

CC4302 Sistemas Operativos

Tarea 3 – Semestre Otoño 2017 – Prof.: Luis Mateu

En esta tarea Ud. deberá programar un driver para Linux que implemente un buffer de 10 bytes que servirá para comunicar procesos pesados. Un proceso envía datos por el pipe escribiendo en `/dev/pipe`. Un proceso recibe los datos leyendo de `/dev/pipe`. El dispositivo `/dev/pipe` debe tener número *major* 61.

El siguiente ejemplo usa los comandos estándares de Linux `\echo` y `cat` para demostrar el comportamiento esperado de `/dev/pipe`. Su driver debe reproducir exactamente el mismo comportamiento. Si hay aspectos que el ejemplo no aclara, decida Ud. mismo tratando de simplificar su tarea. Las filas de la tabla están ordenadas cronológicamente. Lo que escribió el usuario aparece en **negritas**. Observe que el prompt `$` indica cuando debe terminar un comando. Si el prompt `$` no aparece es porque hay una llamada al sistema pendiente (como `read` o `write`).

Shell 1	Shell 2
<pre>\$ \echo "abc" > /dev/pipe ⁽¹⁾ \$ \echo \$? 0 \$ \echo "def" > /dev/pipe ⁽¹⁾ \$ \echo "ghi" > /dev/pipe ⁽²⁾</pre>	
<pre>\$</pre>	<pre>\$ cat /dev/pipe ⁽³⁾ abc def ghi</pre>
<pre>\$ \echo "los 4 puntos cardinales son 3:" \ > /dev/pipe ⁽⁴⁾ \$ \$ \echo "el norte y el sur" > /dev/pipe ⁽⁴⁾ \$</pre>	<pre>los 4 puntos cardinales son 3: el norte y el sur</pre>
	<pre><control-C> ⁽⁵⁾ \$ \echo \$? 1 \$</pre>
<pre>\$ \echo "hola que tal" > /dev/pipe ⁽⁶⁾</pre>	
<pre><control-C> ⁽⁷⁾ \$ \echo \$? 1 \$</pre>	
	<pre>\$ cat /dev/pipe ⁽⁸⁾ hola que t</pre>
	<pre><control-C> ⁽⁵⁾ \$</pre>

Notas:

- (1) Se invoca el comando `\echo` en el shell 1 para escribir 4 caracteres por `/dev/pipe` (los caracteres a, b y c seguidos de un *newline*). Ud. debe usar `\echo`. Si usa solo `echo`, sin el `\`, se invocará el que viene integrado en el shell. Este podría no actuar correctamente ante un *control-C*. Su driver debe implementar *write* de tal modo que se depositen los 4 caracteres en el buffer y se retorne de inmediato, haciendo que `\echo` termine. El segundo `\echo` envía los caracteres d, e, f y *newline* por `/dev/pipe` completando 8 bytes en el buffer. *Write* debe retornar de inmediato para que `\echo` termine.
- (2) El tercer `\echo` envía 4 nuevos caracteres. El buffer se llena después de depositar la g y la h. Su driver debe implementar *write* de modo que este se bloquee hasta que un proceso use *read* para desocupar el buffer.

- (3) Se invoca el comando *cat* en el shell 2 para leer de */dev/pipe*. Se invocará una vez *read* para leer probablemente unos 8192 bytes. Ud. debe hacer que *read* entregue solo lo que hay disponible en el buffer, es decir 10 bytes, vaciando el buffer. Entonces Ud. debe retomar el *write* pendiente para depositar los 2 bytes faltantes: la *i* y el *newline*, con lo que el tercer *lecho* termina. El comando *cat* invocará *read* por segunda vez. Su implementación de *read* debe entregar solo los 2 bytes recién depositados, vaciando el buffer. El comando *cat* invocará *read* por tercera vez. Como el buffer está vacío, su implementación debe hacer que *read* se bloquee hasta que se escriban nuevos bytes en */dev/pipe*.
- (4) El comando *lecho* envía con un solo *write* una sentencia de *n* bytes que exceden el tamaño del buffer. Su implementación de *write* debe llenar el buffer, esperar a que un *read* vacíe el buffer, luego rellenar el buffer y esperar nuevamente, hasta que se hayan depositado los *n* bytes solicitados en ese único *write*. Por otra parte el comando *cat* hace varias invocaciones de *read* para vaciar en cada ocasión el buffer. Cuando *read* encuentra el buffer vacío, se debe esperar hasta que se deposite al menos un byte.
- (5) El usuario usa *control-C* para que el núcleo envíe la señal SIGINT al proceso *cat*. Como el proceso se encuentra bloqueado en una invocación de *read*, Ud. debe hacer que *read* retorne -EINTR, con lo que el proceso *cat* termina con código de retorno 1.
- (6) El comando *lecho* envía con un solo *write* una sentencia de *n* bytes que exceden el tamaño del buffer. Su implementación de *write* debe hacer que *write* se bloquee hasta que se depositen los *n* bytes. Pero esto no ocurrirá porque ningún proceso está leyendo de */dev/pipe*.
- (7) El usuario usa *control-C* para que el núcleo envíe la señal SIGINT al proceso *lecho*. Como el proceso se encuentra bloqueado en una invocación de *write*, Ud. debe hacer que *write* retorne -EINTR, con lo que el proceso *lecho* termina con código de retorno 1.
- (8) El comando *cat* lee del buffer de */dev/pipe* los 10 caracteres que el último *lecho* alcanzó a depositar. El *control-C* del usuario termina la ejecución de *cat*.

Restricción importante: Su driver debe implementar un buffer de solo 10 bytes. En ningún caso use un arreglo de más de 10 caracteres.

Ayuda: Este problema es similar al buffer del productor/consumidor. La operación *put* corresponde al *write* y *get* al *read*. La diferencia es que *write* y *read* depositan/extraen múltiples ítems. Cada ítem corresponde a un byte. Por simplicidad, cuando invoque *copy_to_user* o *copy_from_user*, transfiera de a un solo byte. Ignore completamente el parámetro **f_pos*. Las operaciones *open* y *write* no hacen nada. Basta que retornen 0.

Recursos

Baje de U-cursos el archivo *modules2017-1.tgz*. Contiene enunciados y soluciones de tareas de semestres anteriores con instrucciones para compilarlas y ejecutarlas (ver archivos *README.txt* en cada directorio). Además se adjunta un tutorial sobre programación de módulos y drivers en Linux. Le será de especial utilidad el directorio *Syncread* con el enunciado y la solución de la tarea 3 del semestre otoño de 2013 (que se vio o se verá en clase auxiliar).

Programe su solución en el archivo *pipe-impl.c* del directorio *Pipe*. En ese directorio encontrará un *Makefile* para compilar su tarea y un archivo *README.txt* con las instrucciones para crear el dispositivo */dev/pipe*. Puede resolver su tarea usando los mutex y condiciones incluidos en el directorio *Pipe*. Como ejemplo de utilización estudie la solución de *Syncread*. Estos mutex y condiciones son análogos a los de pthreads y están implementados a partir de los semáforos del núcleo de Linux en el archivo *kmutex.c* con encabezados en *kmutex.h*. Sin embargo, si lo prefiere no es difícil resolver esta tarea usando solo los semáforos del núcleo de Linux.

Antes de cargar y probar su tarea asegúrese de ejecutar el comando Unix *sync* para garantizar que sus archivos hayan sido grabados en disco y no están pendientes en un caché de Unix. Recuerde que los errores en su driver pueden hacer que Linux se bloquee indefinidamente y tenga que reiniciar el sistema operativo.

Entrega

La tarea se entrega *funcionando* en U-cursos. Para ello entregue solo el archivo *pipe-impl.c* que implementa el driver pedido. Se descontará medio punto por día de atraso, excepto días sábado, domingo o festivos. Resuelva la tarea antes del control 3, le servirá de estudio.