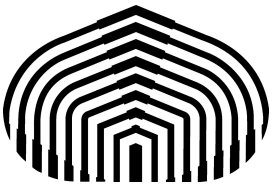

Universidad Simón Bolívar

Sistemas Operativos I

Proyecto I

Historia de un País

Nicolás Jaua | 15-10732

Marcos Lerones | 15-10778

Sartenejas

23 de Mayo de 2019

Introducción

En este proyecto se nos pidió simular un sistema gubernamental de 3 poderes públicos (Ejecutivo, Legislativo y Judicial) mediante el uso de procesos e hilos en el lenguaje de programación C. La intención es que dichos procesos se comuniquen entre sí y con otro proceso llamado Prensa que es el que se encarga de imprimir en pantalla todo lo que está pasando en la simulación.

Nos vimos obligados a cambiar algunos detalles del enunciado para que el problema fuera más manejable y funcional. Estas simplificaciones fueron 2 en específico:

1) Quitar los días de descanso: Esto lo hicimos simplemente porque dejaba la posibilidad de que el programa entrara en un ciclo infinito ya que si los tres procesos caen a la vez en un descanso, no hay forma de saber el paso del tiempo (ya que se mide en titulares impresos por la prensa) y como el programa solo debe terminar después de una cantidad de días especificada por el usuario, si no pasan los días, el programa nunca termina. En lugar de esto, lo que se hizo fue que si uno de los poderes no realizó ninguna acción después de leer todo su plan de gobierno, este le manda a la prensa un titular que dice que se tomó un dia de receso.
2) Eliminar los ministros y magistrados: Esto le agregaba un nivel de dificultad increíble, ya que se tenían que mantener estructuras para saber cuántos (y cuáles) ministros existen y en particular, en la implementación con procesos, trae muchos problemas con respecto al manejo de señales ya que es difícil para un proceso manejar tantas señales complejas.

Para correr el programa se debe compilar con el comando `make` y para ejecutarlo, se debe escribir en el terminal

```
./paisProcesos <duracion> [directorio]
./paisHilos <duracion> [directorio]
```

dónde duración es la duración en días de la simulación y directorio es el directorio donde se encuentran los archivos de gobierno. Si no se incluye el campo directorio, se buscan los archivos de gobierno en el directorio actual. Los archivos `Judicial.acc`, `Legislativo.acc` y `Ejecutivo.acc` deben estar en el directorio a utilizar, y estos deben

cumplir las especificaciones dadas en el enunciado del proyecto para el buen funcionamiento del programa.

Implementación

Procesos:

En esta implementación, la prensa es el proceso principal, el que empieza todo. La prensa se encarga de crear los pipes nominales para las futuras comunicaciones, un espacio de memoria compartida para almacenar las estadísticas de cada uno de los poderes y crear los 3 poderes públicos usando fork() y execvp(). Después de crear los 3 procesos nuevos, prensa le escribe a cada uno de ellos a través de un pipe, los PID's de los 3 procesos para que puedan comunicarse entre sí. Luego la prensa espera (mediante un semáforo) a que los 3 procesos establezcan su manejador de señales para continuar, ya que si no se hace esto, se pueden perder señales de aprobación entre poderes. Lo siguiente que hace la prensa es entrar en un loop que itera hasta que pasen los días especificados en los argumentos del programa y dentro del loop, espera a recibir un titular a través de un pipe, luego lo imprime, incrementa la cantidad de días pasados y vuelve al inicio.

Los procesos que representan los poderes tienen comportamientos similares. Primero se encargan de obtener todas los elementos que necesitan para funcionar correctamente (inicializar semáforos, obtener PID's, obtener directorio, establecer manejador de señales) y luego le informan a la prensa que están listos y entran en un ciclo infinito. En el ciclo lo que hacen es leer el archivo que contiene su plan de gobierno, escoger una acción, guardarla como un arreglo de strings, luego ejecutar esa acción y, dependiendo si la acción fue exitosa o no, escribe uno u otro titular al pipe de la prensa y actualiza sus estadísticas.

Para sincronizar las escrituras en el pipe de la prensa utilizamos un flock (file lock) y un semáforo. La estrategia es que el semáforo es inicializado en 0, la prensa, al intentar leer del pipe, se bloquea hasta que haya un mensaje disponible (por la naturaleza de read()) y luego de leer, hace sem_post() al semáforo. Mientras tanto, los escritores justo antes de escribir adquieren un lock exclusivo para el pipe (que funciona como un mutex), escribe en el pipe, utiliza sem_wait() en el semáforo y por último, libera el lock. El objetivo de esto es que el escritor no libere el lock del archivo hasta que la prensa haya leído su mensaje, con esto nos aseguramos que la prensa al leer, solo leerá un titular por dia y que los titulares de distintos procesos no se mezclarán.

Para elegir una acción del plan de gobierno, este se abre con un lock compartido, y se lee el nombre de la acción. Con una probabilidad de 0.2, decidimos si se escoge esa acción o no. En caso de ser escogida se lee hasta una línea vacía y se guarda cada línea leída en una posición de un arreglo de strings. En caso de no ser escogida, se lee hasta una línea vacía sin guardar y se repite el ciclo con la nueva acción que acabamos de encontrar o, si no hay más acciones disponibles, se manda un mensaje a la prensa de Iddleness y no se realiza ninguna acción ese día.

Para ejecutar la acción realizamos una función que recibe un arreglo de strings que representa la acción a realizar, el número de líneas que conforman a la acción, el directorio donde se encuentran los recursos de gobierno y los PID's de los 3 procesos. la función inicializa una variable de tipo FILE *, que indica el archivo que está abierto en ese momento (comienza siendo igual a NULL), y una int llamado success que será el valor retornado por la función. La función itera por las instrucciones de la acción secuencialmente y para cada acción separa la primera palabra del resto (para ver cual es el comando especificado) y luego realiza la instrucción indicada. Para escrituras, solo se escribe al final del archivo abierto (o devuelve 0 si ocurre un error). Si se pide leer o anular, se busca la línea especificada en el archivo abierto, y se retorna si es encontrada o no, y en caso de ser contradictoria la respuesta (si para anular o no para leer) se le asigna 0 a success y se sale del ciclo. Para abrir archivos lo primero que se hace es, si existe un archivo abierto, se libera el lock que se tenga del mismo y se cierra el archivo, luego, se abre el nuevo archivo y se coloca el lock necesario (exclusivo o inclusivo). Podemos ver que de esta forma podemos evitar cualquier riesgo de deadlock con archivos ya que cada proceso tiene a lo sumo 1 archivo abierto a la vez, por lo tanto, no puede existir espera circular.

Por último, falta explicar cómo se manejan las aprobaciones. Para esto, cada proceso establece en su manejador de señales una señal que le indica que alguno de los otros dos procesos está pidiendo su aprobación. Se usan las señales SIGUSR1 y SIGUSR2 para este fin. Por ejemplo, si el proceso ejecutivo recibe una señal SIGUSR1, entonces sabe que el proceso legislativo requiere de su aprobación. El manejador lo que hace es escoger aleatoriamente una respuesta (si o no) y escribirla en el pipe que comunica a los dos procesos en esa dirección. El proceso que pide aprobación simplemente tiene que mandar la señal apropiada al proceso que desea y esperar a que este responda a través del pipe.

Las instrucciones de disolver, censurar, destituir, nombrar y asignar no fueron implementadas ya que no se implementaron ministros ni magistrados. El presidente se encarga de todas las aprobaciones que le deberían corresponder a algún ministro.

Hilos:

En este programa, como en el de procesos, se implementa el hilo de la prensa como el hilo principal, el cual crea los hilos de los poderes ejecutivo, legislativo y judicial. Se utiliza la estructura *threadArguments* pasandola como un apuntador a los distintos hilos para tener acceso desde cada uno a distinta información y poder comunicarse entre ellos. *threadArguments* contiene un arreglo de 10 strings para comunicar los titulares formando una cola circular para que prensa los lea y los hilos no tengan que esperar a que otro termine de escribir y prensa termine de leer. Usa semáforos simulando el problema de productores-consumidores para que dos hilos no escriban sobre un mismo string y para que prensa no lea mientras un hilo escribe. Prensa espera a que el semáforo que le indica que ha recibido un nuevo titular para poder imprimirlo, y cuenta los días hasta alcanzar el número máximo de días dado al comienzo del programa. Luego de esto imprime el número de acciones exitosas de cada poder y el porcentaje de las acciones exitosas sobre la cantidad de acciones que trato de pasar ese poder.

Los hilos para los poderes ejecutivo, legislativo y judicial lo primero que realizan es crear un segundo hilo cada uno que se encarga de recibir aprobar o reprobrar permisos pedidos por otros poderes. Usando los *threadArguments* al que tienen acceso todos los procesos, crea una estructura *secretaryArguments* que contiene apuntadores a enteros de threadArguments y semáforos. Usando estos y las funciones *pthread_cond_wait*, *pthread_mutex_lock*, *pthread_cond_broadcast* y *pthread_mutex_unlock*, el hilo secretary se encarga de esperar a recibir una solicitud de aprobación para responderle al poder solicitante.
Luego, cada poder utiliza un while donde espera a que end en *threadArguments* se vuelva cero, cosa que haría el hilo de prensa. Cada ciclo del while suma uno en stats para representar el número de acciones que intenta realizar. Utiliza la función *readAction*, como se explicó en la sección de procesos, para conseguir una acción completa. Opera sobre readAction de igual forma que la versión del programa que usa procesos, excepto en el caso de las aprobaciones, en las que usa una función distinta. Dependiendo de a cual poder le esté solicitando aprobación utiliza los enteros *execReq* y *execAns*, *judReq* y *judAns* o *legisReq* y *legisAns*

asignandeselos a *req* y *ans* de secretaryArguments. Luego la función *approvalRequest* se encarga de avisar, utilizando estas variables y los mutex dados en *threadArguments*, al *secretary* del poder del que requiere aprobación. Este utiliza un random para dar su aprobación la cual luego le envía en ans a *approvalRequest*.

Resultados

Procesos:

Primera corrida:

```
Poder Ejecutivo   : 13 acciones exitosas (44.83% de exito)

Poder Legislativo : 7 acciones exitosas (35.00% de exito)

Poder Judicial    : 33 acciones exitosas (62.26% de exito)
```

Segunda corrida:

```
Poder Ejecutivo   : 13 acciones exitosas (54.17% de exito)

Poder Legislativo : 10 acciones exitosas (47.62% de exito)

Poder Judicial    : 34 acciones exitosas (59.65% de exito)
```

Tercera corrida:

```
Poder Ejecutivo   : 15 acciones exitosas (50.00% de exito)

Poder Legislativo : 10 acciones exitosas (45.45% de exito)

Poder Judicial    : 22 acciones exitosas (43.14% de exito)
```

Hilos:

Primera corrida:

```
Poder Ejecutivo   : 6 acciones exitosas (35.29% de exito)

Poder Legislativo : 4 acciones exitosas (19.05% de exito)
```

```
Poder Judicial    : 33 acciones exitosas (51.56% de exito)
```

Segunda corrida:

```
Poder Ejecutivo   : 11 acciones exitosas (45.83% de exito)

Poder Legislativo : 14 acciones exitosas (42.42% de exito)

Poder Judicial    : 28 acciones exitosas (62.22% de exito)
```

Tercera corrida:

```
Poder Ejecutivo   : 9 acciones exitosas (33.33% de exito)

Poder Legislativo : 12 acciones exitosas (44.44% de exito)

Poder Judicial    : 26 acciones exitosas (54.17% de exito)
```

Comparación

Tiempo procesos:

```
real       0m0, 007s
user       0m0, 003s
sys        0m0, 010s
```

Tiempo hilos:

```
real       0m0, 003s
user       0m0, 004s
sys        0m0, 002s
```

Con estos resultados podemos ver que el programa usando hilos es más veloz que el que usa procesos. Esto se debe a varias razones. Una es el uso de *secretary* en hilos, que permite seguir con pedir aprobaciones sin interrumpir el hilo principal de los poderes, mientras que el programa que usa procesos utiliza señales para pedir aprobaciones y el manejador de señales interrumpe al poder que se estaba ejecutando. También está el hecho de que el programa con

hilos utiliza memoria común entre lo hilos para comunicarse, mientras que el programa que usa procesos requiere de pipes, en los que tiene que escribir y leer desde disco, lo que conlleva más tiempo. La implementación de una cola circular en hilos para pasar los titulares a prensa permite que los hilos no se tengan que turnar entre escribir y leer, sino que varios procesos distintos pueden ir escribiendole su titular a prensa en strings distintos y prensa los pueda leer a medida que le van llegando en el orden en el que los recibe. Estos factores permiten un funcionamiento más rápido del programa usando hilos, que resultan del beneficio de tener memoria común y poder pasarle argumentos a los hilos al ser creados.

Conclusiones

Con este proyecto podemos concluir que trabajar con hilos es mucho más eficiente y (en general) más fácil de manejar a la hora de escribir programas paralelos. Esto se debe a que puedes compartir el mismo espacio de memoria y no existe la necesidad de crear pipes y manejadores de señales. Además, la programación con hilos te da herramientas mucho más poderosas como las variables condicionales, que te facilitan el trabajo como programador y te abren las puertas para pensar los problemas de sincronización desde otro enfoque.

Sin embargo, para la comunicación asíncrona, los hilos no son la mejor opción ya que es complicado manejar las señales desde hilos. Aún así, se puede ingeniar una forma de usar las herramientas que proveen los hilos para simular una comunicación asíncrona (así no esté pasando realmente) y dado que en procesos es complicado utilizar bien las señales a tiempo real. recomendamos que en general, si hay mucha comunicación, se debería implementar con hilos y que se usen procesos para programas que no tengan que interactuar tanto entre sí.