CC4302 Sistemas Operativos

Tarea 7 – Semestre Otoño 2024 – Profs.: Mateu, Torrealba, Arenas

En esta última tarea Ud. deberá programar un driver para Linux que implemente *pipes* que servirán para comunicar parejas de procesos pesados. Un pipe permite que un proceso que abrió el dispositivo */dev/disco* en modo escritura envíe datos a otro proceso que abrió */dev/disco* en modo lectura. Si hay 3 procesos escritores y 3 procesos lectores se formas 3 parejas, cada pareja con un proceso escritor y otro lector comunicados mediante un pipe unidireccional.

Cuando un proceso abre */dev/disco* ya sea para leer o escribir y no hay otro proceso que le sirva de pareja, entonces espera. Cuando hay múltiples procesos en espera todos deben ser escritores o todos lectores, pero Ud. no puede hacer esperar escritores y lectores al mismo tiempo. O sea que si llega un lector y hay escritores esperando, Ud. debe escoger una pareja para ese lector (cualquiera). De la misma manera si llega un escritor y hay lectores esperando, Ud. debe escoger una pareja para ese escritor. Este problema está inspirado en el problema de las parejas de baile que se divierten en una discoteca. Ud. lo estudió en clase auxiliar de programación de software de sistemas.

El siguiente ejemplo usa el comando estándar de Unix *cat* para demostrar el comportamiento esperado de */dev/disco*. 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 *open*, *read* o *write*).

Shell 1	***Shell 2***	***Shell 3***	***Shell 4***	***Shell 5***	***Shell 6***
$ **cat > /dev/disco** (1) **hola**					
	$ **cat < /dev/disco** (2) hola				
			$ **cat < /dev/disco** (3)		
		$ **cat > /dev/disco** (4) **hi** hi			
que tal (5)	que tal				
				$ **cat > /dev/disco** (1) **bonjour**	
					$ **cat < /dev/disco** (2) bonjour
<control-D> (6) $	$			**ca va** (5)	ca va
		<control-C> (7) $	$		***<control-C>*** (8) $
				<control-D> **$**	
$ **cat > /dev/disco** ***<control-C>*** (9) $					

Notas:

(1) Se invoca el comando *cat* en el shell 1 para escribir en */dev/disco*. Al abrir con *open* el dispositivo, *open* queda en espera hasta que llegue un proceso lector.

(2) Llega un proceso lector. Al abrir con *open* el dispositivo, *open* debe establecer una pareja con el proceso en espera. Se debe crear un pipe que permita que el *cat* escritor envíe datos al *cat* lector.

(3) Similar a (1), solo que es el proceso lector el que queda en espera del proceso escritor.

(4) Llega el proceso escritor.

(5) El pipe que comunica los procesos en los shells 1 y 2 es independiente del pipe para los procesos de los shells 3 y 4 (y después 5 y 6).

(6) El proceso escritor cierra el pipe. Esto hace que se invoque *release* del *file descriptor* del proceso escritor, pero no en el proceso lector. Esto debe cerrar el pipe. Es decir cualquier lectura pendiente en el proceso lector debe retornar 0 bytes para indicarle a *cat* que termine.

(7) Se termina el proceso escritor con *control-C*, lo que hace se invoque *release* del *file descriptor* del proceso escritor. Esto es igual que el punto (6).

(8) El proceso lector recibe la señal SIGINT mientras está en *read* (en el drive de */dev/disco*). La función *read* debe retornar -EINTR, lo que hará que el *cat* lector termine, pero no el *cat* escritor. Este se termina con *control-D*.

(9) Un proceso en espera en *open* recibe la señal SIGINT. Esta función debe retornar -EINTR, lo que hará que el *cat* escritor (o lector) termine.

Recursos

Descargue de U-cursos el archivo *modules2020-2.tgz*, publicado en novedades del 14 de junio. 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 (no lo necesita si estudió las clases sobre módulos). Le será de especial utilidad el directorio *Syncread* con el enunciado y la solución de la tarea 3 del semetre otoño de 2013 (que se vio o se verá en clase auxiliar) y *Pipe* con el enunciado y la solución de la tarea 3 de un semestre anterior que corresponde a un pipe compartido entre todos los procesos. Es el problema que se estudió en cátedra.

Descargue además el archivo *t7.zip* y descomprímalo. En ese directorio encontrará un *Makefile* para compilar su tarea y un archivo *README.txt* con las instrucciones para crear el dispositivo */dev/disco*. Cuidado: para que pueda cargar el archivo *.ko* como módulo en el núcleo, el archivo debe situarse en un directorio local de Linux. También es importante que en el camino de directorios hasta llegar a la raíz, no deben haber nombres con espacios en blanco.

Ayuda

Resuelva su tarea usando los mutex y condiciones incluidos en el directorio *T7*. Como ejemplo de utilización estudie las soluciones de *Syncread* y *Pipe*. 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*.

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.

Observe que en el archivo *Pipe/pipe-impl.c* se implementa el pipe, con variables globales para el mutex, la condición, el arreglo de caracteres, las variables *in*, *out* y *size*. Como en su tarea debe utilizar múltiples pipes, defina una estructura de datos para representar el pipe y coloque ahí todas estas variables globales como campos en la estructura. Así cada pipe será independiente. Agregue un campo adicional para señalar cuando se cerró el pipe por el lado del escritor.

En el *open* Ud. puede distinguir entre lectura y escritura de la misma manera que en Syncread. Use un mutex y una condición para esperar el *open* que establece la pareja. El problema de sincronización es el mismo de las parejas de baile en una discoteca. Cree la estructura que representa el pipe y almacene su dirección en el campo *filp→private_data* (tipo void*). Este campo está ahí para que Ud. lo use para lo que estime conveniente. En *read* y *write* rescate el pipe accediento a ese mismo campo.

Entrega

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