UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema Operativos

Examen 2

Comunicación y Sincronización de Procesos Respaldo de archivos

Integrantes

Andres Urbano Guillermo Gerardo.

Guadarrama Ortega César Alejandro.

Fecha entrega: 17 de Noviembre 2020.

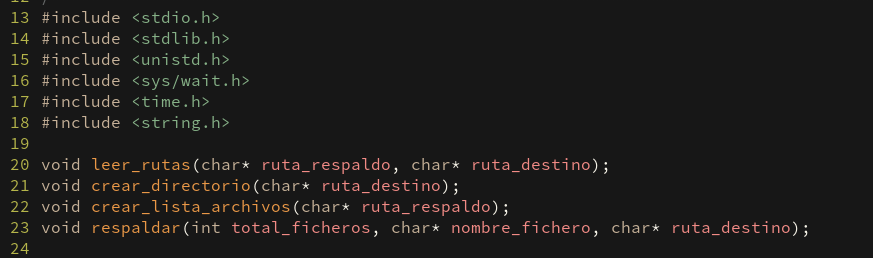
Ing. Patricia Del Valle Morales.

**Descripción general**

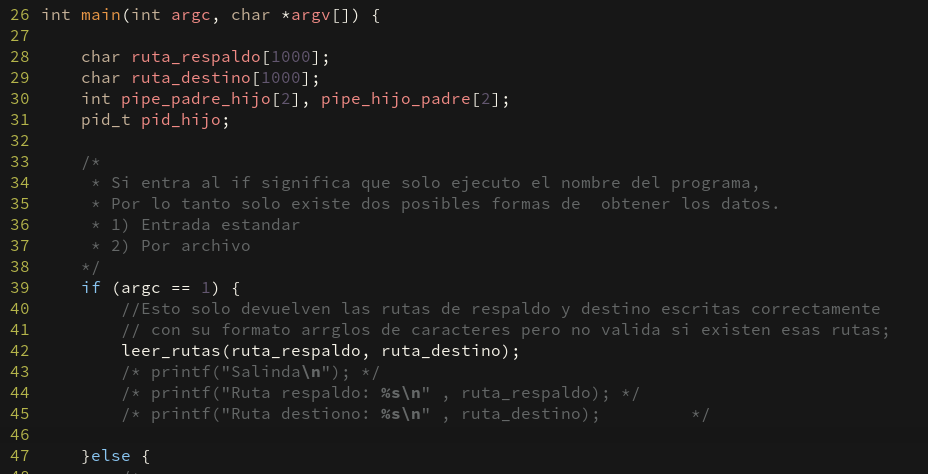
Realizar un programa en lenguaje C para la realización de backups automáticos internos, el programa utilizará procesos y pipes para distribuir el trabajo y que se hagan paralelamente.

**Análisis**

Primeramente incluimos todos nuestras bibliotecas necesarias que nos proporcionan las funciones y datos necesarios para la realización de nuestro programa.

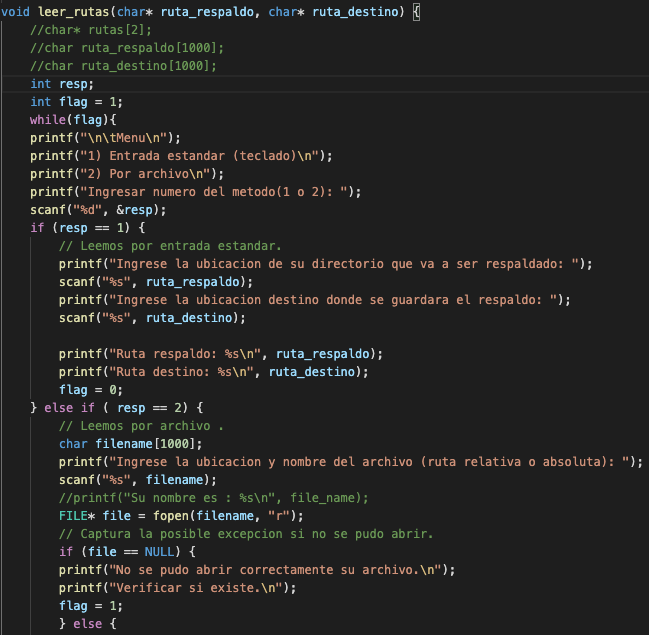
****

Utilizamos un enfoque modular para la realización de backups, abstrayendo la complejidad en pequeñas funciones que nos serán útiles a lo largo del programa.



Iniciamos con nuestra función main que es la se ejecutara al inicio de la ejecución, aprovecharemos los argumentos que sean pasados por el programa. Declaramos nuestras primeras variables, conforme vayamos avanzando, se irán explicando paso a paso. Observaremos que al inicio utilizamos un if, esto es debido a que si el el usuario no ingresa como parámetros las rutas de respaldo y destino en el momento de ejecutar nuestro programa, llamamos a nuestra función leer\_rutas() la cual será un menú interactivo para que el usuario ingrese las rutas de destino y respaldo, ya sea vía terminal por entrada estándar o ingresando el nombre del archivo que las contiene.

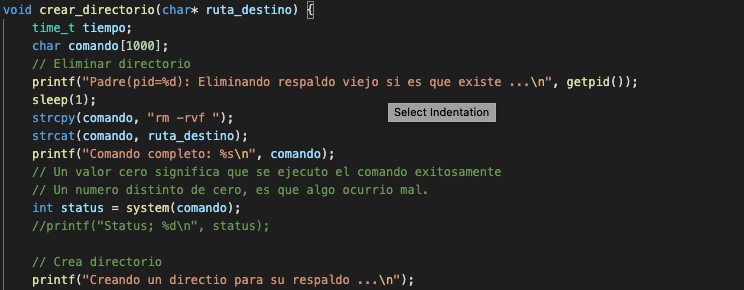
En la función leer\_rutas() es la primera en ejecutarse que prácticamente es un menú del programa, en el caso que no ingrese las rutas como parámetros**.** La función le pide al usuario que coloque el tipo de opción que desea para leer las rutas, las cuales sería la de destino y respaldo. Las opciones pueden ser por un archivo o por la entrada estándar. Si el directorio o el archivo no es localizado correctamente, entonces manda una impresión de pantalla indicando su error y vuelve al menú para volver escoger una opción.

****

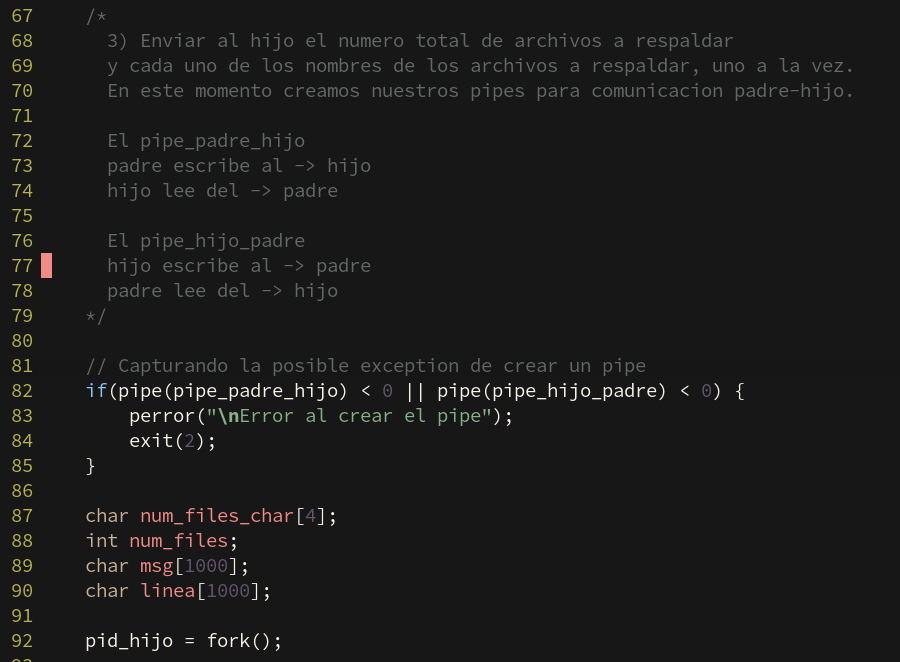
La función crear\_lista\_archivos, crea como tal la lista de los archivos del directorio que va hacer respaldado. Tenemos como parámetro la ruta de respaldo solamente, ya que en ella ejecutaremos llamadas al system para ejecutar los comandos ls y wc respectivos, para guardar la nombres de los archivos a respaldar en un archivo llamado lista\_archivos.txt que se localizara en la carpeta actual donde se ejecute el programa.

****

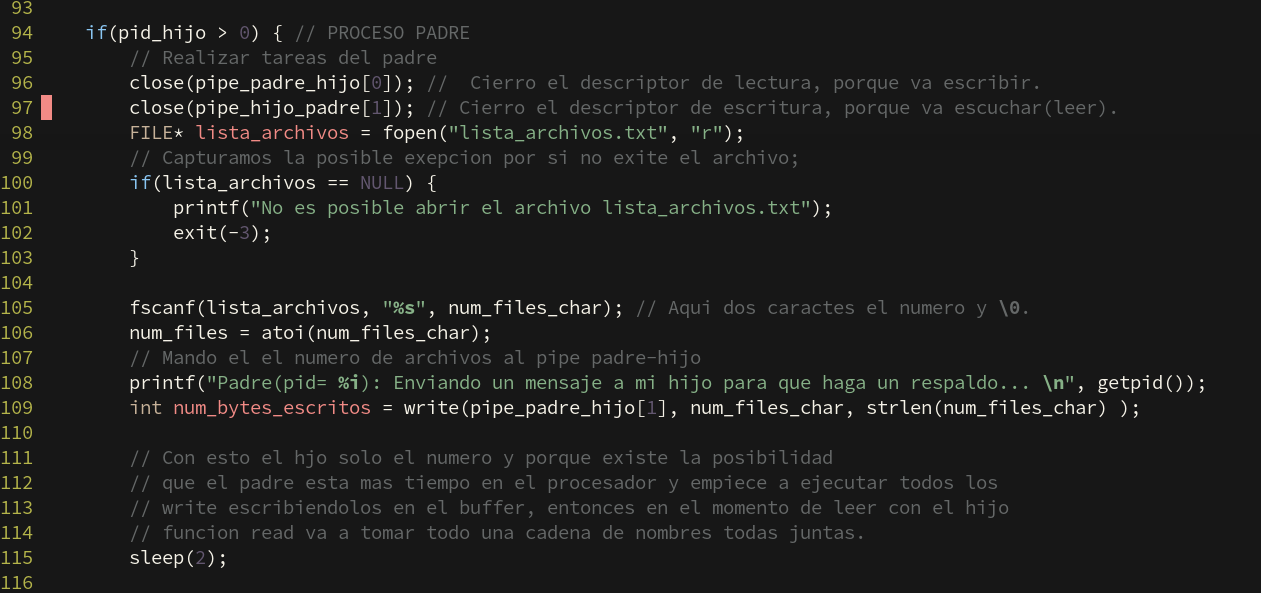
Antes de poner a trabajar al padre crearemos el directorio donde el usuario quiere su respaldo como también eliminaremos si es que ya existe, debido a podría haber conflictos sobre nuestros archivos para verificar si realmente está cumpliendo con hacer backups. La función crear\_directorio recibe solo el parámetro ruta\_destino que se proporcionó anteriormente por el usuario. Al igual que en el anterior código tenemos que concatenar cadenas para hacer una llamada al sistema y ejecutar comandos de Linux. Esta llamada creará el directorio con el comando “mkdir”.



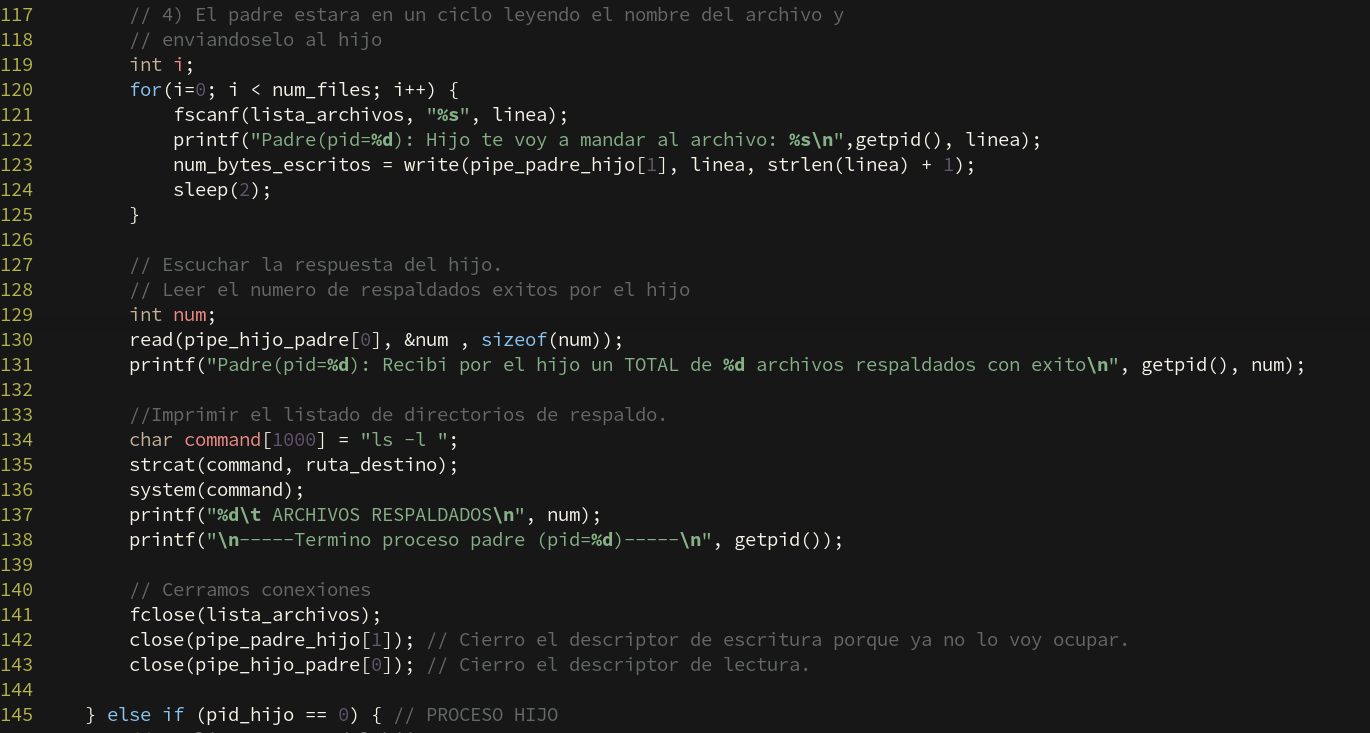
Ya con esto estaría todo listo para que empiece el proceso del padre, primeros crearemos nuestros pipes para la comunicación padre-hijo, utilizaremos dos pipes debido a que será una comunicación bidireccional, una vez creados vamo a verificar si se creamos correctamente utilizando un if seguido de una proposición lógica, garantizando que se hayan creado correctamente.



Declaramos nuestras variables que serán demasiado útiles después, una vez declaradas nuestras variables es momento de clonarnos y crear nuestro primer hijo.



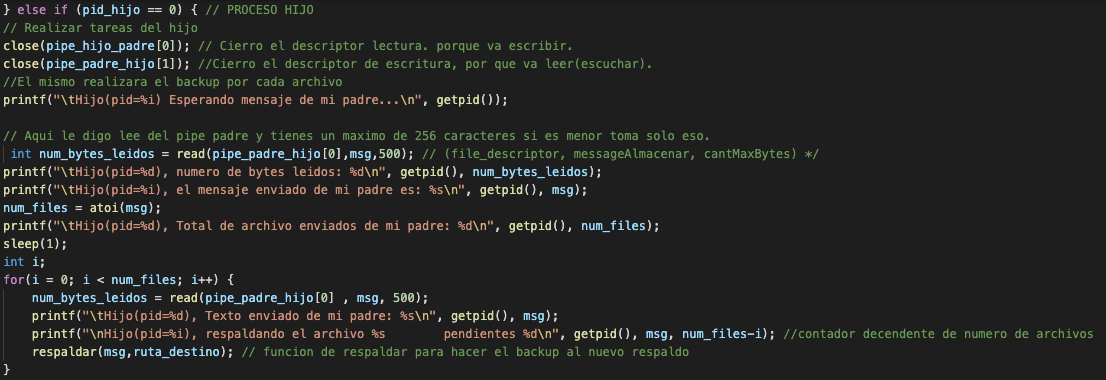
Al igual que los pipes vamos prevenir cualquier posible excepción que pueda surgir, utilizaremos if else if para el pid de nuestro, que nos indicará si somo el padre, el hijo o salir error si es un valor menor que cero, en el primer caso cuando somo el padre, vamo cerrar los descriptores que no nos corresponde con la función close indicando el descriptor, una vez hecho esto abrimos nuestra lista\_archivos.txt donde contendrá todos los nombres que serán respaldados, verificamos si nuestro archivo se abrió correctamente, una vez hecho esto en la primera línea de nuestro archivo corresponde a la cantidad total, vamos a leerla para después pasarlo a nuestro hijo. Utilizaremos la función write para escribir al pipe de nuestro padre para que nuestro hijo lo puede leer de ahí, una cosa importante es la siguiente función sleep, está será invocada debido a que nuestro proceso padre puede ser estar mucho más tiempo en el procesador, debido a los quantus que le da la CPU a un proceso, debido a esto podría ser que nuestro proceso intentara escribir todas las líneas a su pipe, pero el hijo podría confundirse al momento de leer, ya que la función read toma como parámetros la cantidad de bytes a leer, y por consiguiente podría tomar más de un nombre que le corresponde y combinar los nombres, esa la razón principal por la cual utilizamos sleep para que no suceda y que el hijo puede leer nombre por nombre.

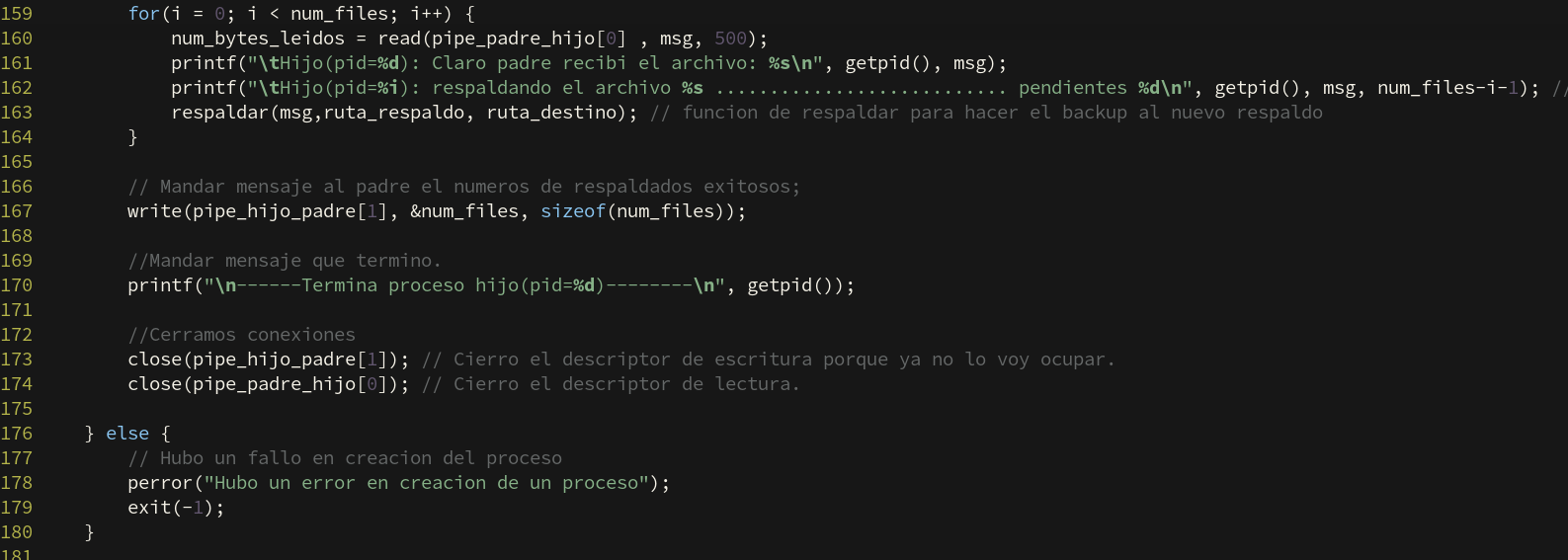


Ahora en las siguiente líneas de código, se verá una de las sentencias más importante que hace que nuestro programa funcione, es precisamente nuestra sentencia for, en ella vamos leyendo linea por linea de nuestro archivo que contiene los nombres que vamos a respaldar y vamos escribiendo al pipe correspondiente para que el hijo puede leer del pipe y puede hacer los respaldos.

Una vez terminado de enviar todos los nombres de los archivos, es solo cuestión de esperar al hijo, para que le mande al padre el número total de respaldos exitoso, una vez hecho esto solo no queda imprimir el directorio donde se hicieron los respaldos con la función system y con ayuda de la función strcat para concatenar cadenas y formar nuestros comandos. Y por último cerrar los pipes que ya no ocupemos. Una vez terminado con eso llegará el turno cuando el PID del proceso sea igual a cero que corresponde con las tareas del hijo.

Al igual que con el padre cerramos los descriptores correspondientes e imprimos los mensajes que digan que estamos esperando al padre, Una vez leído el numero de archivos que vamos a leer proporcionado por el pipe, vamos realizar un for que realice todas las lecturas de los archivos, la función read recibe el número de bytes que vamos al leer. Estos datos son importantes para poder realizar la función de respaldar. Utilizando la programación modular que es utilizar funciones para encapsular las complejidades, creamos una funcion precisamente que haga los respaldos pasándose el nombre del fichero y las rutas de respaldo y destino.

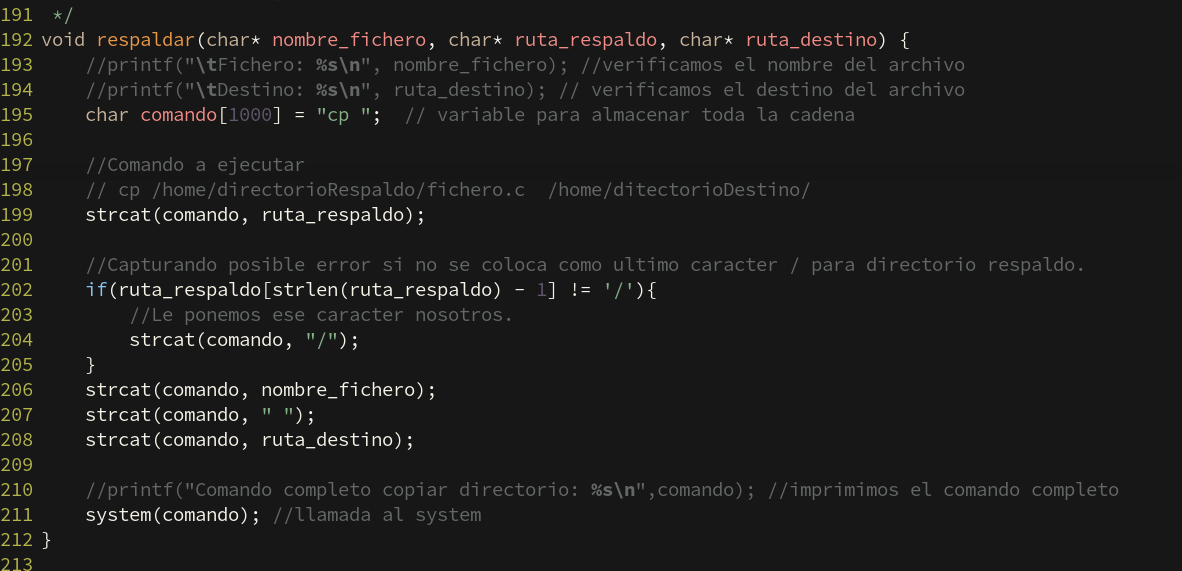
****



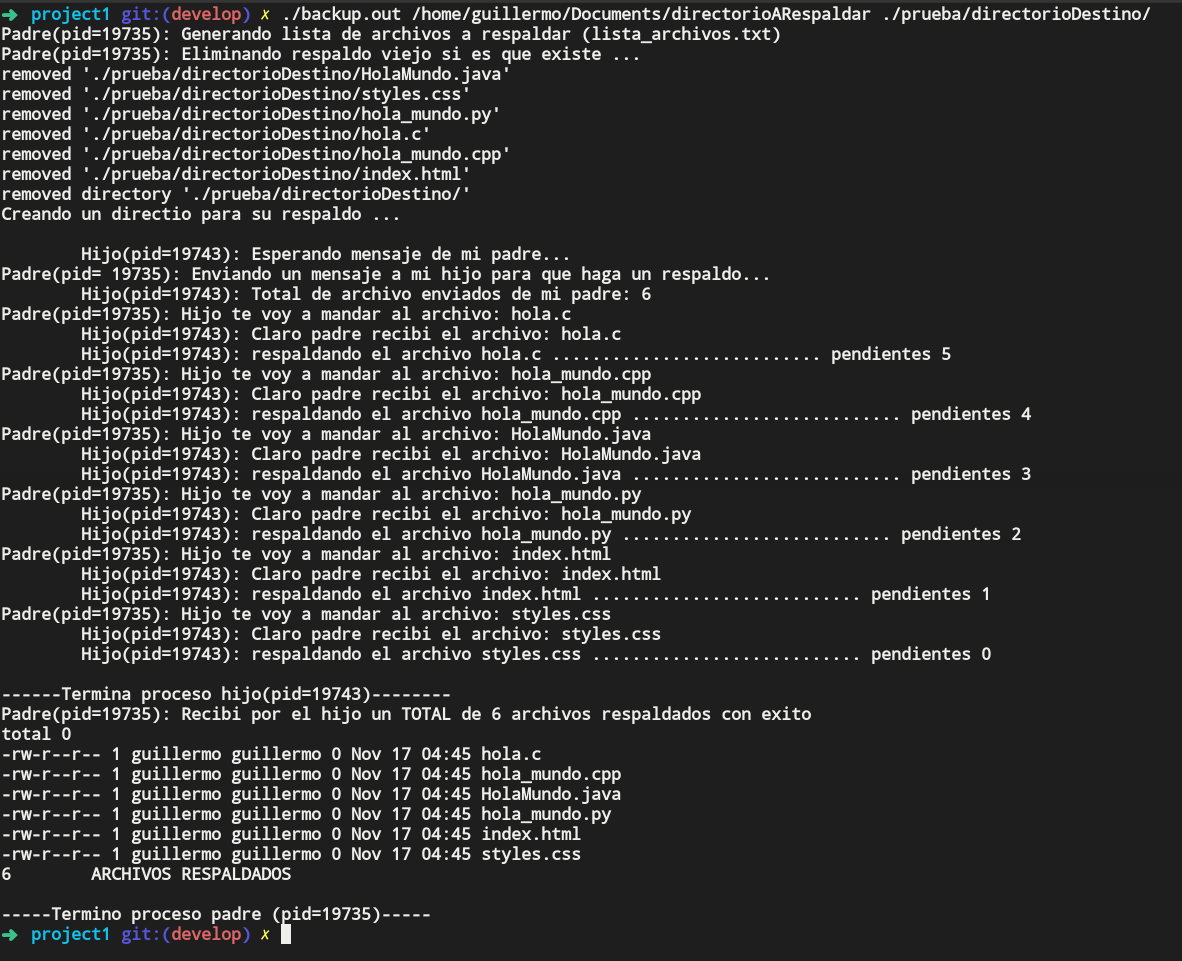
Por último el hijo imprimirá el archivo que está leyendo y cuántos están pendientes y acabará el ciclo, después de eso nos queda por mandarle un mensaje al padre sobre los archivos de respaldos exitosos y mandar una impresión sobre el proceso terminado y finalmente se cerrar los pipes.

En la función respaldar podemos ver que como parámetros recibe el nombre del fichero o archivo y las rutas de respaldo y destino. Dentro de la función comprobamos si los datos de los parámetros de la función son correctos, esto es para tener un seguimiento del programa, pero se puede omitir por completo.

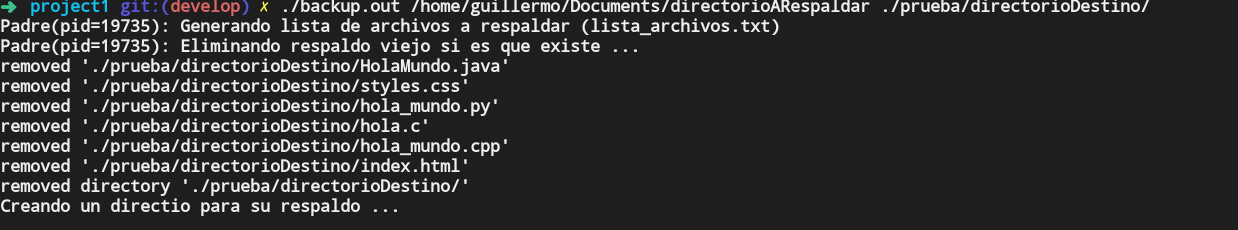
Después la función crea variables auxiliares para realizar nuestros comandos con función system y utilizar las rutas para concatenar con los comandos y así poder realizar el comando cp de Linux que nos permitirá realizar las copias de los archivos.

****

**Corrida del programa**



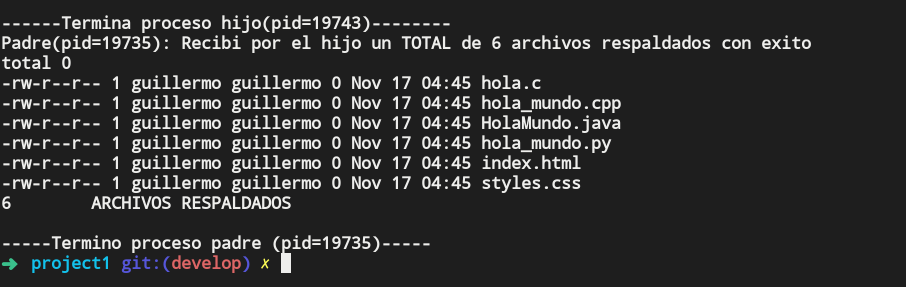
**Analizando la salida**



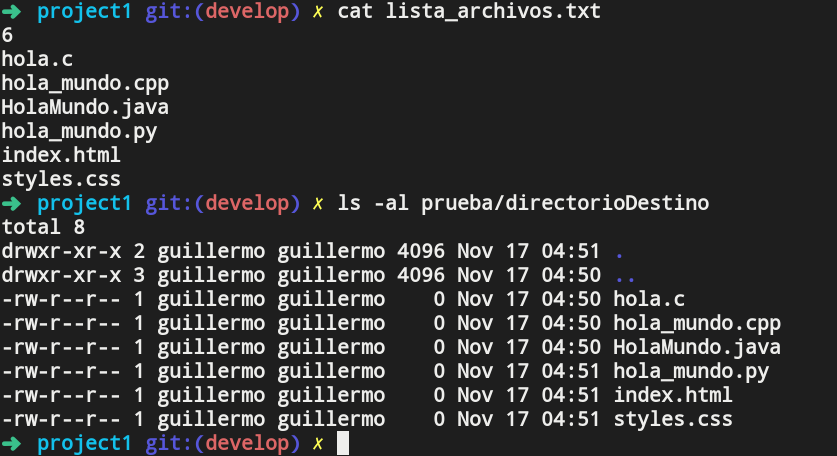
Si la carpeta destino ya existe, eliminará esa carpeta y creará una nueva.

****

Aquí observamos como el padre le va pasando los nombres de los archivos a los hijos y el hijo lo va escuchando.



Por último el padre enlista la carpeta donde se guarda los respaldos.



Aquí comprobamos que es verdad que sí se realizó.

**Reflexión Andres Urbano Guillermo Gerardo**

La creación de procesos y comunicación en ellos es sumamente entretenido, buscar las maneras de poder comunicarse adecuadamente y que no haya conflicto es retante pero a la vez te deja una gran satisfacción, no solamente por el hecho de resolverlo sino también por hecho de relacionarlos con otras cosas, por ejemplo en el desarrollo web cuando deseamos comunicarnos con una api con javascript, realmente estamos utilizando hilos asíncronos para resolver la tarea o ver la relación que tienen nuestros comandos unos con otros en nuestra terminal, cuando ejecutamos uno y está esperando a otro programa para terminar de ejecutarlo. Posiblemente una de las dificultades que surgieron al realizar el programa fue el correcto entendimiento de las función read y write, utilizadas para mandar información a los pipes, indudablemente fue mi debilidad al realizar el programa, ya que cuando es solamente un mensaje es sencillo pero cuando mandamos más de uno a los pipes, siento que es poco confuso entender ese aspecto sobre cómo están llegando las cosas al escribir o al leer.

Sin duda alguna este conocimiento será de gran ayuda en la vida profesional, es lo que nos hace diferentes sobre otras personas que solo estudian computación, además de que hoy en día la capacidad de los procesadores va incrementando y surge la necesidad de aprovechar al máximo la CPU e indudablemente aparecen términos como programación paralela o concurrente.

**Reflexión César Alejandro Guadarrama Ortega**

Utilizar pipes para trabajar con procesos, para mi aun es un poco complicado. La manera en cómo se comunican más allá de mandar un simple mensaje hace que ocurran varios errores en la ejecución del programa, como directorio vacío, nombres incompletos, llamas al sistema incorrectas o copias incompletas de archivos. Sin embargo es muy interesante la interacción de los procesos por medio de uno o más pipes. Hablando como equipo tardamos en llegar a la solución por falta de planteamiento por lo menos en el proceso hijo.