Programación Concurrente

Recuperatorio

1. [**Exclusión Mutua**] Considere la siguiente extensión del algoritmo de Dekker para n procesos (con $n > 2$) que utiliza las siguientes variables compartidas:

```
global boolean[n] flags = {false, ..., false};
global int turno = 0;
```

Además cada thread está identificado por el valor de la variable local `threadId` (que toma valores entre 0 y $n - 1$). Cada thread utiliza el siguiente protocolo.

```
// seccion no critica
otro = (threadId+1) % n;
flags[threadId] = true;
while (flags[otro])
  if (turno != threadId) {
    flags[id] = false;
    while (turno != threadId);
    flags[id] = true;
  }
// SECCION CRITICA
turno = otro;
flags[threadId] = false;
// seccion no critica
```

Mostrar que esta variación no resuelve el problema de la exclusión mutua para n procesos. Indique claramente cuál condición es violada.

2. [**Semáforos**] Se desea modelar la torre de control de un aeropuerto con una única pista. La torre otorga permiso para aterrizar y despegar a distintos aviones. Resuelva los siguientes problemas usando semáforos, modelando cada avión como un thread independiente que desea utilizar la pista. Tenga en cuenta que aterrizar y despegar no son acciones atómicas, y por lo tanto, requieren de cierto tiempo.

   a) De una solución que garantice en todo momento que el número máximo de aviones utilizando la pista es uno. Considerando que los aviones que desean aterrizar tienen prioridad por sobre los que desean despegar.

   b) Modifique la solución anterior de forma tal de evitar inanición sobre los aviones intentando despegar. La torre debe priorizar el despegue sobre el aterrizaje cada diez aterrizajes y mantener el comportamiento del punto anterior el resto del tiempo.

3. [**Monitores**] Un sistema de reservas para cines es utilizado por varias boleterías (threads). Cada boletería puede solicitar la reserva de una cantidad de asientos en la sala, o avisar que un número de asientos han quedado libres cuando un número de entradas son canceladas. La cantidad de asientos reservados nunca puede ser superior a la cantidad de asientos de la sala. Si una boletería intenta reservar una cantidad de asientos mayor a la disponible, debe quedar bloqueado hasta que la cantidad de asientos sea suficiente para efectuar la reserva. Si la reserva es exitosa, la operación `reservar()` debe retornar `true`. Asuma que ningún vendedor puede reservar ni cancelar nunca un número negativo de asientos.

   Implemente utilizando monitores:

   a) El monitor `SalaDeCine` de modo que toda boletería pueda reservar asientos mientras haya asientos disponibles en ese momento. Recuerde que puede definir variables de condición sobre las que puede efectuar únicamente las operaciones `wait()`, `notify()` y `notifyAll()`.

   b) Extienda el monitor con una nueva función `iniciarPelicula()` que se bloquee en caso que la sala no esté llena (una sala se llena cuando no hay mas asientos libres). En caso de estar llena, `iniciarPelicula()` debe desbloquearse y permitir la proyección de la función. Todas las reservas pendientes en ese momento deben retornar `false` y no efectuarse. Modificar todas las funciones del Monitor que considere necesario.

4. [**Mensajes**] Un servicio online de competencias de ajedrez permite que jugadores de todo el mundo se conecten y compitan. Se desea implementar el servidor y los clientes para los siguientes puntos usando mensajes como mecanismo de sincronización:

   a) Un servidor que apareja a los jugadores a medida que se conectan. Observe que no es necesario que el servidor interactúe con los jugadores una vez que comienza la partida, sólo es necesario que medie la conexión de los usuarios.

   b) Extienda la solución anterior de forma tal que los jugadores informan su ELO (número que indica su nivel) y el servidor arma parejas de ELO similar (con una diferencia de a lo sumo un 10 %, asuma que dispone de un operador `=%=` que hace esta comparación). Los jugadores quedan en espera mientras no se conecte otro jugador de nivel similar.