CC4302 Sistemas Operativos

Tarea 7 – Semestre Primavera 2025 – Prof.: Luis Mateu

En esta tarea Ud. deberá implementar un driver para Linux que permita *escribir de manera síncrona y con prioridad*. Una escritura síncrona (*write*) se bloquea hasta que haya una lectura (*read*). Ud. debe implementar este tipo de escrituras en 2 dispositivos: */dev/syncwrite0* con número *minor* 0 que se usará para las escrituras no prioritarias, y */dev/syncwrite1* con *minor* 1 que se usará para las escrituras prioritarias. Ambos con número *major* 65. Cuando se lee pero no hay ninguna escritura pendiente, *read* debe entregar 0 bytes leídos. Esto es normalmente interpretado por programas como *cat* como el fin de archivo.

El siguiente ejemplo usa los comandos estándares de Unix *echo* y *cat* para demostrar el comportamiento que se espera para */dev/syncwrite0* y */dev/syncwrite1*. Su driver debe reproducir exactamente el mismo comportamiento. Si hay aspectos que el ejemplo no aclara, decida Ud. mismo tratando de simplificar su tarea. Su tarea será probada con este mismo ejemplo. 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 *open*, *read* o *write*).

Shell 1	Shell 2	Shell 3	Shell 4
`$ echo abc > /dev/syncwrite0` (1)			
	`$ echo def > /dev/syncwrite0` (1)		

		`$ cat < /dev/syncwrite0` (2)
		`abc`
		`def`	

`$` (3)	`$` (3)	`$` (3)	
			`$ cat < /dev/syncwrite0` `$` (4)
	`$ echo ghi > /dev/syncwrite0`		
		`$ echo jkl > /dev/syncwrite1` (5)	
			`$ echo hij > /dev/syncwrite0`

`$ cat < /dev/syncwrite0`
`jkl` (6)
`ghi`
`hij`			

`$`	`$`	`$`	`$`

	`$ echo klm > /dev/syncwrite0`
	<*control-C*> (7)
	`$`		

Notas:

(1) El comando *echo* se implementa invocando *write* de sus argumentos en la salida estándar. Como esta salida es su dispositivo de escrituras síncronas, *echo* se debe bloquear hasta que alguien lea su dispositivo.

(2) El comando *cat* invoca *read* de su entrada estándar repetidamente y la arroja a la salida estándar hasta que el número de bytes leídos entregados por *read* sea 0. Por lo tanto muestra en pantalla lo escrito en los *shell 1* y *2* y termina. Ud. debe lograr que las lecturas se hagan en el mismo orden que se hicieron las escrituras, es decir primero *abc* y luego *def*.

(3) Observe que los 2 *echo* terminan inmediatamente después que *cat* leyó sus escrituras.

(4) El comando *cat* invoca *read*, pero este retorna de inmediato 0 bytes leídos, porque no hay ninguna escritura pendiente. Esto es el fin de archivo y *cat* termina.

(5) Observe que la escritura se hace en */dev/syncwrite1* y no en */dev/syncwrite0*. Esto significa que la escritura es prioritaria y por lo tanto se debe leer antes que cualquier escritura no prioritaria.

(6) Observe que *jkl* se leyó antes que cualquier otra escritura no prioritaria.

(7) La operación *write* debe ser interrumpible con *control-C* mientras está en espera.

Recursos

Baje de U-cursos el archivo *modules2020-2.tgz*. Descomprímalo con el comando:

```
$ tar zxvf modules2020-2.tgz
```

Contiene enunciados y soluciones de tareas de semestres anteriores con instrucciones para compilarlas y ejecutarlas (ver archivos README.txt en cada directorio). Le serán de especial utilidad los directorios *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) y *Pipe* con el enunciado y la solución de la tarea 3 del semestre otoño de 2017 que corresponde a un pipe compartido entre todos los procesos. El directorio *Syncwrite* contiene los archivos que Ud. necesitará para resolver esta tarea. Lea el archivo *README.txt*. Programe su solución en el archivo *syncwrite-impl.c* dentro del directorio *Syncwrite*.

Ayuda

Siga las instrucciones de *https://users.dcc.uchile.cl/~lmateu/CC4302/relator/* para ver una presentación sobre cómo programar módulos y drivers. Al final se explican los mutex y condiciones que Ud. necesita para resolver la tarea. 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*.

Ud. puede rescatar el número minor de un archivo con la expresión: *iminor(filp->f_path.dentry->d_inode)*

Antes de cargar y probar su tarea asegúrese de ejecutar el comando de 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. Abuse del comando *printk* para escribir en los logs del núcleo lo que hace su driver. Le ayudarán a depurar los errores que seguramente cometerá. No intente usar ddd o gdb.

Entrega

La tarea se entrega *funcionando* en U-cursos. Para ello entregue solo el archivo *syncwrite-impl.c* que implementa el driver pedido. Se descontará medio punto por día de atraso, excepto días sábado, domingo, festivos o recesos.