Práctica 4
Programación Concurrente y de Tiempo Real
Universidad de Cádiz

Alejandro Serrano Fernández

October 20, 2020

1 Tercer Intento

En este intento, se inicializan las variables C1 y C2 a true para que indicar que los procesos desean entrar en la región crítica. Después comprueba si el otro proceso no quiere entrar en su sección región crítica. Si lo desea, será necesario esperar, en caso contrario, entrará en la sección crítica. Al salir de ella, el proceso establecerá su variable a false, permitiendo al otro proceso que entre en su sección crítica.

Observamos que en este intento satisface el requerimiento de la exclusión mutua.

```
Proceso 1:{
for(int i=0; i<nVueltas; i++){
        C1 = true;
        while(C2==true);
        n++;
        C1 = false;
     }
 break;
}

Proceso 2: {
for(int i=0; i<nVueltas;i++){
        C2 = true;
        while(C1==true);
        n--;
        C2 = false;
    }
}
     break;
```

Sin embargo si P0 y P1 realizan a la misma vez la primera asignación, nos encontramos en un bucle infinito, concretamente en la instrucción while de ambos procesos.

2 Cuarto Intento

Las variables C1 y C2 se incializan a true, para indicar que cualquiera de los dos procesos tienen el desea de entrar en su sección crítica. Cuando uno de los dos procesos comprueba que el otro proceso también lo desea, cede su turno , espera un tiempo, y luego reclama su derecho a entrar a la región crítica.

```
case 1:{
     for(int i=0; i<nVueltas; i++){
        C1 = true;
        while(C2==true)
        {
         C1 = false;
         C1 = true;
        }
        n++;
        C1 = false;
      }
      break;
}

case 2: {
   for(int i=0; i<nVueltas;i++){
         C2 = true;
         while(C1==true)
         {
            C2 = false;
            C2 = true;
         }
         n--;
         C2 = false;
   }
}break;
```

Aunque los procesos puedan quedarse de forma indefinida cediéndose el paso

mutuamente, no se llega a producir espera ilimitada. Este algoritmo, aunque nos devuelva el resultado esperado, no cumple con las propiedades de exactitud.

3 Algoritmo de Dekker

Para este ejercicio, he implementado el algoritmo de Dekker de tal manera que a través de dos procesos vamos modificando la variable n. Un proceso se dedica a incrementarla y otro se dedica a decrementarla, de tal manera que el resultado teórico final es 0. En la ejecución del programa, tras varias ejecuciones, verificamos que la cumple. Con este algoritmo aseguramos que se cumple la exclusión mutua, pues si un proceso se encuentra en la sección crítica, otros procesos no han podido pasar de bucle while más externos.
También aseguramos la condición de progreso en la ejecución, ya que si por ejemplo p1 quiere entrar y p0 no, C0 = false y por lo tanto p1 puede entrar.
Y finalmente, satisface también las limitación en la espera, puesto que cuando un proceso sale de la sección crítica indica que no quiere entrar y le cede el turno a otro.

4 Algoritmo de Eisenberg-McGuire

Para la ejecución de este algoritmo utilizaré la misma sección crítica que en el apartado anterior. En mi caso, con este algoritmo, tras varias ejecuciones, se cumple siempre el mismo resultado.
El algoritmo utiliza las siguientes estructuras de datos:

·Indicador: int[0...n-1], cuyos elementos son del tipo enumerado, cons tres posibles valores:
-Ocioso: No desea entrar en la sección crítica.
-Esperando: Desea entrar en la sección crítica.
-Ejecutando: se encuentra ejecutando la sección crítica.
·Indice: indica que proceso puede entrar en la sección crítica.

Primero se comprueba que entra el proceso que tenga permiso y los demás estén esperando. Si alguno no está esperando, vuelve a comprobar desde el principio. Si hay algún proceso en la sección crítica, espera. Esto se repite hasta que ningún proceso esté ne la sección crítica e id tenga el turno o el que lo tenga, esté en estado ocioso. Posteriormente se da turno al primer proceso que quiera entrar, si ninguno quiere, se queda con el turno.

5 Algoritmo de Lamport

Este algoritmo es conocido como el algoritmo de la panadería. Podemos observar que para este algoritmo se cumple/verifica la exclusión mutua cuando un proceso está en su sección crítica, evitando así que otros pasen a su sección crítica e

interfiera en la solución final. Cabe destacar que se garantiza que los procesos se ejecutan de forma lineal (P0,P1...).
En mi caso, para comprobar que se cumple la condición de exclusión mutua, he utilizado la misma condición que los ejercicios anteriores (incrementos de 10000 y decrementos de 10000), y el resultado obtenido tras varias ejecuciones, sigue siendo el mismo, es decir, 0.

En cuanto a su funcionamiento, cada proceso recibe un número. El proceso con el número más bajo es el primero en ser ejecutado.
Con el algoritmo de Lamport se verifica la exclusión mutua, se garantiza el progreso en la ejecución y asegura la limitación en la espera.

6 Algoritmo de Hyman

Cuando ejecutamos ambos procesos, podemos observar que ambos entran simultáneamente en la sección crítica, de tal manera que acaba bloqueando el programa, pues ninguno puede avanzar en su ejecución.
Es decir, turno = 0, el proceso 1 modifica C1 a true, y encuentra que C0 = true y se para. Luego, P0 hace C0 = true, encuentra que turno = 0 y entra en la sección crítica. Luego P0 hace turno = 1 y entra también en la sección crítica, dando lugar a una solución incorrecta.
Este algoritmo se presentó como una solución a la exclusión mutua, un desafío para los investigadores desde los años 60, aunque solo se presentó con intereses educacionales, ya que ayuda a clarificar las cualidades de una solución correcta.