Repositorio de Entregas

Cada rama tiene su versión del código. Nosotros vamos a generar una primera rama donde vamos a integrar el esqueleto del lab. Generamos un checkpoint en ese repositorio con el estado del esqueleto inicial. Después vamos a trabajar sobre una segunda rama que es la de la entrega y donde vamos a estar subiendo el código. La forma de entregar el lab es generar un pull request que es una forma que nos da git de mostrar diferencias entre dos estados de código. Comparamos el estado base del esqueleto con el estado del TP terminado.

Para este lab, trabajamos sobre la rama entrega\_fork. Eidtamos, commiteamos etc todo ahí.

Una vez que terminamos, desde github en nuestro repo vamos a pull requests y generamos uno nuevo entre base\_fork contra entrega\_fork. Van a aparecer todas las diferencias de código que hicimos para resolver y agregamos de título: entrega lab fork y podemos poner comentarios, etc.

Repositorio de Pruebas

<https://github.com/fisop/labs-tests>

Para lab fork y lab shell.

Leer el README antes :)

Entramos a …/fork/ y ejecutamos ./test-fork [ruta donde tenemos el lab fork/fork/]

## Syscall wait(2)

Espera a que algún proceso hijo termine. Es bloqueante si hay procesos que esperar. Nos devuelve el processid del hijo que terminó.

* WIFEXITED(wstatus): returns true if the child terminated normally
* WEXITSTATUS(wstatus): returns the exit status of the child
* WIFSIGNALED(wstatus): returns true if the child process was terminated by a signal
* WTERMSIG(wstatus): returns true if the child process was terminated by a signal
* getppid(2): returns the parent process PID
* waitpid(2): allows to wait a specific child process
* Ver WNOWAIT y WNOHANG

### huérfanos y zombies

Cuando un proceso termina toda la memoria que pidió se retorna al pull de memoria libre, todos los archivos que abrió se cierran, todos los recursos que el proceso haya pedido se restaura y el proceso a efectos prácticos muere pero el kernel no lo borra de la lista de procesos sino que le deja una meta data mínima y guarda el código de salida. Todo proceso que termina va a parar a ese estado de que sigue existiendo pero está en un estado minimalista donde se guardó solamente la información de salida y nada más. Eso es un proceso **zombie**: no está muerto del todo pero tampoco va a volver a correr. Eso es lo que pasa si un proceso hijo termina antes de que el proceso le haga wait. La syscall wait encuentra un proceso que esté en estado zombie, decirle al kernel que libere ese espacio y me devuelva la metadata que guardó para ese proceso y se la devuelvo al usuario. En ese momento el zombie muere completamente y se libera ese espacio en el kernel. Si el padre no tiene ningún proceso zombie pero tiene hijos corriendo espera a que alguno de ellos muera, que el kernel lo transforme en un zombie y recién ahí el wait devuelve sacando esa metadata.

Un proceso **huérfano** es un proceso que sigue corriendo, o bien está en estado zombie, pero el proceso que le podía hacer wait murió y no le hizo wait, lo dejó abandonado. Si un proceso huérfano murió es huérfano y zombie :)

### subreaper

Adopta a los procesos huérfanos y los mata.

Lab Fork II

xargs

Especie de clonador de comandos. Recibe un comando y una lista. Genera un nuevo programa ejecutando el comando que le pasamos y para cada línea de entrada da un comando.

execve(2) para generar nuevos procesos.

Se recomienda usar execvp()

find

Va a aceptar una cadena y recorrer recursivamente todos los directorios donde estemos parados e imprimir aquellos archivos que matcheen en el nombre del archivo con la cadena que le pasamos.

Lo implemento con recursividad excepto cuando encuentro “.” o “..”.