso hijo.
        exit (1);
    //*pid2 = pid; // Devolvemos el «pid». int *pid2
    if (*pid == 0)  // PROCESO HIJO.
    {
        // Redireccionamos la salida estandar al pipe.
        // Ya se puede cerrar el pipe entero fd[0] y fd[1].
        // Ahora cuando se ejecute «execlp» su salida estandar
        // será al «pipe» en modo «write».
        close(fd[READ]); // EXTREMO NO NECESARIO en el hijo.
        dup2(fd[WRITE], STDOUT_FILENO);
        close(fd[WRITE]);
        if (execve(argVec[0], argVec, ex_env) == -1)
            exit(1);
    }
    close(fd[WRITE]); // EXTREMO NO NECESARIO en el padre.
}

void cretate_middle_child(int *pid, int *fd, int *fd_2)
{
```

```c
    *pid = fork();  //  Creamos el proceso.
    if (*pid < 0)  //  Control de errores.
        exit (1);
    if (*pid == 0)  // PROCESO HIJO.
    {
        close(fd_2[READ]); // NO NECESARIO.

        dup2(fd[READ], STDIN_FILENO);
        close(fd[READ]);

        dup2(fd_2[WRITE], STDOUT_FILENO);
        close(fd_2[WRITE]);
        execlp("grep", "grep", ".c", NULL);
    }
    close(fd[READ]);
    close(fd_2[WRITE]);
}

void cretate_final_child(int *pid, int *fd)
{
    *pid = fork(); //  Creamos el proceso hijo.
    if (*pid < 0)   //  Control de errores del proceso hijo.
        exit (1);
    if (*pid == 0)  //  PROCESO HIJO.
    {
        dup2(fd[READ], STDIN_FILENO); // Redireccionamos la entrada estandar al pipe.
        close(fd[READ]); // Ya se puede cerrar el pipe entero fd[0] y fd[1].
        //  Ahora cuando se ejecute el proceso hijo
        //  su entrada estandar será al «pipe» en «read».
        execlp("grep", "grep", "red", NULL);
    }
    close(fd[READ]);
}

int main (void)
{
    int fd[2];
    int fd_2[2];
    int pipe[2];
    int pid[3];
    int i;

    create_pipe(&pipe[0], fd);
    cretate_initial_child(&pid[0], fd);

    create_pipe(&pipe[1], fd_2);
    cretate_middle_child(&pid[1],fd, fd_2);

    cretate_final_child(&pid[2], fd_2);

    //  * Mientra que el proceso hijo siga abierto el proceso padre
    //  «parent process» debe existir para que el proceso hijo no se
    //  quede huérfano waitpid(pid1, NULL, 0);
    i=-1;
    while (pid[++i] != 0)
    {
        waitpid(pid[i], NULL, 0);
    }

    return (0);
}
// gcc -Wall -Wextra -Werror simulating_pipe.c -o SP && ./SP
// sudo apt-get install valgrind ==> https://valgrind.org/docs/manual/quick-start.html
// gcc -g -O0 -Wall -Wextra -Werror simulating_pipe.c -o SP && valgrind --leak-
check=yes ./SP
```