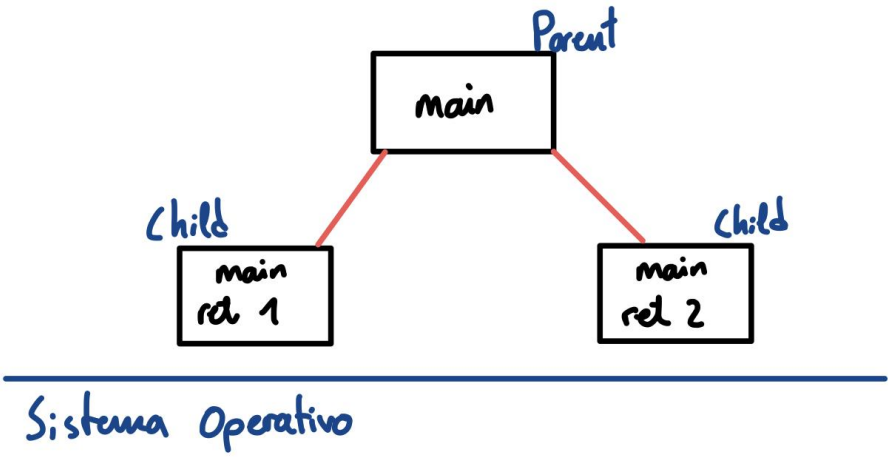
Ejercicio 1

a) Comenta brevemente (dibujando en caso que haga falta) cuál es la estructura de padre-hijos que se genera al ejecutar el código.
A partir del padre ejecutaremos dos *forks* creando así dos procesos hijo (ret1 y ret2). Los hijos no crearán más procesos copia debido a la instrucción *execv* (reemplaza la imagen del proceso con la especificada en *execv*; no se crea ningún proceso nuevo).

Esquema de avuda:



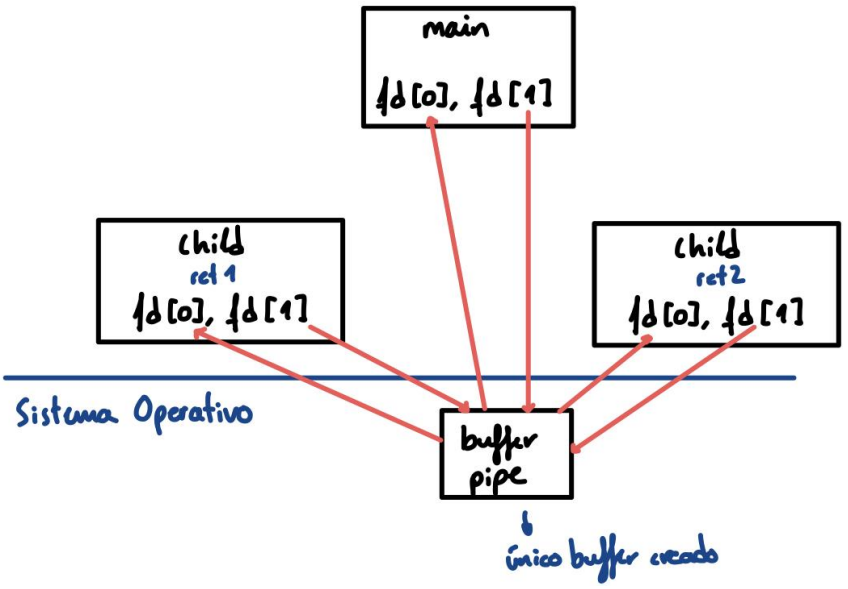
b) Basándonos en la salida mostrada del código, ¿qué partes del código son las que ejecutan primero y cuáles después? Razona tu respuesta.
Si solamente nos fijamos en la salida mostrada del código, el programa sigue el siguiente camino:

1. La primera parte que se ejecuta del código es el mensaje “Waiting…” (ya que aún haber hecho el *fork*, lo primero que se ejecuta es todo el código del padre o en paralelo) El padre esperará a que acabe ret2 (hijo) para finalizar (instrucción *waitpid*).
2. Posteriormente se imprime el mensaje del segundo condicional (if ret2 == 0): “Going to execute /usr/bin/wc”.
3. Entonces se imprime el mensaje del primer condicional (if ret1== 0): “Going to execute /usr/bin/find. Además, se ejecuta la instrucción *find* y se envía a través de la tubería al segundo condicional, ya que lo siguiente que vemos es el resultado de la ejecución del comando *wc -l*.
4. Finalmente se imprime el mensaje de “Finished”.

c) Comenta brevemente qué es lo que hace la línea 7 del código y dibuja el esquema de conexiones de comunicación asociado.
El programa crea un buffer interno en el sistema operativo que gestiona la comunicación para transferir datos entre procesos con relación padre-hijo. Además, crea dos descriptores de fichero: uno de escritura y otro de lectura.

Main es el programa original (o padre) y *child* el hijo del main. Cabe destacar que hay un total de 2 hijos (debido a los dos *forks* que realizamos en el código).

El esquema de conexiones es el siguiente:



d) Comenta brevemente qué es lo que hace la línea 9 del código. ¿Qué posible(s) valor(es) devuelve esta llamada a sistema?
La función fork crea un nuevo proceso, una copia del proceso padre.
La función fork devuelve dos veces, una del padre y otra del hijo. Para la del padre, devuelve el número que identifica al hijo; para el hijo devuelve 0.

e) Comenta brevemente la diferencia entre la línea 11 y la línea 21. ¿Qué proceso(s) entrará(n) en cada uno de estos casos?
En la línea 11 se está comprobando si el proceso actual es el proceso hijo creado por el comando ret1 = fork().

En cambio en la línea 21 se está comprobando si el proceso actual es el proceso hijo creado por la comanda ret2 = fork().

Ambos son procesos hijos del proceso padre (creado al ejecutarse el programa).

f) (0.5 puntos) ¿Qué es lo que hace la línea 13 del código? ¿Por qué se ejecuta esta línea? Es decir, ¿cuál es el objetivo de ejecutar esta línea?
Asocia fd[1] al descriptor 1 (conocido como salida estándar) para que funcione como salida (stdout).

Todo lo que se imprima al descriptor 1 estará asociado a fd[1], en lugar de la pantalla. Podemos decir que “se redirige la salida estándar de pantalla a fd[1] (a un fichero)”.

g) (0.5 puntos) De forma similar, ¿qué es lo que hace la línea 23 del código? ¿Por qué se ejecuta esta línea? Es decir, ¿cuál es el objetivo de ejecutar esta línea?
Asocia fd[0] al descriptor 0 (conocido como entrada estándar) para que funcione como entrada (stdint).

Todo lo que se lea en el descriptor 0 estará asociado a fd[0], en lugar de a teclado (que se captura con scanf). Podemos decir que “se redirige la entrada estándar de teclado a fd[0] (podemos redirigir un fichero a la entrada estándar)”.

h) En las líneas 16 y 26 se ejecutan dos procesos. Dadas las instrucciones ejecutadas anteriormente, ¿qué es lo que hace el código? Puedes apoyarte a responder la pregunta indicando cuál es el comando equivalente a ejecutar en el terminal.
Lo que hace el código es buscar en el directorio gutenberg elementos de tipo fichero (‘find gutenberg -type f’) para posteriormente contar cuántos hay (‘wc -l’).

i) ¿Qué es lo que realiza la línea 33? ¿Por qué es necesaria?
La instrucción *waitpid* se utiliza cuando un proceso crea varios hijos (como es el caso) y el padre necesita supervisar a un hijo en concreto.
La instrucción suspende la ejecución del proceso en curso hasta que el hijo especificado por argumento (aquí ret2) haya terminado o hasta que se produzca una señal cuya acción es finalizar el proceso actual.

Ejercicio 2

a) La IEEE (Institute of Electrical and Electronics Engineers) desarrolló el estándar POSIX para facilitar al núcleo de cualquier sistema UNIX el envío de excepciones a los procesos.

INCORRECTA

POSIX (Portable Operating System Interface) és un estàndard per mantenir compatibilitat entre diferents implementacions del sistema operatiu Unix.

Unix was selected as the basis for a standard system interface partly because it was "manufacturer-neutral". However, several major versions of Unix existed—so there was a need to develop a common-denominator system.V VC C

The goal of POSIX is to ease the task of cross-platform software development by establishing a set of guidelines for operating system vendors to follow.

b) La llamada a sistema pipe permite a los sistemas operativos UNIX realizar operaciones de comunicación entre procesos padre, hijos y nietos (un nieto es el hijo creado por un hijo).

La canonada es una forma bàsica de comunicació interprocés: l’objectiu és dirigir les dades que un procés A genera cap a un altre procés B.

Les canonades s’utilitzen per comunicar diferents processos. Aquests processos han de tenir una relació pare-fill. Llavors, aquesta afirmació és **INCORRECTA**, ja que no comuniquem pare-fill-net, sinó que són comunicacions pare-fill; és a dir, segons l’afirmació hauria de ser una comunicació de l’estil: pare - fill_1 ; fill_1 (pare ara) - fill2 (segons l’afirmació, el net).

c) Los procesos ejecutados por un usuario sin derechos de administrador (pej, oslab) de un sistema UNIX no podrán ejecutar instrucciones en modo núcleo al realizar una llamada a sistema.

Una crida a sistema és qualsevol funció proveïda pel sistema operatiu i que pot ser cridada per l'usuari.

És la interfície que ofereix el sistema operatiu als processos per accedir als dispositius o realitzar una altra operació a què només té dret d'accés el nucli.

INCORRECTE Todos los usuarios del sistema, tanto si tienen derechos de administrador como si no, pueden ejecutar instrucciones en modo núcleo al realizar una llamada al sistema.

d) Las interrupciones son señales síncronas utilizadas por el sistema operativo para tomar el control del mismo.
INCORRECTE Una interrupció és un senyal asíncron de maquinari produït per un esdeveniment (no produït per l'execució del procés).

Es incorrecto, las interrupciones son señales asíncronas debido a que pueden producirse en cualquier instante de la ejecución del programa, sin acompañar al compás de los ciclos del clock.

e) En un sistema con una única CPU, dos procesos pueden estar ejecutándose en un instante determinado, cada uno utilizando el 50 % de la CPU.
Es incorrecto, en un sistema con una única CPU dos procesos no pueden ejecutarse en un instante determinado a la vez, siempre el sistema va alternando entre la ejecución de uno y la ejecución del otro.

Tot i que només hi hagi una única CPU, el sistema operatiu s'encarrega d'executar múltiples processos al mateix ordinador al mateix temps.

El sistema operatiu ho aconsegueix executant cada procés només per unes desenes o centenars de milisegons. En acabar aquesta "llesca de temps" per al procés el sistema operatiu s'encarrega de canviar de procés.

En cada instant de temps només s'executa un únic procés, però en un segon poden executar múltiples processos (inclús el mateix procés múltiples vegades), donant a l'usuari l'il·lusió de paral·lelisme.