CC4302 Sistemas Operativos

Tarea 3 – Semestre Otoño 2019 – Prof.: Luis Mateu

En esta tarea Ud. deberá implementar un driver para Linux que implemente el buffer del problema del productor/consumidor. Para entender como funciona este driver compararemos su comportamiento con el de */dev/mem* que se vió en clases. Supongamos que un proceso abre */dev/mem* y escribe 10 bytes usando la llamada al sistema *write*. Luego escribe otros 20 bytes. Si un segundo proceso abre */dev/mem* y lee 100 bytes con *read*, entonces leerá 30 bytes que es todo lo que hay en */dev/mem* hasta ese momento. Se le pide programar un driver para el dispositivo */dev/buffer*. El buffer debe tener tamaño 3. El número *major* del driver debe ser 61. Se diferencia de */dev/mem* en que si se escriben en él 10 y 20 bytes sucesivamente, al leer 100 bytes de /dev/buffer se leerán solo 10 bytes. Habrá que abrir y leer una segunda vez */dev/buffer* para obtener exactamente 20 bytes, independiente de que a posteriori se hayan escrito más datos. En buenas cuentas si se escriben sucesivamente $n_1, n_2, \ldots, n_k$ bytes en cada apertura y lectura posterior se leerán $n_1, n_2, \ldots, n_k$ bytes.

El siguiente ejemplo usa los comandos estándares de Unix *echo* y *cat* para demostrar el comportamiento que se espera para */dev/buffer*. 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 "los 4 puntos" > /dev/buffer** $ *(1)*			
	$ **cat < /dev/buffer** los 4 puntos $ *(2)*		
		$ **cat < /dev/buffer** *(3, espera)*	
$ **echo "cardinales son 3:" > /dev/buffer** $		cardinales son 3: $ *(4)*	
$ **echo "el norte" > /dev/buffer** $ **echo -n "y el" > /dev/buffer** $ **echo "sur" > /dev/buffer** $ **echo "." > /dev/buffer** *(5, espera)*			
$ *(6)*			$ **cat < /dev/buffer** el norte $
	$ **cat < /dev/buffer** y el$ **cat < /dev/buffer** sur $ **cat < /dev/buffer** . $ *(7)*		
			$ **cat < /dev/buffer** *(8)*
			control-C $ *(9)*

Notas:

- En (1) el comando *echo* invoca un solo *write* para escribir un texto en */dev/buffer*. Esto equivale a la operación *put*. Por simplicidad suponga que cada apertura en escritura llevará un solo *write* asociado.
- En (2) *cat* pide leer 8192 bytes. Esto equivale a la operación *get*. Ud. debe entregar los 13 bytes de "los 4 puntos\n". El proceso *cat* intentará leer una segunda vez. Ud. debe retornar 0 bytes para que *cat* lo interprete como fin de archivo y termine. De otro modo no se mostrará el prompt $ como se pide en el ejemplo. Note que en la primera lectura **f_pos* es 0 y en la segunda es distinto de 0.

- En (3) *cat* pide leer otros 8192 bytes, pero el buffer está vació y por lo tanto espera el próximo *write*.
- En (4) se escribe en el buffer, lo que desbloquea al shell 3.
- En (5) el buffer está vacío. Se escriben 4 elementos en el buffer. Al escribir el cuarto, el buffer está lleno y por lo tanto el *write* debe esperar (recuerde que el buffer permite almacenar solo 3 escrituras). La opción -n de *echo* hace que no se escriba el *newline* al final. Por eso en la correspondiente extracción el texto "y el", el prompt para el siguiente comando aparecen en la misma línea de "y el". Esto es para recalcar que no es el *newline* el que termina un elemento en el buffer si no que la cantidad de bytes que se escriben con *write*.
- En (6) se extrae un elemento por lo que el *write* pendiente puede continuar. El *echo* pendiente del shell 1 termina.
- En (7) se extraen todos los elementos restantes hasta vaciar el buffer.
- En (8) el buffer está vacío y por lo tanto *cat* espera.
- En (9) el usuario ingresa *control-C* para abortar la lectura. Se debe obtener nuevamente el *prompt* $.

Recursos

Baje de U-cursos el archivo *modules2019-1.tgz*. Descomprímalo con el comando:

```
$ tar zxvf modules2019-1.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. Programe su solución en el archivo *buffer-impl.c* del directorio *Buffer*. Ahí encontrará un Makefile para compilar su tarea, las instrucciones para crear el dispositivo */dev/buffer* y la implementación de monitores a partir de los semáforos del núcleo de Linux.

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.

Plazo de entrega

La tarea se entrega *funcionando* en U-cursos. Para ello entregue solo el archivo *buffer-impl.c* que implementa el driver pedido. No entregue archivos binarios. Se descontará medio punto por día de atraso, excepto días sábado, domingo o festivos. Intente resolver la tarea antes del control 3, le servirá de estudio.