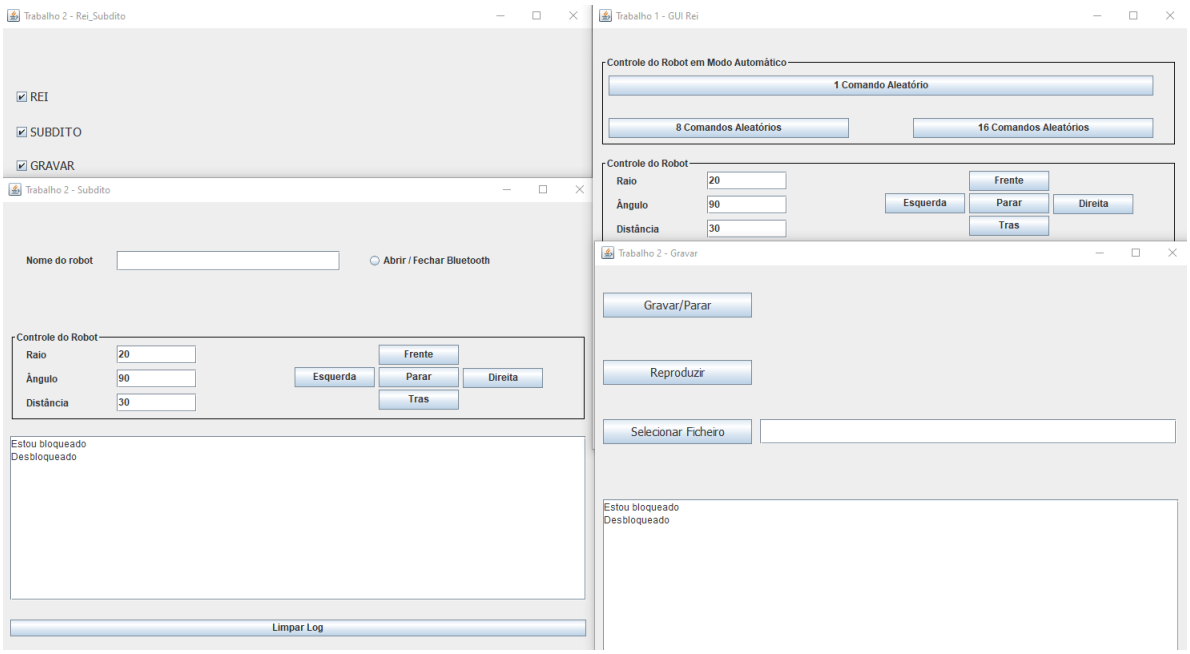
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos - 2324SI

2º Trabalho Prático - Aula prática 5



Docente Carlos Carvalho

Realizado por (Grupo 7):
Diogo Santos 48626
Pedro Silva 48965
João Fonseca 49707

26 de setembro de 2024

Conteúdo

Lista de Figuras

1 Introdução

Esta aula consistiu na implementação e teste do lançamento e sincronização entre o processo REI_SUBDITO e as tarefas REI, SUBDITO e GRAVAR de modo ao processo REI_SUBDITO conseguir ativar ou desativar os diferentes comportamentos. Isto significa que a qualquer momento na execução de uma das três tarefas, o processo REI_SUBDITO pode bloquear ou desbloqueá-las. Este mecanismo já tinha sido implementado em aulas anteriores para as tarefas REI e SUBDITO mas vamos explicá-las aqui assim como adicionar esta função para a tarefa GRAVAR.

2 Desenvolvimento

2.1 Tarefas Rei e Súbdito

Este mecanismo vai ser implementado da mesma forma ambos para o Rei como para o Subdito, em que vai existir um semáforo comum à GUI_REI_SUBDITO e a ambas estas tarefas que vai ser controlado através do clique do respetivo botão. Vamos olhar por exemplo para a máquina de estados da tarefa Rei:

```
case dormir:
    //System.out.println("sleep");
    try {
        Thread.sleep(1000);
        //System.out.println("permits" + reiAvailable.availablePermits());
        if(reiAvailable.availablePermits() == 0) {
            state = bloqueado;
            break;
        }
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

Figura 1: Estado 'dormir' - Classe App_Rei

Em ambos os estados dormir destas tarefas fazemos uma verificação do número de "permits"do respetivo semáforo e se for igual a 0, ou seja o botão da tarefa na GUI_REI_SUBDITO está desligado, vamos para o estado bloqueado se não o programa continua normalmente.

```
case bloqueado:
    try {
        gui.write("Estou bloqueado \n");
        reiAvailable.acquire();
        gui.write("Desbloqueado \n");
        state = dormir;
        break;
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

Figura 2: Estado 'bloqueado' - Classe App_Rei

Neste estado avisamos a respetiva tarefa que estamos bloqueados e ficamos à espera, "acquire", de que exista um "release"algo que só acontece quando carregam no botão da tarefa na GUI principal. Com este mecanismo garantimos que todas as ações sejam levadas até ao fim e só depois disso vamos para o estado bloqueado.

2.2 Tarefa Gravar

2.2.1 Gravação

Nesta tarefa vai ser um pouco diferente da anterior, pois como sabemos a gravação é feita através do súbdito, logo vamos ter que encontrar uma forma de poder bloquear o método gravador. Isto é importante pois, pode haver a ocasião em que estamos em modo de gravação e escolhemos bloquear, nessa ocasião, apesar de estarmos em modo de gravação, não pode haver gravação de informação no ficheiro. A nossa solução é então:

```
synchronized void reta(Mensagem msg) {
    //System.out.println(gravador.getGravar());
    if (gravador!=null && gravador.getGravar() && gravador.bdGravar.getGravarOff())
        gravador.reta(msg);
    //System.currentTimeMillis();
}
```

Figura 3: Método 'reta' - Classe RobotEV3

Antes de fazermos a gravação no ficheiro verificamos se estamos bloqueados ou não. Esta verificação é feita através duma variável guardada na base de dados e que é modificada pela atividade na GUI principal. Como é óbvio se estivermos bloqueados não guardamos, caso contrário guardamos.

2.2.2 Reproduzir

O reproduzir vai seguir a mesma estratégia do Rei e do súbdito mas devido à característica das suas mensagens terem um compasso de espera umas entre as outras vamos ter que colocar a verificação dos "permits"antes e depois dessa espera. Fica então:

```
long sleepTime = currentTime - lastTime;
//System.out.println(sleepTime);
// Check if the sleep time is non-negative
if (lastTime!= 0) {
    // Introduce a delay based on the time difference
    System.out.println("sleep");
    Thread.sleep(sleepTime);
    System.out.println("wake");
    if (!bd.getReproduzir())
        break;
}
if(gravarAvailable.availablePermits() == 0)
    bloqueado();
```

Figura 4: Estado 'reproduzir' - Classe App_Gravar

Colocamos a verificação, como mencionado em cima, no inicio do estado mas também colocamos a seguir a este "sleep", simula o tempo de espera entre as mensagens, pois o utilizador pode ter carregado no botão para bloquear enquanto estávamos à espera. Desta vez o estado bloqueado é um método pois queremos que, quando exista a libertação da tarefa esta continue onde parou. Este método vai ter um aspeto idêntico ao estado.

3 Conclusões

Como explicámos na introdução, a maior parte desta aula já tínhamos feito nas outras faltando-nos então a tarefa Gravar, algo que conseguimos implementar sem quaisquer problemas. Esta funcionalidade dá-nos uma maior robustez pois podemos controlar todas as execuções a partir da GUI do processo. Fica-nos a faltar então limar umas arestas para fazermos a validação final na próxima aula.

4 Bibliografia

1. Folhas de Computação Física - Jorge Pais, 2023/2024

5 Código Java GUI_Rei_Subdito, App_Rei, App_Subdito, App_Gravar

```
1
2  Classe GUI_Rei_Subdito
3
4  package ptrabalho;
5
6  import java.awt.EventQueue;
7
8  import javax.swing.JFrame;
9  import javax.swing.JPanel;
10 import javax.swing.border.EmptyBorder;
11 import javax.swing.JCheckBox;
12 import java.awt.Font;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.awt.event.WindowAdapter;
16 import java.awt.event.WindowEvent;
17 import java.util.concurrent.Semaphore;
18
19 import javax.swing.SwingConstants;
20
21 public class GUI_Rei_Subdito extends GUI_Base {
22
23     private JPanel contentPane;
24     private App_Rei appRei;
25     private App_Subdito appSub;
26     private App_Gravar appGravar;
27     private GUI_Subdito gui_Subdito;
28     private GUI_Gravar gui_Gravar;
29
30     private BD_Rei bdRei = new BD_Rei();
31     private BD_Gravar bdGravar = new BD_Gravar();
32     private Gravador gravador = new Gravador(bdGravar);
33     private BD_Subdito bdSub = new BD_Subdito(gravador);
34     private Semaphore subAvailable = new Semaphore(0);
35     private Semaphore reiAvailable = new Semaphore(0);
36     private Semaphore gravarAvailable = new Semaphore(0);
37
38     private BufferCircular bcG = new BufferCircular();
39     private Semaphore haTrabalhoG = new Semaphore(1);
40
41
42     public GUI_Rei_Subdito(BD_Base bd, BufferCircular bc, Semaphore
       ht)
43   {
44     super(bd);
45     EventQueue.invokeLater(new Runnable()
46     {
47       public void run()
48       {
49         try
50         {
51           init_Rei_Subdito(bc, ht);
52
53         } catch (Exception e)
54         {
55           e.printStackTrace();
56         }
```

```
57        }
58      });
59    }
60
61
62    public void init_Rei_Subdito(BufferCircular bc, Semaphore ht) {
63
64      setTitle("Trabalho 2 - Rei_Subdito");
65      //setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66      setBounds(0, 0, 754, 600);
67
68
69
70        //setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
71        getContentPane().setLayout(null);
72        JCheckBox chckbxRei = new JCheckBox("REI");
73        chckbxRei.setHorizontalAlignment(SwingConstants.LEFT);
74        chckbxRei.setFont(new Font("Tahoma", Font.PLAIN, 15));
75        chckbxRei.setBounds(16, 56, 97, 59);
76        getContentPane().add(chckbxRei);
77        JCheckBox chckbxSubdito = new JCheckBox("SUBDITO");
78        chckbxSubdito.setHorizontalAlignment(SwingConstants.LEFT);
79        chckbxSubdito.setFont(new Font("Tahoma", Font.PLAIN, 15));
80        chckbxSubdito.setBounds(16, 100, 195, 59);
81        getContentPane().add(chckbxSubdito);
82        JCheckBox chckbxGravar = new JCheckBox("GRAVAR");
83        chckbxGravar.setHorizontalAlignment(SwingConstants.LEFT);
84        chckbxGravar.setFont(new Font("Tahoma", Font.PLAIN, 15));
85        chckbxGravar.setBounds(16, 142, 108, 59);
86        getContentPane().add(chckbxGravar);
87        appRei = new App_Rei(bdRei, bc, ht, reiAvailable);
88        Thread tRei = new Thread(appRei);
89        tRei.start();
90
91        //appRei.run();
92        //gui_Subdito = new GUI_Subdito(bdSub);
93        appSub = new App_Subdito(bdSub, bc, ht, subAvailable, bcG,
    haTrabalhoG);
94        Thread tSub = new Thread(appSub);
95        tSub.start();
96        appGravar = new App_Gravar(bdGravar, gravador,
    gravarAvailable, bcG, haTrabalhoG);
97        Thread tGravar = new Thread(appGravar);
98        tGravar.start();
99
100       chckbxRei.addActionListener(new ActionListener()
101     {
102       public void actionPerformed(ActionEvent e)
103       {
104         if(chckbxRei.isSelected()) {
105           if (!appRei.gui.isVisible())
106             appRei.gui = new GUI_Rei(bdRei);
107           appRei.gui.start();
108           System.out.println(bdRei.getMensagens().size());
109           reiAvailable.release();
110           reiAvailable.release();
111           //appRei.run();
112           //t.run();
113           write("Ativei a GUI_REI \n");
114         }
```

```
115         else {
116           appRei.gui.off();
117           try {
118             reiAvailable.acquire();
119           } catch (InterruptedException e1) {
120             // TODO Auto-generated catch block
121             e1.printStackTrace();
122           }
123           write("Desativei a GUI_REI \n");
124         }
125 
126       }
127     });
128 
129       chckbxSubdito.addActionListener(new ActionListener()
130     {
131       public void actionPerformed(ActionEvent e)
132       {
133         if(chckbxSubdito.isSelected()) {
134           if (!appSub.gui.isVisible())
135             appSub.gui = new GUI_Subdito(bdSub);
136           appSub.gui.start();
137           System.out.println(bdRei.getMensagens().size());
138           subAvailable.release();
139           subAvailable.release();
140           write("Ativei a GUI_SUBDITO \n");
141         }
142         else {
143           appSub.gui.off();
144           try {
145             subAvailable.acquire();
146           } catch (InterruptedException e1) {
147             // TODO Auto-generated catch block
148             e1.printStackTrace();
149           }
150           write("Desativei a GUI_SUBDITO \n");
151         }
152 
153 
154       }
155     });
156 
157       chckbxGravar.addActionListener(new ActionListener()
158     {
159       public void actionPerformed(ActionEvent e)
160       {
161         if(chckbxGravar.isSelected()) {
162           if (!appGravar.gui.isVisible())
163             appGravar.gui = new GUI_Gravar(bdGravar);
164           appGravar.gui.start();
165           gravarAvailable.release();
166           gravarAvailable.release();
167           bdGravar.setGravarOff(true);
168           write("Ativei a GUI_GRAVAR \n");
169         }
170         else {
171           appGravar.gui.off();
172           try {
173             bdGravar.setGravarOff(false);
174             gravarAvailable.acquire();
```

```
175             } catch (InterruptedException e1) {
176               // TODO Auto-generated catch block
177               e1.printStackTrace();
178             }
179             write("Desativei a GUI_GRAVAR \n");
180           }
181
182
183         }
184       });
185
186         addWindowListener(new WindowAdapter(){
187             public void windowClosing(WindowEvent e){
188                 if (appSub.getBD().isLigado())
189                 {
190                   appSub.gui.write(" Desconectando o robot ... \n");
191                   write(" Desconectando o robot ... \n");
192                   appSub.getBD().getRobot().getRobot().CloseEV3();
193                   appSub.getBD().setLigado(false);
194                   System.exit(0);
195                 }
196                 else {
197                     System.out.println("Closing program");
198                     System.exit(0);
199                 }
200             }
201         });
202
203         setVisible(true);
204     }
205
206
207 }
208
209
210 Classe App_Rei
211
212 package ptrabalho;
213
214 import java.util.concurrent.Semaphore;
215
216 public class App_Rei extends Thread
217 {
218     @SuppressWarnings("unused")
219     protected GUI_Rei gui;
220     private BD_Rei bd;
221     Mensagem msg;
222     Mensagem myMensagem = null;
223     private int state = 2;
224     private int counter = 0;
225     private final int escreverMensagem = 1;
226     private final int dormir = 2;
227     private final int bloqueado = 4;
228
229     BufferCircular bufferCircular;
230     Semaphore haTrabalho, livreMyMensagem, ocupadaMyMensagem,
    acessoMyMensagem, reiAvailable;
231
232
```

```
233     public App_Rei(BD_Rei bdRei, BufferCircular bc, Semaphore ht,
    Semaphore ra)
234     {
235       bd = bdRei;
236         gui = new GUI_Rei(bd);
237         bufferCircular= bc;
238       haTrabalho= ht;
239       reiAvailable = ra;
240       myMensagem = null;
241       livreMyMensagem= new Semaphore(1);
242       ocupadaMyMensagem= new Semaphore(0);
243       acessoMyMensagem= new Semaphore(1);
244     }
245 
246     public BD_Rei getBD()
247     {
248         return bd;
249     }
250 
251     public void setMensagem(Mensagem m)
252     {
253       try {
254       livreMyMensagem.acquire();
255       acessoMyMensagem.acquire();
256       } catch (InterruptedException e) {}
257       myMensagem= m;
258       acessoMyMensagem.release();
259       ocupadaMyMensagem.release();
260     }
261 
262 
263     public void run()
264     {
265         while(true) {
266 
267           switch (state) {
268 
269             case escreverMensagem:
270               System.out.println("escreve");
271               System.out.println("Mensagens à espera: " + bd.
    getMensagens().size());
272               msg = bd.getMensagens().get(0);
273               msg.setId(counter);
274               setMensagem(msg);
275               try {
276                 ocupadaMyMensagem.acquire();
277                 acessoMyMensagem.acquire();
278               } catch (InterruptedException e) {}
279                 bufferCircular.inserirElemento(myMensagem);
280                 acessoMyMensagem.release();
281                 livreMyMensagem.release();
282                 haTrabalho.release();
283                 System.out.println("ht:" + haTrabalho.
    availablePermits());
284               gui.write(" Enviei = " + msg + "\n");
285               counter =++ counter % bd.getNMensagens();
286               bd.removeMensagem();
287               state = dormir;
288               break;
289 
```

```
290             case dormir:
291               //System.out.println("sleep");
292               try {
293                 Thread.sleep(1000);
294                 //System.out.println("permits" + reiAvailable.
    availablePermits());
295                 if(reiAvailable.availablePermits() == 0) {
296                   state = bloqueado;
297                   break;
298                 }
299               } catch (InterruptedException e) {
300                 // TODO Auto-generated catch block
301                 e.printStackTrace();
302               }
303               //System.out.println(bd.getMensagens().size());
304               if(bd.getMensagens().size() !=0) {
305                 state = escreverMensagem;
306                 break;
307               }
308               else {break;}
309
310             case bloqueado:
311               try {
312                 gui.write("Estou bloqueado \n");
313                 reiAvailable.acquire();
314                 gui.write("Desbloqueado \n");
315                 state = dormir;
316                 break;
317               } catch (InterruptedException e) {
318                 // TODO Auto-generated catch block
319                 e.printStackTrace();
320               }
321
322       }
323     }
324     /*System.out.println("sai");
325         try {
326       Thread.sleep(100);
327     } catch (InterruptedException e) {
328       // TODO Auto-generated catch block
329       e.printStackTrace();
330     }*/
331
332     }
333
334     /*public static void main(String[] args) throws
    InterruptedException
335     {
336
337         App_Rei app = new App_Rei();
338         System.out.println("A aplicação começou.");
339         app.run();
340         System.out.println("A aplicação terminou.");
341     }*/
342
343 }
344
345
346 Classe App_Subdito
347
```

```
348 package ptrabalho;
349 import java.lang.Math;
350 import java.util.concurrent.Semaphore;
351
352 public class App_Subdito extends Thread
353 {
354     @SuppressWarnings("unused")
355     protected GUI_Subdito gui;
356     private BD_Subdito bd;
357     Mensagem msg = null;
358     Mensagem myMensagem = null;
359     private int state = 2;
360     private int counter = 0;
361     private final int receberMensagem = 1;
362     private final int dormir = 2;
363     private final int esperarTempoExecucao = 3;
364     private final int bloqueado = 4;
365     private final int reproduzir = 5;
366     private final int RETA = 1;
367     private final int CURVARDIR = 2;
368     private final int CURVARESQ = 3;
369     private final int PARAR = 4;
370     private boolean aReproduzir = false;
371
372     BufferCircular bufferCircular, bufferCircularReproduzir;
373     Semaphore haTrabalho, livreMyMensagem, ocupadaMyMensagem,
    acessoMyMensagem, subAvailable, haTrabalhoG;
374
375     public App_Subdito(BD_Subdito bdSub, BufferCircular bc,
    Semaphore ht, Semaphore sub, BufferCircular bcG, Semaphore htG)
376     {
377       bd = bdSub;
378
379         gui = new GUI_Subdito(bd);
380
381         subAvailable = sub;
382         bufferCircular= bc;
383         bufferCircularReproduzir = bcG;
384         haTrabalho= ht;
385         haTrabalhoG= htG;
386         myMensagem= null;
387         livreMyMensagem= new Semaphore(1);
388         ocupadaMyMensagem= new Semaphore(0);
389         acessoMyMensagem= new Semaphore(1);
390     }
391
392     public BD_Subdito getBD()
393     {
394         return bd;
395     }
396
397     public Mensagem getMensagem() {
398       try { ocupadaMyMensagem.acquire();
399       acessoMyMensagem.acquire();
400       } catch (InterruptedException e) {}
401       Mensagem s= myMensagem;
402       acessoMyMensagem.release();
403       livreMyMensagem.release();
404       return s;
405       }
```

```
406
407
408     public void run()
409     {
410           while(!bd.getTerminar()) {
411
412               switch (state) {
413
414               case receberMensagem:
415                 System.out.println("recebe");
416                 gui.write(" Recebi = " + msg + "\n");
417                 bd.addMensagem(msg);
418                 if (bd.isLigado())
419                   state = esperarTempoExecucao;
420                 else
421                   state = dormir;
422                 break;
423
424           case dormir:
425             try {
426             Thread.sleep(100);
427             //System.out.println("permits" + subAvailable.
    availablePermits());
428                 if(subAvailable.availablePermits() == 0) {
429                   state = bloqueado;
430                   break;
431                 }
432           } catch (InterruptedException e) {
433             // TODO Auto-generated catch block
434             e.printStackTrace();
435           }
436             //System.out.println("busca" + aReproduzir );
437             if(bufferCircularReproduzir.available() != 0 ||
    aReproduzir)
438             {
439               System.out.println("REPRODUZIR");
440               state = reproduzir;
441               break;
442             }
443             if (bufferCircular.available() != 0) {
444               //System.out.println(haTrabalho.availablePermits());
445               try {
446                 haTrabalho.acquire();
447                 livreMyMensagem.acquire();
448                 } catch (InterruptedException e) {}
449               Mensagem m= bufferCircular.removerElemento();
450               try { acessoMyMensagem.acquire(); }
451               catch (InterruptedException e) {}
452               myMensagem = m;
453               acessoMyMensagem.release();
454               ocupadaMyMensagem.release();
455               msg = getMensagem();
456               //System.out.println(bd.getNMensagens());
457               state = receberMensagem;
458               break;
459             }
460             else
461               if (bd.isLigado() && bd.getMensagens().size() != 0)
462                 state = esperarTempoExecucao;
463               break;
```

```
464
465
466         case esperarTempoExecucao:
467           msg = bd.getMensagens().get(0);
468           //System.out.println("esperaExec id:" + msg.getId());
469           int tipo = msg.getTipo();
470           //System.out.println("Tipo:" + tipo);
471           try {
472           switch (tipo) {
473             case RETA:
474               gui.write(" O robot avançou " + msg.getArg1() + "\n
    ");
475               bd.getRobot().reta(msg);
476               bd.getRobot().parar(msg);
477               Thread.sleep((long) ((Math.abs(msg.getArg1())) /
    0.03));
478               break;
479             case CURVARDIR:
480               gui.write(" O robot virou direita com raio = " +
    msg.getArg1() +" e angulo = " + msg.getArg2() + "\n");
481               bd.getRobot().curvarDireita(msg);
482               bd.getRobot().parar(msg);
483               Thread.sleep((long) ((msg.getArg1() * (msg.getArg2
    () * 0.017)) / 0.03));
484               break;
485             case CURVARESQ:
486               gui.write(" O robot virou esquerda com raio = " +
    msg.getArg1() +" e angulo = " + msg.getArg2() + "\n");
487                 bd.getRobot().curvarEsquerda(msg);
488                 bd.getRobot().parar(msg);
489               Thread.sleep((long) ((msg.getArg1() * (msg.getArg2
    () * 0.017)) / 0.03));
490               break;
491             case PARAR:
492               gui.write(" O robot parou \n");
493                 bd.getRobot().parar(msg);
494               Thread.sleep(100);
495               break;
496           }
497           } catch (InterruptedException e) {
498           // TODO Auto-generated catch block
499           e.printStackTrace();
500         }
501           bd.removeMensagem();
502           state = dormir;
503           break;
504
505         case bloqueado:
506             try {
507               gui.write("Estou bloqueado \n");
508               subAvailable.acquire();
509               gui.write("Desbloqueado \n");
510               state = dormir;
511               break;
512             } catch (InterruptedException e) {
513               // TODO Auto-generated catch block
514               e.printStackTrace();
515             }
516
517         case reproduzir:
```

```
518
519             try {
520               haTrabalhoG.acquire();
521             System.out.println("haT");
522             livreMyMensagem.acquire();
523           } catch (InterruptedException e) {}
524           Mensagem m= bufferCircularReproduzir.removerElemento();
525
526           try { acessoMyMensagem.acquire(); }
527           catch (InterruptedException e) {}
528           myMensagem = m;
529           acessoMyMensagem.release();
530           ocupadaMyMensagem.release();
531           msg = getMensagem();
532           //System.out.println(bd.getNMensagens());
533
534           if (msg.getTipo() != 5)
535             gui.write(" Vou reproduzir = " + msg + "\n");
536           else
537             gui.write(" Acabou a reprodução \n");
538
539                aReproduzir = true;
540                //if (bd.isLigado())
541                if(msg.tipo == 5) {
542                  aReproduzir = false;
543                  state = dormir;
544                  break;
545                }
546                bd.addMensagem(msg);
547                state = esperarTempoExecucao;
548           break;
549
550           }
551         }
552         System.out.println("sai");
553
554     }
555
556     /*public static void main(String[] args) throws
    InterruptedException
557     {
558         App_Subdito app = new App_Subdito();
559         System.out.println("A aplicação começou.");
560         app.run();
561         System.out.println("A aplicação terminou.");
562     }*/
563
564 }
565
566
567
568 Classe App_Gravar
569
570 package ptrabalho;
571
572 import java.io.BufferedReader;
573 import java.io.FileReader;
574 import java.io.IOException;
575 import java.util.concurrent.Semaphore;
576
```

```
577 public class App_Gravar extends Thread {
578
579   private Gravador gravador;
580   protected GUI_Gravar gui;
581     private BD_Gravar bd;
582     Mensagem msg;
583     Mensagem myMensagem = null;
584     private int state = 2;
585     private int counter = 0;
586     private final int reproduzir = 1;
587     private final int dormir = 2;
588     private final int gravar = 3;
589     private final int bloqueado = 4;
590     private final int ler = 5;
591
592     BufferCircular bufferCircular;
593     private Semaphore haTrabalho, livreMyMensagem,
      ocupadaMyMensagem, acessoMyMensagem, gravarAvailable;
594
595   public App_Gravar(BD_Gravar bg, Gravador g, Semaphore ga,
       BufferCircular bc, Semaphore ht)
596     {
597       gravador = g;
598         gui = new GUI_Gravar(bg);
599         bd = bg;
600         bufferCircular= bc;
601         haTrabalho = ht;
602         gravarAvailable = ga;
603         livreMyMensagem= new Semaphore(1);
604       ocupadaMyMensagem= new Semaphore(0);
605       acessoMyMensagem= new Semaphore(1);
606
607     }
608
609   public void setMensagem(Mensagem m)
610     {
611       try {
612       livreMyMensagem.acquire();
613       acessoMyMensagem.acquire();
614       } catch (InterruptedException e) {}
615       myMensagem= m;
616       acessoMyMensagem.release();
617       ocupadaMyMensagem.release();
618     }
619
620   public void bloqueado()
621   {
622     try {
623       gui.write("Estou bloqueado \n");
624       gravarAvailable.acquire();
625       gui.write("Desbloqueado \n");
626       state = dormir;
627     } catch (InterruptedException e) {
628       // TODO Auto-generated catch block
629       e.printStackTrace();
630     }
631   }
632
633   public void run()
634     {
```

```
635         while(true) {
636 
637           switch (state) {
638 
639             case dormir:
640               //System.out.println("sleep");
641               try {
642                 Thread.sleep(100);
643                 //System.out.println("permits" + reiAvailable.
    availablePermits());
644                 if(gravarAvailable.availablePermits() == 0) {
645                   state = bloqueado;
646                   break;
647                 }
648               } catch (InterruptedException e) {
649                 // TODO Auto-generated catch block
650                 e.printStackTrace();
651               }
652               if (bd.getGravar())
653                 state = gravar;
654               if(bd.getReproduzir())
655                 state = reproduzir;
656               break;
657 
658 
659             case gravar:
660               System.out.println(bd.getGravarS().availablePermits()
    );
661               gui.write("Estou a Gravar \n");
662               try {
663             bd.getGravarS().acquire();
664           } catch (InterruptedException e) {
665             // TODO Auto-generated catch block
666             e.printStackTrace();
667           }
668             gui.write("Parei de gravar \n");
669             state = dormir;
670             break;
671 
672             case reproduzir:
673                 String filePath = bd.getNome();
674                 long currentTime = 0, lastTime = 0;
675                 // Try-with-resources to automatically close
    resources (like BufferedReader)
676                 try (BufferedReader reader = new BufferedReader(new
     FileReader(filePath))) {
677                     String line;
678                     int numLine = 0;  // Initialize the line number
679                     while ((line = reader.readLine()) != null) {
680                           // Increment the line number for each
    line read
681 
682                       if(gravarAvailable.availablePermits() == 0)
683                         bloqueado();
684 
685                         // Split the line into parts using a
    delimiter (assuming it's a CSV format)
686                         String[] parts = line.split(",");
687 
688                         // Assuming the first part is an integer
```

```
689                         int tipo = Integer.parseInt(parts[0]);
690 
691                         switch (tipo) {
692                             case 1:
693                                 // Assuming numLine is declared and initialized somewhere in your code
694                                 msg = new MensagemReta(numLine, tipo, Integer.parseInt(parts[1]));
695                                 currentTime = Long.parseLong(parts[2]);
696                                 bd.addMensagem(msg);
697                                 break;
698 
699                             case 2:
700                                 // Assuming numLine is declared and initialized somewhere in your code
701                                 msg = new MensagemCurvar(numLine, tipo, Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
702                                 currentTime = Long.parseLong(parts[3]);
703                                 bd.addMensagem(msg);
704                                 break;
705 
706                             case 3:
707                                 // Assuming numLine is declared and initialized somewhere in your code
708                                  msg = new MensagemCurvar(numLine, tipo, Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
709                                 currentTime = Long.parseLong(parts[3]);
710                                 bd.addMensagem(msg);
711                                 break;
712 
713                             case 4:
714                               msg = new MensagemParar(numLine, tipo, false);
715                                 //currentTime = Long.parseLong(parts[2]);
716                                 bd.addMensagem(msg);
717                                 break;
718 
719                             case 5:
720                               msg = new MensagemVazia(numLine, 5);
721                               bd.addMensagem(msg);
722                                 break;
723                             // Handle other cases if needed
724                         }
725 
726 
727                         if(tipo != 4)
728                           gui.write("Li: " + msg + "\n");
729                         System.out.println("escreve");
730                       System.out.println("Mensagens à espera: " + bd.getMensagens().size());
731                       msg = bd.getMensagens().get(0);
732                       setMensagem(msg);
733                       try {
734 
735                         ocupadaMyMensagem.acquire();
736                         acessoMyMensagem.acquire();
```

```
737
738                         long sleepTime = currentTime - lastTime;
739                         //System.out.println(sleepTime);
740                           // Check if the sleep time is non-
    negative
741                           if (lastTime!= 0) {
742                               // Introduce a delay based on the
    time difference
743                             System.out.println("sleep");
744                               Thread.sleep(sleepTime);
745                               System.out.println("wake");
746                               if (!bd.getReproduzir())
747                                 break;
748                           }
749                           if(gravarAvailable.availablePermits() ==
    0)
750                             bloqueado();
751
752                       } catch (InterruptedException e) {}
753                         bufferCircular.inserirElemento(myMensagem);
754                         acessoMyMensagem.release();
755                         livreMyMensagem.release();
756                         haTrabalho.release();
757                         if(tipo != 4)
758                           gui.txtLog.append("Enviei = " + msg + "\n
    ");
759
760                         bd.removeMensagem();
761                         lastTime = currentTime;
762                         numLine++;
763
764                     }
765                 } catch (IOException | NumberFormatException e) {
766                     // Handle exceptions
767                     System.out.println("Error reading or parsing
    the file: " + e.getMessage());
768                     e.printStackTrace();
769                 }
770                 state = dormir;
771                 bd.setReproduzir(false);
772                 gui.tglReproduzir.setSelected(false);
773                 break;
774
775
776             case bloqueado:
777               try {
778                 gui.write("Estou bloqueado \n");
779                 gravarAvailable.acquire();
780                 gui.write("Desbloqueado \n");
781                 state = dormir;
782                 break;
783               } catch (InterruptedException e) {
784                 // TODO Auto-generated catch block
785                 e.printStackTrace();
786               }
787
788       }
789     }
790
791     }
```

```
792 }
```