1. Se desea resolver un problema de comunicación entre dos procesos, llamados Productor yConsumidor, mediante un buffer que es capaz de albergar un único valor entero en cada instante.

|  |
| --- |
| package pc.ejercicios3.prodcons; class Buffer {  private volatile int valor; **private WatchDog guardiaBuffer;**  public Buffer() {  **this.guardiaBuffer = new WatchDog();**  }    public void meter(int valor) {  **this.guardiaBuffer.waitWatchDogProduccer();**  System.out.println("Buffer: meter " + valor);  this.valor = valor; **this.guardiaBuffer.warn();**  }  public int sacar() {  **this.guardiaBuffer.waitWatchDogCostumer();**  System.out.println("Buffer: sacar " + valor);  //Int aux=valor;  **this.guardiaBuffer.done();**  return valor;//return aux;  *try{*  *return valor*  *} finaly{*  *this.guardiaBuffer.done();*  *}*  *}*  } |
| **public class WatchDog {**  **private volatile boolean go;**  **//false = buffer vacio**  **//true = buffer lleno**  **public WatchDog() { go=false;**  **}**    **/\*\***   * **Avisa que el buffer está lleno**   **\*/**  **public void warn(){ go=true;**  **}**    **/\*\***   * **Avisa que el buffer está vacio**   **\*/**  **public void done(){ go=false;**  **}**    **/\*\***   * **Espera si el buffer está vacio**   **\*/**  **public void waitWatchDogProduccer(){**  **while(go);**  **}**    **/\*\***   * **Espera si el buffer está lleno**   **\*/** |

**public void waitWatchDogCostumer(){**

**while(!go);**

**}**

**}**

1. Busque en la bibliografía un algoritmo para resolver el problema de la exclusión mutua para Nprocesos usando únicamente variables compartidas y bucles de espera activa.

|  |  |
| --- | --- |
| **module**Exclusion\_Mutua\_P; **var** flag1,flag2: boolean; turno: integer;  **procedure** bloqueo(**var** mi\_flag, su\_flag: boolean; su\_turno:integer); **begin** mi\_flag := true; turno := su\_turno;  **while** su\_flag **and** (turno = su\_turno) **do** ; **end**  **end** bloqueo;  **procedure** desbloqueo(**var** mi\_flag: boolean); begin mi\_flag := false;  **end** desbloqueo;  **process** P1 begin **loop** | |
|  | bloqueo(flag1,flag2,2);  (\* Uso del recurso Sección Crítica \*) desbloqueo(flag1);  (\* resto del proceso \*) |
| **end end** P1;  **process** P2 **begin loop** | |
|  | bloqueo(flag2,flag1,1);  (\* Uso del recurso Sección Crítica \*)  desbloqueo(flag2);  (\* resto del proceso \*) |
| **end**  **end** P2;  **begin** (\* Exclusion\_Mutua\_P\*) flag1 := FALSE; flag2 := FALSE; | |
|  | **cobegin**  P1;  P2; |
| **coend**  **end** Exclusion\_Mutua\_P | |

Si sólo uno de los procesos intenta acceder a la sección crítica lo podrá hacer sin ningún problema. Sin embargo, si ambos intentan entrar a la vez el valor de turno se pondrá a 1 y 2 pero sólo un valor de ellos permanecerá al escribirse sobre el otro, permitiendo el acceso de un proceso a su región crítica.

El algoritmo permite resolver el problema de la exclusión mutua y garantiza que ambos procesos usarán de forma consecutiva el recurso en el caso de que lo soliciten a la vez y se impedirá el cierre del otro proceso.

Si, por ejemplo, P1 ha entrado en su sección crítica bloqueando el acceso de P2 éste entrará una vez haya finalizado aquél, ya que cuando P1 sale de su región crítica desactiva su indicador, permitiendo el acceso de P2; si una vez que P1 sale de su sección crítica P2 no está en ejecución, P1 deberá permanecer en espera hasta que P2 haya entrado y haya desactivado su indicador. De esta manera se evita que P2 pueda quedar relegado por P1 en el uso del recurso, es decir ambos gozan de la misma prioridad en el uso del recurso.