Ejercicios parciales

Ejercicio 1) El algoritmo de elección de *token ring* es un mecanismo descentralizado que se utiliza para decidir un coordinador entre un conjunto de procesos. A continuación presentamos una variación de este algoritmo.

Inicialmente, se organizan N procesos identificados del 0 al $N-1$ en una estructura de anillo, de forma tal que cada uno tiene un único antecesor y un único sucesor. A la vez, cada proceso tiene un número mágico asociado, que lo calcula llamando a la función `int numeroMagico()`. Una vez armado el anillo, el proceso 0, llamado iniciador, envía un mensaje a su sucesor que consiste en un arreglo de dos enteros: su número de proceso (en este caso 0) y su número mágico. Este mensaje es recibido por el siguiente proceso del anillo, que verifica si su propio número mágico es mayor que el que recibió de su antecesor. Si lo es, entonces envía al proceso siguiente su propio arreglo (con su número de proceso y su número mágico); si no lo es, envía el mismo arreglo que recibió de su antecesor. Esta secuencia es repetida hasta que el proceso iniciador recibe el último mensaje del anillo. Una vez que se recorrió todo el anillo, el número de proceso que recibió el iniciador corresponde al proceso elegido como coordinador. Es decir, el proceso coordinador acaba siendo el que tiene mayor número mágico. Finalmente, el iniciador envía un mensaje (que también debe recorrer todo el anillo), indicando cuál es el proceso coordinador.

Se pide implementar un programa que cree N procesos y utilice el algoritmo anterior para elegir un proceso coordinador entre el conjunto de procesos. Una vez finalizada la elección, el resto de los procesos debe ejecutar la función `double operacionComplicada()` y enviar su resultado al coordinador, que debe imprimirlos en pantalla. Finalmente, todos los procesos deben terminar.

Tener en cuenta que:

- Los procesos deben comunicarse entre sí utilizando pipes.
- Puede asumir que posee las funciones `int anterior(int i)` e `int siguiente(int i)`, que dado un `i` entre 0 y $N-1$ devuelven cuál es el `i` anterior o posterior del anillo respectivamente.
- Puede asumir que, al escribir varios procesos en un mismo pipe, sus datos no se mezclarán.
- Se deben cerrar todos los pipes que no se utilicen.

Ejercicio 2) Se tienen los siguientes tres procesos: `curly`, `larry` y `moe`, que son ejecutados concurrentemente. Además, estos tres procesos comparten los semáforos S, R y T, todos inicializados en 1, y una variable global x, inicializada en 0.

```
void curly( ) {
    do {
        semWait(S);
        semWait(R);
        x++;
        semSignal(S);
        semSignal(R);
    } while (1);
}
```

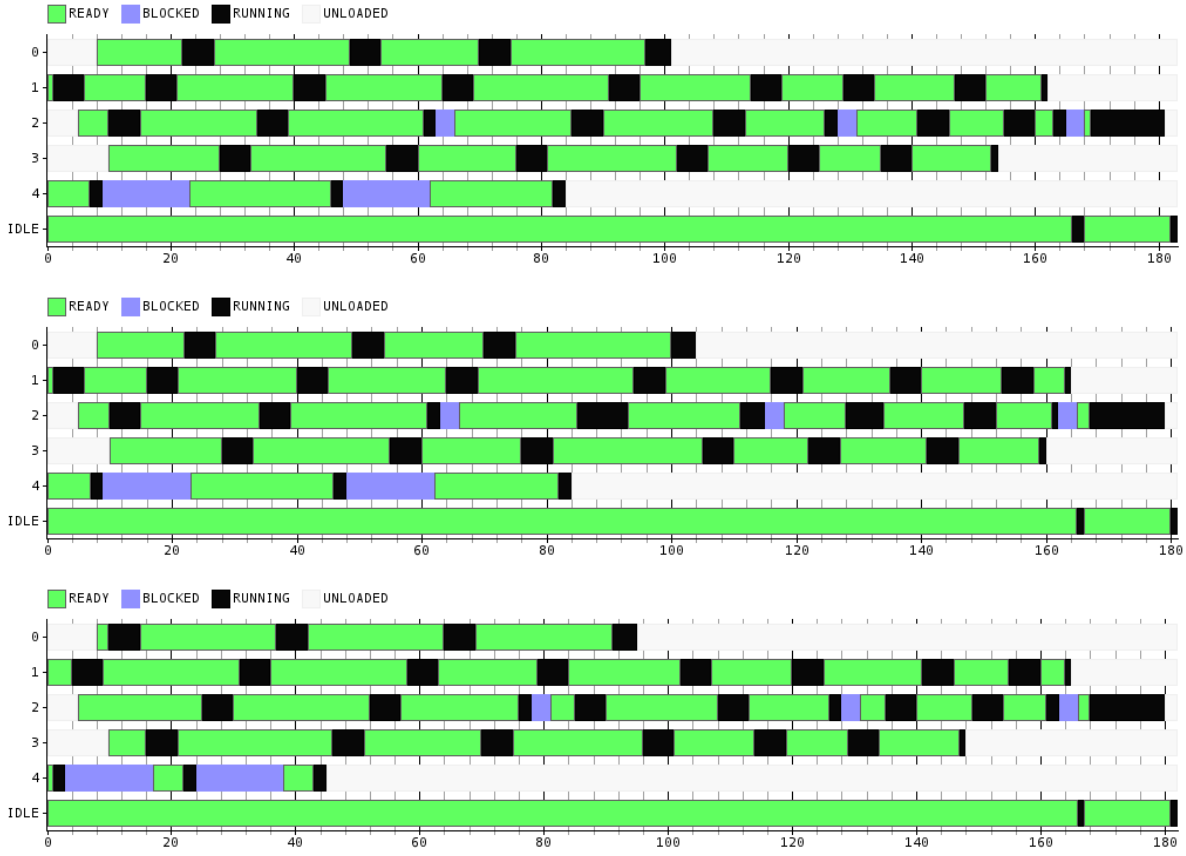
```
void larry( ) {
    do {
        semWait(R);
        semWait(T);
        x--;
        semSignal(R);
        semSignal(T);
    } while (1);
}
```

```
void moe( ) {
    do {
        semWait(R);
        semWait(T);
        semWait(S);
        x--;
        semSignal(R);
        semSignal(T);
        semSignal(S);
    } while (1);
}
```

a) ¿Se puede decir que el sistema está libre de deadlock? Justifique.

b) ¿Se puede afirmar que el sistema está libre de inanición? Justifique.
c) ¿Se le ocurre un método genérico para evitar el deadlock en casos como los del ejemplo?

Ejercicio 3) Los siguientes diagramas de Gantt muestran los tiempos de ejecución del mismo lote de tareas para un conocido algoritmo de scheduling. El segundo y tercer diagrama corresponde a una variante del algoritmo del primer diagrama. Los tres diagramas usan los mismos parámetros, la diferencia está en variaciones en el algoritmo de scheduling.



Explicar y justificar:

- ¿En qué consisten las variantes?
- ¿Cuál de los tres elegiría para el caso general? ¿Cuáles son las ventaja/s y desventaja/s de las variantes? Indique al menos dos ejemplos concretos: uno que aproveche las ventajas de la variante y otro donde la utilización de la variante sea contraproducente.
- Especifique qué proceso es el que le indicó en que consiste la variantes y en que intervalo de tiempo se manifiesta.

Soluciones

Ejercicio 1) Pueden descargar la solución de: `https://gist.github.com/sromano/de661f2745d5bff37afc`

Ejercicio 2)

a) No, el sistema no está libre de deadlock.

Veamos las condiciones de Coffman.

Exclusión mutua: Los semáforos están inicializados en 1 y, por lo tanto, sólo pueden recibir un wait de un sólo proceso hasta que este haga el correspondiente signal.

Hold and wait: Los procesos piden otro semáforo después de haber pedido el primero.

No preemption: No se describe ningún mecanismo para quitarle los semáforos a un proceso.

Espera circular: Ver la siguiente ejecución de los procesos

1. Se ejecuta en curly semWait(S) y adquiere S
2. Se ejecuta en moe semWait(R) y adquiere R
3. Se ejecuta en moe semWait(T) y adquiere T

Ahora curly quiere el semaforo R, que lo tiene moe, y moe quiere el semaforo S, que tiene curly. Por lo tanto, el sistema entró en Deadlock.

b) Que el sistema este libre de inanición o no depende del scheduler que utilice el sistema operativo. Si el scheduler cicla una cantidad finita de veces entre larry y curly, de manera que siempre alguno de los dos tenga el lock R, entonces moe nunca va a ejecutar (a pesar de que se le asigna quantum).

Esto se puede hacer constructivamente: Simulamos la ejecución de los procesos y cada vez que alguien tenga el lock R saltamos a moe. Mientras que cada vez que ni curly ni larry tengan el lock R saltamos entre ellos hasta que alguno lo adquiera.

En definitiva no es posible afirmar que el sistema esta libre de inanición sin garantizar la política del scheduler.

c) Una solución fácil es tomar todos los locks en el mismo orden en todos los procesos. (Lo explicamos en clase, pero van a ver esto con un poco más de detalle en Bases de Datos cuando vean el protocolo 2PL)

Ejercicio 3) La primer variante corresponde al algoritmo de Round Robin tradicional.

En la segunda variante, se le suma al quantum de la siguiente ejecución de un proceso que se había bloqueado, el quantum perdido en la corrida anterior.

En la tercera, cada vez que se libera, se lo pone primero en la lista de ready y se le otorga un nuevo quantum.

Para el caso general eligiría la primer variante, puesto que en la segunda y la tercera un proceso puede dominar a los demás a través de bloquearse en momentos estratégicos. En el caso de la segunda, acumulando mucho quantum al principio a través de numerosas llamadas bloqueantes y en el caso de la tercera bloquéandose un instante antes que se termine su quantum.