REPORTE – PRACTICA 3

Programación concurrente y paralela

29 DE MARZO DE 2025

ALUMNO: ENRIQUE HERNANDEZ LUNA
DOCENTE: DRA CARMEN CERON GARCIA

En esta practica se desarrollaron 3 programas los cuales fueron:

1. El implementar el programa del consultorio médico el cual se logró desarrollar haciendo uso de semáforos
2. Fue desarrollar el programa del árbol de 5 procesos los cuales pueden ir iniciando hasta que el proceso previe a ellos ha terminado, excepto para el primero, el cual es quien inicia todo.
3. El tercer programa es desarrollar un programa similar como el anterior pero ahora serán 7 procesos en total. Este programa en sus dos versiones, tanto Thread como Runnable. En el cual se deben de completar los siguientes puntos:
   3.1. Par **Cobegin / Coend**

```
cobegin
  P1:
    S1;
    SIGNAL(a);
    SIGNAL(b);

  P2:
    WAIT(a);
    S2;
    SIGNAL(c);

  P3:
    WAIT(b);
    S3;
    SIGNAL(e);

  P4:
    WAIT(c);
    S4;
    SIGNAL(d);
    SIGNAL(f);

  P5:
    WAIT(f);
    S5;
    SIGNAL(g);
```

```
  P6:
    WAIT(d);
    WAIT(e);
    S6;
    SIGNAL(g);

  P7:
    WAIT(g);
    WAIT(g);
    S7;
coend;
```

3.2. Par **Begin / End** (uno por proceso)

```
P1
Begin
  S1
  SIGNAL(a)
  SIGNAL(b)
End

P2
Begin
  WAIT(a)
  S2
  SIGNAL(c)
End

P3
Begin
  WAIT(b)
  S3
  SIGNAL(e)
End

P4
Begin
  WAIT(c)
```

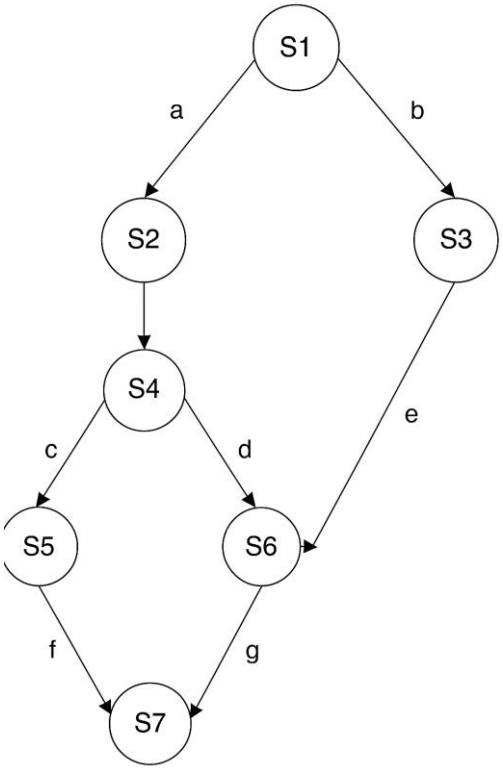
```
  S4
  SIGNAL(d)
  SIGNAL(f)
End

P5
Begin
  WAIT(f)
  S5
  SIGNAL(g)
End

P6
Begin
  WAIT(d)
  WAIT(e)
  S6
  SIGNAL(g)
End

P7
Begin
  WAIT(g)
  WAIT(g)
  S7
End
```

3.3. Notación con semáforos (mutex + sincronización)

```
P1
Begin
  mutex.WAIT()
  S1
  mutex.SIGNAL()
  a.SIGNAL()
  b.SIGNAL()
End

P2
```

```
Begin
  a.WAIT()
  mutex.WAIT()
  S2
  mutex.SIGNAL()
  c.SIGNAL()
End

P3
Begin
  b.WAIT()
  mutex.WAIT()
  S3
  mutex.SIGNAL()
  e.SIGNAL()
End

P4
Begin
  c.WAIT()
  mutex.WAIT()
  S4
  mutex.SIGNAL()
  d.SIGNAL()
  f.SIGNAL()
End

P5
Begin
  f.WAIT()
  mutex.WAIT()
  S5
  mutex.SIGNAL()
  g.SIGNAL()
End

P6
Begin
```

```
  d.WAIT()
  e.WAIT()
  mutex.WAIT()
  S6
  mutex.SIGNAL()
  g.SIGNAL()
End

P7
Begin
  g.WAIT()
  g.WAIT()
  mutex.WAIT()
  S7
  mutex.SIGNAL()
End
```



```java
package Practica3;
import java.util.concurrent.Semaphore;

class ConsultorioManager {
  // Semáforo para controlar el acceso a los consultorios
  private final Semaphore consultorios;

  // Constructor que inicializa el semáforo con el número de consultorios disponibles
  public ConsultorioManager(int numConsultorios) {
    this.consultorios = new Semaphore(numConsultorios, true);
  }

  // Método para que un paciente ingrese a un consultorio
  // Si no hay consultorios disponibles, el paciente esperará hasta que uno se libere
  public void ingresarConsultorio(String nombrePaciente) throws InterruptedException {
    consultorios.acquire();
    System.out.println(nombrePaciente + " está usando el consultorio.");
  }

  // Método para que un paciente salga de un consultorio
  // Libera el semáforo para que otro paciente pueda ingresar
  public void salirConsultorio(String nombrePaciente) {
    System.out.println(nombrePaciente + " terminó en el consultorio.");
    consultorios.release();
  }
}

```

```java
package Practica3;

class Paciente implements Runnable {
  // El ID del paciente
  private final int id;

  // Instancias de los manejadores de consultorios y turnos

  private final ConsultorioManager consultorios;

  // Instancia del manejador de turnos
  // Este manejador controla el orden en que los pacientes son atendidos
  private final TurnoManager turnos;

  // Constructor: recibe ID del paciente y las referencias necesarias para sincronización
  public Paciente(int id, ConsultorioManager consultorios, TurnoManager turnos) {
    this.id = id;
    this.consultorios = consultorios;
    this.turnos = turnos;
  }

  // Simula la atención inicial por parte de la enfermera
  private void atencionEnfermera() throws InterruptedException {
    System.out.println("Paciente " + id + " está siendo atendido por la enfermera.");
    Thread.sleep(500); // Pausa el hilo para simular el tiempo de atención
    System.out.println("Paciente " + id + " terminó con la enfermera.");
  }

  // Simula la atención médica posterior al consultorio
  private void atencionMedico() throws InterruptedException {
    System.out.println("Paciente " + id + " está siendo atendido por el médico.");
    Thread.sleep(700); // Pausa el hilo para simular el tiempo de consulta médica
    System.out.println("Paciente " + id + " terminó con el médico.");
  }

```

```
36.   // Método run que se ejecuta cuando el hilo del paciente inicia
37.   @Override
38.   public void run() {
39.     try {
40.       // Espera hasta que sea el turno del paciente
41.       turnos.esperarTurno(id);
42.
43.       // Atención con la enfermera
44.       atencionEnfermera();
45.
46.       // Solicita acceso a un consultorio
47.       consultorios.ingresarConsultorio("Paciente " + id);
48.       Thread.sleep(1000);
49.       consultorios.salirConsultorio("Paciente " + id);
50.
51.       // Atención con el médico
52.       atencionMedico();
53.
54.       // // Mensaje final de atención completa
55.       System.out.println("Paciente " + id + " ha terminado su atención.\n");
56.
57.       // Permite que el siguiente paciente continúe
58.       turnos.siguienteTurno();
59.
60.     } catch (InterruptedException e) {
61.       Thread.currentThread().interrupt();
62.       System.out.println("Paciente " + id + " fue interrumpido.");
63.     }
64.   }
65. }
66.
```

```
 1. package Practica3;
 2.
 3. public class PacientesSemaforo {
 4.   public static void main(String[] args) {
 5.     int numConsultorios = 1;
 6.     int numPacientes = 5;
 7.
 8.     ConsultorioManager consultorios = new ConsultorioManager(numConsultorios);
 9.     TurnoManager turnos = new TurnoManager();
10.
11.     for (int i = 1; i <= numPacientes; i++) {
12.       Thread hiloPaciente = new Thread(new Paciente(i, consultorios, turnos));
13.       hiloPaciente.start();
14.     }
15.   }
16. }
17.
```

```
 1. package Practica3;
 2.
 3. class TurnoManager {
 4.   // Variable que almacena el número de turno actual
 5.   private int turnoActual = 1;
 6.
 7.   // Método sincronizado que bloquea al paciente hasta que sea su turno
 8.   public synchronized void esperarTurno(int miTurno) throws InterruptedException {
 9.     // Mientras no sea el turno del paciente, espera (wait libera el monitor y bloquea el
hilo)
10.     while (miTurno != turnoActual) {
```

```
11.         wait();
12.       }
13.     }
14.
15.     // Método sincronizado que avanza al siguiente turno y notifica a todos los hilos
esperando
16.     public synchronized void siguienteTurno() {
17.       turnoActual++; // Se incrementa el turno actual
18.       notifyAll(); // Se despiertan todos los hilos para que verifiquen si ahora es su turno
19.     }
20. }
21.
```

```
Paciente 2 estÃ¡ siendo atendido por la enfermera.
Paciente 2 terminÃ³ con la enfermera.
Paciente 2 estÃ¡ usando el consultorio.
Paciente 2 terminÃ³ en el consultorio.
Paciente 2 estÃ¡ siendo atendido por el mÃ©dico.
Paciente 2 terminÃ³ con el mÃ©dico.
Paciente 2 ha terminado su atenciÃ³n.

Paciente 3 estÃ¡ siendo atendido por la enfermera.
Paciente 3 terminÃ³ con la enfermera.
Paciente 3 estÃ¡ usando el consultorio.
Paciente 3 terminÃ³ en el consultorio.
Paciente 3 estÃ¡ siendo atendido por el mÃ©dico.
Paciente 3 terminÃ³ con el mÃ©dico.
Paciente 3 ha terminado su atenciÃ³n.

Paciente 4 estÃ¡ siendo atendido por la enfermera.
Paciente 4 terminÃ³ con la enfermera.
Paciente 4 estÃ¡ usando el consultorio.
Paciente 4 terminÃ³ en el consultorio.
Paciente 4 estÃ¡ siendo atendido por el mÃ©dico.
Paciente 4 terminÃ³ con el mÃ©dico.
Paciente 4 ha terminado su atenciÃ³n.

Paciente 5 estÃ¡ siendo atendido por la enfermera.
Paciente 5 terminÃ³ con la enfermera.
Paciente 5 estÃ¡ usando el consultorio.
Paciente 5 terminÃ³ en el consultorio.
Paciente 5 estÃ¡ siendo atendido por el mÃ©dico.
Paciente 5 terminÃ³ con el mÃ©dico.
Paciente 5 ha terminado su atenciÃ³n.
```

```java
package Practica3.cincoProcesos;

// Clase que modela un semáforo binario simple
public class SemaforoBinario {
  private int contador;

  // Constructor: inicializa el semáforo con 0 o 1
  public SemaforoBinario(int valorInicial) {
    this.contador = valorInicial;
  }

  // Operación WAIT: espera si el semáforo está en 0
  public synchronized void WAIT() {
    while (contador == 0) {
      try {
        wait(); // El hilo se bloquea hasta que alguien llame a SIGNAL()
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
    }
    contador--; // Reduce el contador a 0 (bloqueado)
  }

  // Operación SIGNAL: libera el semáforo
  public synchronized void SIGNAL() {
    contador = 1; // Establece a 1 (liberado)
    notify();     // Despierta a un hilo en espera
  }
}
```

```java
package Practica3.cincoProcesos;

// Clase base que define los semáforos compartidos entre procesos
public class Procesos2 {
  // Semáforos de sincronización entre instrucciones
  protected static final SemaforoBinario A = new SemaforoBinario(0);
  protected static final SemaforoBinario B = new SemaforoBinario(0);
  protected static final SemaforoBinario C = new SemaforoBinario(0);
  protected static final SemaforoBinario D = new SemaforoBinario(0);

  // Semáforo para proteger el uso de la consola (exclusión mutua)
  protected static final SemaforoBinario mutex = new SemaforoBinario(1);
}
```

```java
package Practica3.cincoProcesos;

public class Proceso4 extends Procesos2 implements Runnable {
  public void run() {
    C.WAIT(); // Espera señal de P2
    D.WAIT(); // Espera señal de P3

    mutex.WAIT(); // Bloquea consola
    System.out.println("Soy el proceso #4 y la instruccion S4");
    try {
      Thread.sleep(1000); // Pausa tras S4
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
    mutex.SIGNAL(); // Libera consola

```

```
17.     mutex.WAIT(); // Bloquea consola nuevamente
18.     System.out.println("Soy el proceso #4 y la instruccion S5");
19.     try {
20.       Thread.sleep(1000); // Pausa tras S5
21.     } catch (InterruptedException e) {
22.       Thread.currentThread().interrupt();
23.     }
24.     mutex.SIGNAL(); // Libera consola
25.   }
26. }
27.
```

```
 1. package Practica3.cincoProcesos;
 2.
 3. public class Proceso3 extends Procesos2 implements Runnable {
 4.   public void run() {
 5.     B.WAIT(); // Espera señal de P1
 6.
 7.     mutex.WAIT(); // Bloquea consola
 8.     System.out.println("Soy el proceso #3 y la instruccion S3");
 9.     try {
10.       Thread.sleep(200); // Pausa para legibilidad
11.     } catch (InterruptedException e) {
12.       Thread.currentThread().interrupt();
13.     }
14.     mutex.SIGNAL(); // Libera consola
15.
16.     D.SIGNAL(); // Señal para P4
17.   }
18. }
19.
```

```
 1. package Practica3.cincoProcesos;
 2.
 3. public class Proceso2 extends Procesos2 implements Runnable {
 4.   public void run() {
 5.     A.WAIT(); // Espera señal de P1
 6.
 7.     mutex.WAIT(); // Bloquea consola
 8.     System.out.println("Soy el proceso #2 y la instruccion S2");
 9.     try {
10.       Thread.sleep(1000); // Pausa para salida visible
11.     } catch (InterruptedException e) {
12.       Thread.currentThread().interrupt();
13.     }
14.     mutex.SIGNAL(); // Libera consola
15.
16.     C.SIGNAL(); // Señal para P4
17.   }
18. }
19.
```

```
 1. package Practica3.cincoProcesos;
 2.
 3. public class Proceso1 extends Procesos2 implements Runnable {
 4.   public void run() {
 5.     mutex.WAIT(); // Bloquea la consola
 6.     System.out.println("Soy el proceso #1 y la instruccion S1");
 7.     try {
```

```
        Thread.sleep(1000); // Pequeña pausa para visualizar mejor la salida
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
      mutex.SIGNAL(); // Libera la consola

      A.SIGNAL(); // Desbloquea P2
      B.SIGNAL(); // Desbloquea P3
    }
  }

```

```
package Practica3.cincoProcesos;

public class MainProcesos {
  public static void main(String[] args) {
    // Se crean los hilos con sus respectivas tareas
    Thread P1 = new Thread(new Proceso1());
    Thread P2 = new Thread(new Proceso2());
    Thread P3 = new Thread(new Proceso3());
    Thread P4 = new Thread(new Proceso4());

    // Inician todos los procesos concurrentemente
    P1.start();
    P2.start();
    P3.start();
    P4.start();
  }
}
```

```
@enrique on c-y-p 3.11.1 on master 136ms java Practica3/cincoProcesos/MainProcesos
Soy el proceso #1 y la instruccion S1
Soy el proceso #2 y la instruccion S2
Soy el proceso #3 y la instruccion S3
Soy el proceso #4 y la instruccion S4
Soy el proceso #4 y la instruccion S5
@enrique on c-y-p 3.11.1 on master 4.218s
```

```
package Practica3.sieteProcesos;

// Clase que simula un semáforo binario para sincronización de hilos
public class SemaforoBinario {
  private int contador;

  // Constructor: inicializa el valor del semáforo (0 o 1)
  public SemaforoBinario(int valorInicial) {
    this.contador = valorInicial;
  }

  // WAIT: bloquea al hilo si el semáforo está en 0
  public synchronized void WAIT() {
    while (contador == 0) {
      try {
        wait(); // El hilo espera hasta recibir señal
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
    }
    contador--; // Al salir del ciclo, decrementa el contador
  }
```

```
23.
24.   // SIGNAL: libera el semáforo y notifica a un hilo en espera
25.   public synchronized void SIGNAL() {
26.     contador = 1; // Activa el semáforo
27.     notify();     // Despierta a un hilo en espera
28.   }
29. }
30.
```

```
 1. package Practica3.sieteProcesos;
 2.
 3. // Clase base que contiene los semáforos compartidos entre procesos
 4. public class ProcesosBase {
 5.   // Semáforos para transiciones del grafo
 6.   protected static final SemaforoBinario a = new SemaforoBinario(0);
 7.   protected static final SemaforoBinario b = new SemaforoBinario(0);
 8.   protected static final SemaforoBinario c = new SemaforoBinario(0);
 9.   protected static final SemaforoBinario d = new SemaforoBinario(0);
10.   protected static final SemaforoBinario e = new SemaforoBinario(0);
11.   protected static final SemaforoBinario f = new SemaforoBinario(0);
12.   protected static final SemaforoBinario g = new SemaforoBinario(0);
13.
14.   // Semáforo para exclusión mutua en la consola
15.   protected static final SemaforoBinario mutex = new SemaforoBinario(1);
16. }
```

```
 1. package Practica3.sieteProcesos;
 2.
 3. public class Proceso7 extends ProcesosBase implements Runnable {
 4.   public void run() {
 5.     g.WAIT(); // Espera primera señal (desde Proceso5)
 6.     g.WAIT(); // Espera segunda señal (desde Proceso6)
 7.
 8.     mutex.WAIT();
 9.     System.out.println("Soy el proceso #7 y la instrucción S7");
10.
11.     // simulación de trabajo
12.     try {
13.       Thread.sleep(1000);
14.     } catch (InterruptedException e) {
15.       Thread.currentThread().interrupt(); // Manejo de interrupción
16.     }
17.
18.     mutex.SIGNAL();
19.   }
20. }
```

```
 1. package Practica3.sieteProcesos;
 2.
 3. public class Proceso6 extends ProcesosBase implements Runnable {
 4.   public void run() {
 5.     d.WAIT(); // Espera señal desde Proceso4
 6.     e.WAIT(); // Espera señal desde Proceso3
 7.
 8.     mutex.WAIT();
 9.     System.out.println("Soy el proceso #6 y la instrucción S6");
10.
11.     // simulación de trabajo
12.     try {
```

```
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL();

    g.SIGNAL(); // Señal para Proceso7
  }
}
```

```
package Practica3.sieteProcesos;

public class Proceso5 extends ProcesosBase implements Runnable {
  public void run() {
    f.WAIT(); // Espera señal desde Proceso4

    mutex.WAIT();
    System.out.println("Soy el proceso #5 y la instrucción S5");

    // simulación de trabajo
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL();

    g.SIGNAL(); // Señal para Proceso7
  }
}
```

```
package Practica3.sieteProcesos;

public class Proceso4 extends ProcesosBase implements Runnable {
  public void run() {
    c.WAIT(); // Espera señal desde Proceso2

    mutex.WAIT();
    System.out.println("Soy el proceso #4 y la instrucción S4");

    // simulación de trabajo
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL();

    d.SIGNAL(); // Habilita Proceso6
    f.SIGNAL(); // Habilita Proceso5
  }
}

```

```
package Practica3.sieteProcesos;

```

```java
public class Proceso3 extends ProcesosBase implements Runnable {
  public void run() {
    b.WAIT(); // Espera señal desde Proceso1

    mutex.WAIT();
    System.out.println("Soy el proceso #3 y la instrucción S3");

    // simulación de trabajo
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL();

    e.SIGNAL(); // Habilita Proceso6 (S6)
  }
}
```

```java
package Practica3.sieteProcesos;

public class Proceso2 extends ProcesosBase implements Runnable {
  public void run() {
    a.WAIT(); // Espera señal desde Proceso1

    mutex.WAIT();
    System.out.println("Soy el proceso #2 y la instrucción S2");

    // simulación de trabajo
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL();

    c.SIGNAL(); // Habilita Proceso4 (S4)
  }
}
```

```java
package Practica3.sieteProcesos;

public class Proceso1 extends ProcesosBase implements Runnable {
  public void run() {
    mutex.WAIT(); // Entra en sección crítica (impresión)
    System.out.println("Soy el proceso #1 y la instrucción S1");

    // simulación de trabajo
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt(); // Manejo de interrupción
    }

    mutex.SIGNAL(); // Sale de sección crítica

    a.SIGNAL(); // Habilita Proceso2 (S2)
    b.SIGNAL(); // Habilita Proceso3 (S3)
  }
```

```
}
```

```
package Practica3.sieteProcesos;

// Clase principal que lanza todos los hilos
public class MainProcesos {
  public static void main(String[] args) {
    new Thread(new Proceso1()).start();
    new Thread(new Proceso2()).start();
    new Thread(new Proceso3()).start();
    new Thread(new Proceso4()).start();
    new Thread(new Proceso5()).start();
    new Thread(new Proceso6()).start();
    new Thread(new Proceso7()).start();
  }
}
```

```
@enrique on c-y-p 3.11.1 on master java Practica3/sieteProcesos/MainProcesos
Soy el proceso #1 y la instrucciÃ³n S1
Soy el proceso #3 y la instrucciÃ³n S3
Soy el proceso #2 y la instrucciÃ³n S2
Soy el proceso #4 y la instrucciÃ³n S4
Soy el proceso #5 y la instrucciÃ³n S5
Soy el proceso #6 y la instrucciÃ³n S6
Soy el proceso #7 y la instrucciÃ³n S7
@enrique on c-y-p 3.11.1 on master 7.189s
```

```
package Practica3.sieteProcesosThread;

// Clase que implementa un semáforo binario
public class SemaforoBinario {
  private int contador;

  // Constructor que inicializa el semáforo con 0 o 1
  public SemaforoBinario(int valorInicial) {
    this.contador = valorInicial;
  }

  // Método WAIT: bloquea el hilo si el semáforo está en 0
  public synchronized void WAIT() {
    while (contador == 0) {
      try {
        wait(); // Libera el monitor y entra en espera
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      }
    }
    contador--; // Reduce el valor del semáforo a 0
  }

  // Método SIGNAL: libera el semáforo y notifica a un hilo
  public synchronized void SIGNAL() {
    contador = 1; // Restaura el valor a 1 (activo)
    notify();     // Despierta a un hilo en espera
  }
}

```

```java
package Practica3.sieteProcesosThread;

// Clase base que contiene todos los semáforos compartidos entre procesos
public class ProcesosBase {
  protected static final SemaforoBinario a = new SemaforoBinario(0);
  protected static final SemaforoBinario b = new SemaforoBinario(0);
  protected static final SemaforoBinario c = new SemaforoBinario(0);
  protected static final SemaforoBinario d = new SemaforoBinario(0);
  protected static final SemaforoBinario e = new SemaforoBinario(0);
  protected static final SemaforoBinario f = new SemaforoBinario(0);
  protected static final SemaforoBinario g = new SemaforoBinario(0);

  // Semáforo para proteger la impresión en consola (exclusión mutua)
  protected static final SemaforoBinario mutex = new SemaforoBinario(1);
}
```

```java
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S7, espera dos señales 'g' (de P5 y P6)
public class Proceso7 extends Thread {
  public void run() {
    ProcesosBase.g.WAIT(); // Desde P5
    ProcesosBase.g.WAIT(); // Desde P6

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #7 y la instrucción S7");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL();
  }
}
```

```java
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S6, tras recibir señales 'd' y 'e'
public class Proceso6 extends Thread {
  public void run() {
    ProcesosBase.d.WAIT(); // Desde S4
    ProcesosBase.e.WAIT(); // Desde S3

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #6 y la instrucción S6");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL();

    ProcesosBase.g.SIGNAL(); // Envía señal a S7 (P7)
  }
}
```

```java
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S5, tras recibir señal 'f'
public class Proceso5 extends Thread {
  public void run() {
    ProcesosBase.f.WAIT(); // Espera a que S4 termine

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #5 y la instrucción S5");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
```

```
    ProcesosBase.mutex.SIGNAL();

    ProcesosBase.g.SIGNAL(); // Envía señal a S7 (P7)
  }
}
```

```
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S4, tras recibir señal 'c'
public class Proceso4 extends Thread {
  public void run() {
    ProcesosBase.c.WAIT(); // Espera a que S2 termine

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #4 y la instrucción S4");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL();

    ProcesosBase.d.SIGNAL(); // Habilita parte de S6 (P6)
    ProcesosBase.f.SIGNAL(); // Habilita S5 (P5)
  }
}
```

```
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S3, tras recibir señal 'b'
public class Proceso3 extends Thread {
  public void run() {
    ProcesosBase.b.WAIT(); // Espera a que S1 termine

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #3 y la instrucción S3");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL();

    ProcesosBase.e.SIGNAL(); // Habilita parte de S6 (P6)
  }
}
```

```
package Practica3.sieteProcesosThread;

// Proceso que ejecuta S2, tras recibir señal 'a'
public class Proceso2 extends Thread {
  public void run() {
    ProcesosBase.a.WAIT(); // Espera a que S1 termine

    ProcesosBase.mutex.WAIT();
    System.out.println("Soy el proceso #2 y la instrucción S2");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL();

    ProcesosBase.c.SIGNAL(); // Habilita S4 (P4)
  }
}
```

```
package Practica3.sieteProcesosThread;

```

```java
// Proceso que ejecuta S1 y habilita a P2 y P3
public class Proceso1 extends Thread {
  public void run() {
    ProcesosBase.mutex.WAIT(); // Entra en sección crítica
    System.out.println("Soy el proceso #1 y la instrucción S1");
    try { Thread.sleep(200); } catch (InterruptedException e) {}
    ProcesosBase.mutex.SIGNAL(); // Sale de sección crítica

    ProcesosBase.a.SIGNAL(); // Habilita S2 (P2)
    ProcesosBase.b.SIGNAL(); // Habilita S3 (P3)
  }
}
```

```java
package Practica3.sieteProcesosThread;

// Clase principal que lanza todos los procesos
public class MainSemaforos {
  public static void main(String[] args) {
    new Proceso1().start();
    new Proceso2().start();
    new Proceso3().start();
    new Proceso4().start();
    new Proceso5().start();
    new Proceso6().start();
    new Proceso7().start();
  }
}
```

```
@enrique on c-y-p 3.11.1 on master 7.189s java Practica3/sieteProcesosThread/MainSemaforos
Soy el proceso #1 y la instrucciÃ³n S1
Soy el proceso #2 y la instrucciÃ³n S2
Soy el proceso #4 y la instrucciÃ³n S4
Soy el proceso #3 y la instrucciÃ³n S3
Soy el proceso #5 y la instrucciÃ³n S5
Soy el proceso #6 y la instrucciÃ³n S6
Soy el proceso #7 y la instrucciÃ³n S7
@enrique on c-y-p 3.11.1 on master 1.624s
```