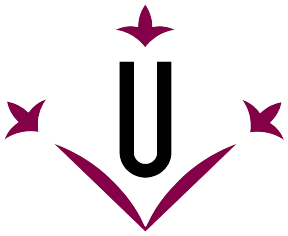

Universitat de Lleida
Escola Politècnica Superior

Sistemes Concurrents i Paral·lels

Problema 2: Sincronització

Sergi Puigpinós Palau
Jordi Rafael Lazo Florensa

22 de desembre de 2020

1 Problema 1

1.1 a)

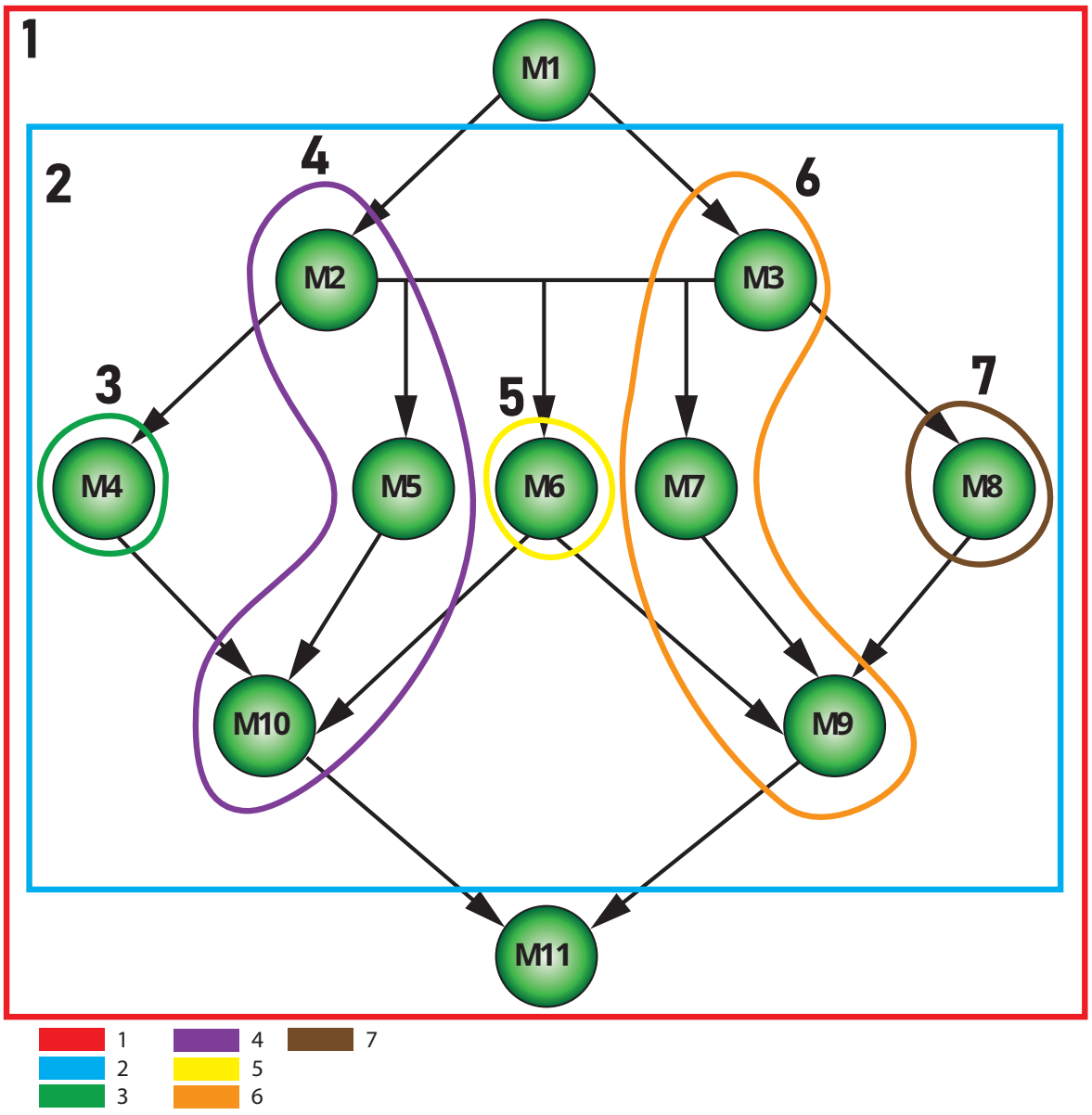

Figura 1: Graf després de la implementació mitjançant el constructor *ParBegin/ParEnd*.

Implementació mitjançant el constructor *ParBegin/ParEnd* assumint que tots els processos s'arrenquen només quan són necessaris.

```
SemáforoNario SemA=0, SemB=0, SemC=0, SemD=0;

Begin:
|    M1;
|    CoBegin:
|    |   Begin:
|    |    |             P(SemA); M4; V(SemB);
|    |    End
|    |
|    |    Begin:
|    |    |                       M2; V(SemA,3);
|    |    |             P(SemC); M5; P(SemB,2);
|    |    |                      M10;
|    |    End
|    |
|    |    Begin:
|    |    |    P(SemA); P(SemC); M6; V(SemB); V(SemD);
|    |    End
|    |
|    |    Begin:
|    |    |                       M3; V(SemC,3);
|    |    |             P(SemA); M7;
|    |    |           P(SemD,2); M9;
|    |    End
|    |
|    |    Begin:
|    |    |             P(SemD); M8; V(SemD);
|    |    End
|    CoEnd
|    M11;
End
```

1.2 b)

Implementació mitjançant els constructors *ParBegin/ParEnd*, assumint que tots els processos s'arrenquen simultàniament a l'inici de l'aplicació.

```
SemáforoNario SemA=0, SemB=0, SemC=0, SemD=0, SemE=0, SemF=0;

CoBegin:
|                      M1; V(SemA,2);
|           P(SemA); M2; V(SemB,4);
|           P(SemA); M3; V(SemC,4);
|           P(SemB); M4; V(SemD);
|  P(SemB); P(SemC); M5; V(SemD);
|  P(SemB); P(SemC); M6; V(SemD); V(SemE);
|  P(SemB); P(SemC); M7; V(SemE);
|           P(SemC); M8; V(SemE);
|         P(SemE,3); M9; V(SemF);
|         P(SemD,3); M10; V(SemF);
|         P(SemF,2); M11;
CoEnd
```

2 Problema 2

Pseudo-codi del procés P_i

```
1  Inicialment: Bandera[i] = Bandera[j] =Falso;
2  Do {
3       Bandera [i] = Cierto;
4       While (Bandera [j]==Cierto) {
5            Bandera [i] = Falso;
6            While (Bandera [j]==Cierto);
7            Bandera [i] = Cierto;
8       }
9       REGION CRITICA
10      Bandera[j] = Falso;
11      .....
12 } while(1);
```

Pseudo-codi del procés P_j

```
1  Inicialment: Bandera[i] = Bandera[j] =Falso;
2  Do {
3       Bandera [j] = Cierto;
4       While (Bandera [i]==Cierto) {
5            Bandera [j] = Falso;
6            While (Bandera [i]==Cierto);
7            Bandera [j] = Cierto;
8       }
9       REGION CRITICA
10      Bandera[i] = Falso;
11      .....
12 } while(1);
```

- Exclusió mútua: Si es compleix.
Es compleix ja que en el primer bucle *While*, en una execució normal, aquest evitarà que els dos processos entrin dins la secció crítica si un d'ells ja ha ficat la seva bandera a *Cierto* i l'altre s'executa més tard o processa el bucle *While* més tard. En l'altre cas, si els dos entren dins del primer bucle (si els dos fiquen la seva bandera a *Cierto* abans que processin el bucle), a causa de la instrucció de la línia 7, farà o que almenys un d'ells surti a executar la *REGION CRITICA* i l'altre es quedi o produirà un *Deadlock* que farà que els dos es quedin atrapats i cap d'ells executi la *REGION CRITICA*.

- Progressió $_a$: Si es compleix.
En cas de no succeir algun dels esdeveniments descrits en els altres apartats, aquesta sí que ho complirà, ja que mentre un procés fiqui la bandera a cert, aquest podrà entrar mentre que l'altre s'anirà a esperar al primer bucle *While*. Fent així que almenys hi hagui una situació en què hi ha progrés.

- Progressió $_b$: No es compleix.
En aquest cas, tal com s'ha comentat en l'exclusió mútua, en la situació en què els dos processos entrin dins el primer bucle. Si aquests processen la instrucció 7 abans que un d'ells processi de nou la condició del primer bucle while, en fixar les banderes i, j a *Cert*, la condició del bucle es complirà per tant els dos es quedaran bloquejats indefinidament. Fent que es produeix un *Deadlock*.

- Espera limitada: No es compleix.
No es compleix, ja que si s'executa el procés P_i fins a la secció crítica i P_j s'executa després aquest es quedarà esperant al segon *While* a què 'i = Cierto', però com que l'únic moment en què això succeeixi en P_i serà quan entri dintre del primer *While*, P_j es quedarà atrapat indefinidament perquè P_i mai entrarà dins del primer *While*, ja que el valor de j sempre serà *Falso* i ningú el canviarà. Aquesta situació dependrà de quin procés s'executi primer. Llavors això fa que no hi hagui una manera que els processos es vagin tornant els torns i que l'espera limitada no es compleixi.

3 Problema 3

3.1 (a)

TRUCADA
/* Indiquem que arriba UNA TRUCADA */

```
sem_signal(Trucada);
```

RECEPTOR
mentre(cert)
{
/*Està atent a l'arribada de noves trucades. Si a la cua de recepció hi ha prou espai, l'encua*/

```
sem_wait(Trucada); sem_wait(Espai); sem_wait(Mutex);
```

guarda la trucada a la cua de recepció

```
sem_signal(Mutex); sem_signal(TrucadesAssignador);
```

}

ASSIGNADOR
mentre(cert)
{
/* Si hi ha alguna trucada en espera a la cua de recepció, i un operador lliure, redirecciona la trucada cap a un operador */

```
sem_wait(TrucadesAssignador); sem_wait(OperadorLliure); sem_wait(Mutex);
```

desencua la trucada i la redirecciona a un operador

```
sem_signal(Mutex); sem_signal(Espai); sem_signal(TrucadesTramitador);
```

}

TRAMITADOR
mentre(cert)
{
/* Atén la trucada */

```
sem_wait(TrucadesTramitador);
```

atén la trucada

```
sem_signal(OperadorLliure);
```

}

3.2 (b)

SemáforsNaris
Trucada=0 (nombre de trucades disponibles que poden agafar els receptors)
Espai=T (nombre d'espais lliures de la cua de recepció)
TrucadesAssignador=0 (nombre de trucades disponibles que poden agafar els assignadors)
OperadorLliure=0 (nombre d'operadors lliures)
TrucadesTramitador=0 (esperar trucada assignada pel assignador)

SemáforsBinaris
Mutex=1 (bloquejar camins crítics i seccions de codi que contenen recursos els quals diferents processos comparteixen)