Universidad Europea

Universidad Europea de Madrid

Escuela de Arquitectura, Ingeniería y Diseño

Programación concurrente y distribuida

Curso académico 2020-2021

Memoria Algoritmo Panadería

GRUPO 1

Universidad Europea

INTEGRANTES DEL GRUPO

Nombre	Repositorio GitHub
Andrés Ayala	https://github.com/Cabanes476/PCD.git
Aitor Pérez	https://github.com/aitorp99/21954146AG1.git
Alvaro Martínez	https://github.com/AlvaroMartinezSilva/21928787Concurrente
Miguel Wang	https://github.com/miguelwang125/PCD---AG1.git
Ignacio Lucas Alfonso	https://github.com/ilucasalfonso/ProgramacionConcurrenteDistribuidaAG1
Carlos Godoy	https://github.com/carlosgodoy888/1PCD.git

Universidad Europea

SOLUCIÓN DEL PROBLEMA Y EXPLICACIÓN DEL ALGORITMO

El algoritmo de la panadería de Lamport, debe su nombre al funcionamiento típico de una panadería o cualquier tienda en general. En la que los clientes van entrando al local y obtienen un número de turno(único) que sirve para que el panadero pueda ir atendiendo en orden de llegada.

Este dependiente sólo puede ir atendiendo a un cliente al mismo tiempo. El dependiente es análogo al recurso, cada cliente es un hilo y mientras este hace uso del recurso es la sección crítica.

El sistema tiene una variable global que establece el turno en el que se encuentra. Los hilos deben esperar en una cola hasta que les llegue el turno. Una vez acabe la sección crítica de un hilo se incrementa en uno la variable global y el siguiente hilo accede al recurso. Si el hilo que sale de la sección crítica desea volver a entrar en ella, deberá volver a coger un turno y colocarse en la última posición de la cola.

PROCESO PARA ENTRAR EN LA SECCIÓN CRÍTICA

Cuando un hilo quiere acceder a la sección crítica, primero obtiene su número de turno, que es la cantidad de turnos de los otros hilos, más uno. Si el cálculo de turnos lo hacen dos hilos a la vez, es posible que tengan el mismo turno, esto se soluciona mediante el algoritmo de desempate, comparando los identificadores y priorizando el más bajo.

Una vez el hilo hace uso del recurso y abandona la sección crítica su turno cambia de valor a uno especial que indique la intención de no volver a entrar en la sección crítica.

Dos procesos al mismo tiempo turno el mismo y al comparar etiquetas se ve cual es mayor por el número de bits pertenecientes.

Universidad Europea

CÓDIGO Y EXPLICACIÓN

```
'''libreria que nos permite manejar hilos'''
import threading
import time

'''el cliente es un hilo'''


'''tiempo de espera en cada sección'''
DELAY=1
'''Aquí podemos modificar el número de hilos de nuestro programa'''
NUM_HILOS = 2
'''calcular el número de turno'''
eligiendo=[False]*NUM_HILOS
numero= [0]*NUM_HILOS
critical=[False]*NUM_HILOS #se usa para monitorear que threads estan en la seccion critica
en un momento dado

'''Cuando un hilo quiere entrar en su sección crítica, primero obtiene su número de turno,
que calcula como el máximo de los turnos de los otros hilos, más uno.'''
def maximo():
    max = -1
    for i in numero:
        if (i>max):
            max=i

    return max

'''hace la comparacion de tuplas'''
'''
a<c : el numero de turno del hilo j es menor que el turno del hilo i
a==c: se puede dar la casualidad de que ambos hilos tengan el mismo turno para eso la
siguiente condicion
b<d : si dos o más hilos tienen el mismo número de turno, tiene más prioridad el hilo que
tenga el identificador(son unicos) con un número más bajo
'''
def comparacion_tuplas(a,b,c,d):
    if(a<c):
        return True
    elif (a==c and b<d):
        return True
    return False
```

```
def bloqueado(i):

    eligiendo[i] = True
    numero[i]= 1 + maximo()
    eligiendo[i]= False


    for j in range(NUM_HILOS):

        '''se espera a que haya elegido'''
        while eligiendo[j]:
            continue
        while ((numero[j] != 0) and (comparacion_tuplas(numero[j],j,numero[i],i))):
            continue
        '''
        (a, b) < (c, d)
        es equivalente a
        (a < c) o ((a == c) y (b < d))
        mirar comparacion_tuplas

        '''

def desbloqueado(i):
    '''esto dice que el hilo no quiere entrar a la seccion critica (mirar el segundo while de
bloqueado donde debe ser != 0 para poder entrar)'''
    numero[i]=0

def ejecutar_hilo(i):
        while(True):
                '''entra a la seccion critica'''
                bloqueado(i)
                print("Thread "+str(i)+" en seccion critica\n")
                critical[i]=True
                time.sleep(DELAY)
                critical[i]=False
                '''sale de la seccion critica'''
                desbloqueado(i)
                print("Thread "+str(i)+" fuera de seccion critica\n")
                time.sleep(DELAY)



'''****monitorea en que seccion estan los threads********
*******True: en la seccion critica************
*******False: fuera de ella**************'''
def monitorear():
        tiempo=1
        while(True):
```

```
            print ("Tiempo: "+str(tiempo))
            '''solo puede haber un True en el vector que se imprime
            porque solo hay un thread en la sección crítica a la vez'''
            print(critical)
            tiempo+=1
            time.sleep(1)
```

'''
Así podemos mostrar el nombre y el identificador del hilo en el que estamos
threading.current_thread().getName()
threading.current_thread().ident'''

```
def panaderia():
      for num_hilo in range(NUM_HILOS):
            hilo = threading.Thread(name='%s' %num_hilo, target=ejecutar_hilo,
args=(num_hilo,))
            hilo.start()


panaderia()
monitorear()
```

POSIBLES PROBLEMAS

Existen varios posibles problemas que podrían darse en este algoritmo:

- ➢ Si el orden de llegada es el mismo, n(1)=n(2), ¿qué ocurre?¿cuál de los dos procesos entra en su sección crítica?

En este caso habría que comparar el pdi (nº de identificador del proceso).
El orden quedaría determinado así, de forma que entraría en su sección crítica el que tuviera mayor pdi.
Este posible problema no se da , puesto que un proceso, al coger número, cede la entrada al otro proceso, de forma que no pueden tener nunca el mismo valor y al salir de su sección crítica, pone su n(i)=0; esto quiere decir que no ha cogido número y, por tanto, no tiene opción de entrar en su sección crítica.

- ➢ Sincronización y comunicación entre los procesos:

Si se cambia el número de iteraciones que se repite cada proceso, tenemos que cuanto menor sea el número (considerando como pequeño menos de 50 iteraciones), la comunicación entre los procesos disminuye en gran medida, es decir, la ejecución de los

procesos no se va "intercalando", sino que un proceso realiza todo su trabajo y cuando finaliza, el otro proceso se ejecuta después.

Para resolver este problema podemos hacer crecer el número de iteraciones o aumentar el tiempo de ejecución de los procesos incluyendo tiempos de retardo ya sea en la sección crítica (función incrementa) o en el resto. Así conseguimos que al aumentar el tiempo de ejecución de un proceso el otro proceso pueda entrar en su sección crítica.