Carreras criticas en el problema del productor-consumidor

Sergio Souto Mourelle, Inés Vilariño Lema

1. Información para la ejecucion de los codigos

Lo dos códigos necesarios para la demostración llevan de nombre "cons_critico.c" y "prod_critico.c". Para su compilacion solo es necesario usar gcc, sin la necesidad de argumentos específicos. Para su ejecución ambos ejecutables tienen que estar en el mismo directorio y ambos tiene que ser ejecutados en terminales distintos para poder observar el programa y los efectos delas carreras criticas. Es necesario ejecutar primero el productor, y es recomendable dejar que produzca aproximadamente 4 o 5 items antes de ejecutor el consumidor para poder apreciar la carrera critica mas facilmente.

2. Explicacion del codigo

Los codigos entregados recrean artificialmente el problema del productor-consumidor, donde dos procesos distintos tratan de leer y escribir en un mismo archivo al mismo tiempo, produciendose carreras criticas donde un proceso lee o escribe valores incorrectos. El productor inserta items en un buffer, mientras el consumidor los lee y los elimina.

Memoria compartida

Los codigos usan un mismo archivo auxiliar que abren mapeando su memoria para poder compartir y actualizarlo a la vez. Dentro de este se guarda la cuenta de items en el buffer, el propio buffer, y dos variables designadas para que cada uno de los procesos pueda despertar al otro en caso de necesitarlo. Esta memoria tiene que cerrarse correctamente una vez se acaben los procesos para asegurarse de que no hay fugas de memoria.

Flags de despertar

Ambos programas tienen cada uno una variable designada y compartida que les indica cuando tienen que actuar. Cuando el buffer esta vacio se duerme el consumidor, que se despierta al meterse el primer item. En el caso de que el buffer este lleno, el productor se duerme y espera a que el consumidor borre un item para que este le despierte. Ambas variables son controladas por ambos, y usan su propia variable para dormirse a si mismos y la del otro para despertarlo.

Carreras criticas

Si interrumpimos al consumidor o al productor después de la lectura y antes de la escritura, y en su lugar se ejecuta el otro, pueden ocurrir problemas como que se elimine un item incorrecto, se salga de rango un indice o que se escriban items en lugares erroneos.

En nuestro codigo se fuerza la aparicion de carreras criticas poniendo un sleep() entre la lectura del numero de items en el buffer y la escritura sobre este de cada uno de los archivos. Al hacer esto, es muy probable que, por ejemplo, el productor escriba un item nuevo mientras el consumidor intente borrar uno anterior, actualice el buffer con una cuenta equivocada, y luego la vuelva a leer inmediatamente antes de hacer el sleep(), formandose asi una desincronizacion entre ambos procesos, donde el consumidor siga eliminando los items ya existentes previamente en vez de eliminar el ultimo que haya puesto el productor, y viceversa, ya que el productor seguira produciendo items sin tener en cuenta los que elimine el consumidor, ya que al salir del sleep, sigue usando el numero de items previo, el cual el mismo acaba de escribir.

Si el consumidor consume más rapido que el productor y vacia el buffer, este se va a dormir, pero al despertarse por el productor este sigue con el mismo valor que leyo anteriormente, lo que da un error por pasarse de indices y da un error de segmento

Conclusion

Podemos sacar de este experimento que dejar a la suerte y sin control el orden en el que se ejecutan distintos procesos que acceden a los mismos datos es generalmente una mala idea, ya que no tenemos ninguna garantía de que no se produzcan carreras criticas y acabemos con datos erroneos o fallos en los programas.