Relatório da Prática 7

Vítor Barbosa

24 de julho de 2020

1 Introdução

Nesta prática há duas tarefas. A Tarefa 1 consiste em implementar um exemplo de sincronização de threads com mutex e semáforo e elucidar seu funcionamento. A Tarefa 2 consiste em aplicar esses conceitos de sincronização a uma aplicação com GUI, usando threads ou runnables, além de algum mecanismo de exclusão mútua.

2 Tarefa 1

O programa desta tarefa implementa um exemplo de treino de Fórmula 1 com threads. São criadas 10 threads, que correspondem a 10 carros. Cada carro é atribuído a uma equipe aleatória (de 7 *scuderias* possíveis) no momento de sua criação. Podem haver até 5 carros na pista, mas apenas 1 de cada equipe.

O código do programa está contido no arquivo *formula1.cpp*

formula1.cpp

```
1 #define WIN32_LEAN_AND_MEAN
2 #include <windows.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <process.h>
6 #include <conio.h>
7
8
9 // nomeamos ponteiro para uma função que retorna unsigned int e↩
     recebe um LPVOID como CAST_FUNCTION
```

```
10 // WINAPI expande para ;__stdcall, e diz respeito a uma ↩
       convenção de chamada de funções, é relevante para ↩
       interações com o SO
11 typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
12 typedef unsigned *CAST_LPDWORD; // nomeamos o tipo unsigned int↩
       * como CAST_LPDWORD
13
14 #define EQUIPES 7
15 #define MAX_CARROS_PISTA 5
16 #define NUM_CARROS 10
17 HANDLE hMutex[EQUIPES];
18 HANDLE hSemaphore;
19
20 //Declaração da função,DWORD expande para unsigned long
21 DWORD WINAPI FuncCar(LPVOID);
22
23 int main(){
24     HANDLE hThreads[NUM_CARROS];
25     DWORD dwThreadId;
26     DWORD dwExitCode = 0;
27     DWORD dwRet;
28     int nEquipe, nCar;
29     char BoxName[5];
30
31     for(nEquipe=0; nEquipe<EQUIPES;nEquipe++){
32         sprintf(BoxName, "Box %d",nEquipe);
33         hMutex[nEquipe] = CreateMutex(NULL,FALSE,BoxName);
34     }
35     //Cria um semáforo com contagem máxima MAX_CARROS_PISTA e o↩
           inicializa com o valor MAX_CARROS_PISTA.
36     hSemaphore = CreateSemaphore(NULL,MAX_CARROS_PISTA,↩
           MAX_CARROS_PISTA,"MAX_CARROS");
37
38     for(nCar = 0;nCar<NUM_CARROS;nCar++){
39         nEquipe = rand()%EQUIPES;
40         hThreads[nCar] = (HANDLE) _beginthreadex(NULL,0,(↩
               CAST_FUNCTION)FuncCar,(LPVOID)((nCar<<8)+nEquipe)↩
               ,0,(CAST_LPDWORD)&dwThreadId);
41         if(hThreads[nCar])
42             printf("Carro %2d - Scuderia %d - ID: %0x \n",nCar,↩
                   nEquipe, dwThreadId);
43     }
44
45     dwRet = WaitForMultipleObjects(NUM_CARROS,hThreads,TRUE,↩
           INFINITE);
46     for(nCar = 0;nCar,NUM_CARROS; nCar++){
47         dwRet = GetExitCodeThread(hThreads[nCar],&dwExitCode);
48         CloseHandle(hThreads[nCar]);
49     }
```

```
50
51      for(nEquipe=0; nEquipe< EQUIPES; nEquipe++) CloseHandle(↩
            hMutex[nEquipe]);
52
53      CloseHandle(hSemaphore);
54      printf("\nPressione Qualquer tecla para terminar\n");
55
56  }
57
58  DWORD WINAPI FuncCar(LPVOID id){
59      long lOldValue;
60      int nCar, iTeam;
61
62      iTeam = (DWORD)id % 256;
63      nCar = (DWORD)id / 256;
64
65      for(int i=0;i<3;i++){
66          printf("Carro %d da equipe %d quer treindar... volta %d↩
                \n",nCar,iTeam, i);
67
68      //Aguardar a disponibilidade do Mutex
69      WaitForSingleObject(hMutex[iTeam],INFINITE);
70
71      //Aguardar disponibilidade do semáforo para decrementá-lo
72      WaitForSingleObject(hSemaphore,INFINITE);
73
74      printf("Carro %d da equipe %d treinando ... volta %d \n", ↩
            nCar, iTeam, i);
75      Sleep(100*(rand()%10));
76
77      //Incrementar o semáforo
78      ReleaseSemaphore(hSemaphore,1,&lOldValue);
79
80      //Liberar o mutex
81      ReleaseMutex(hMutex);
82      printf("Carro %d da equipe %d acabou de treinar %d \n",nCar↩
            ,iTeam,i);
83      }
84      // Terminar execução da Thread com código 0
85      _endthreadex(0);
86      return(0);
87  }
```

O que ocorre no Mutex

Há um array de 7 mutexes, um por equipe. Quando um carro entra na pista, isto é, sua thread é executada pelo método *beginthreadex*, ele tenta adquirir

o mutex "M"de sua equipe. Se já houver outro carro da mesma equipe na pista, ele precisa aguardar até que o carro que já está treinando termine seu treino e libere o mutex M. Lembre-se: só a thread que adquire o mutex pode liberá-lo [1].

Uma vez liberado, o carro solicitante adquire o mutex M, impedindo então qualquer outro carro da mesma equipe de adquiri-lo. Carros de outras equipes podem treinar concomitantemente, pois adquirem outro mutex (diferente de M) do array.

O que acontece no semáforo

O semáforo é inicializado com o valor máximo de carros na pista. Quando a thread de um carro é executada, ela tenta decrementar o semáforo. Dkjistra define [3] para o decremento a operação P no semáforo S:

```
P(x){      while(S<x);                    S -= x;  }
```

No caso, x = 1 (temos um decremento unitário), e P aguarda (bloqueia a execução da thread) até que S seja positivo. Quando P é realizada, o carro executa seu treino. No fim, a thread realiza a operação V:

```
V(x){      S += x;  }
```

No caso, x=1 e temos um incremento unitário. Na prática, se definimos S=5, isso implica que não podemos ter mais que 5 carros na pista.

3 Tarefa 2

Nesta tarefa, implementamos um exemplo de treino de Fórmula 1 com GUI. Os requerimentos principais (há outros não listados) são os seguintes:

- As classes Carro e Moto herdam de veículo
- Até 5 veículos na pista
- Carro dá até 3 voltas e Moto dá 2 voltas

Para implementar o paralelismo, a solução escolhida foi usar a *QThreadPool*.É preciso herdar da classe *QRunnable* e implementar o método *run*.

A Classe QThreadPool gerencia a criação de threads, reciclando-as quando necessário para evitar o alto custo computacional de criar e destruir várias threads. Uma comparação detalhada das soluções do Qt para threads pode ser encontrada em sua Documentação [2].

Para implementar a exclusão mútua, a solução escolhida foram os semáforos disponíveis na classe *QSemaphore*.

Dessa vez, optou-se por descrever os Widgets da interface gráfica completamente no código fonte, sem utilizar o editor WYSIWYG do Qt Creator. A aplicação conta com as classes Vehicle, Moto, Carro e MainWindow, além do arquivo main.

vehicle.h

Este arquivo de cabeçalho contém as declarações da classe Vehicle e suas subclasses Moto e Carro.

```
#ifndef VEHICLE_H
#define VEHICLE_H
#include <QRunnable>
#include <QString>
#include <QSemaphore>
#include <QObject>


class Vehicle : public QObject,public QRunnable
{
    //Precisamos de um sinal para as threads
    Q_OBJECT
public:
    const static int MAX_COUNT = 5;
    Vehicle(QString equipe, int lapInterval_ms);
    virtual void run() = 0; //função puramente virtual, ↩
       forçamos a implementação na subclasse
    void requestAbort(); // mata gentilmente a thread
    static int getCount(){return count;} //contagem de threads

    ~Vehicle();
protected:
    QString equipe;
    static QSemaphore sem;
    int lapInterval;
    bool abortRequested;
    static int count;
    virtual void finish()=0;

signals:
    void sendMsg(QString msg);

};

class Moto: public Vehicle{
public:
    Moto(QString equipe, int lapInterval_ms);
    const static int MAX_LAPS = 2;
```

```
38      void run() override;
39      void init();//chame para inicializar a classe após conectar↩
            seus sinais aos devidos slots
40  private:
41      void finish(); //emite mensagem de término
42  };
43  class Carro: public Vehicle{
44  public:
45      Carro(QString equipe, int lapInterval_ms);
46      const static int MAX_LAPS = 3;
47      void run() override;
48      void init();//chame para inicializar a classe após conectar↩
            seus sinais aos devidos slots
49  private:
50      void finish(); //emite mensagem de término
51  };
52
53  #endif // VEHICLE_H
```

vehicle.cpp

```
1   #include "vehicle.h"
2   #include <QThread>
3   #include <QTextStream>
4
5
6   //Inicialização de membro estático
7   QSemaphore Vehicle::sem(MAX_COUNT);
8   int Vehicle::count = 0;
9
10  Vehicle::Vehicle(QString equipe, int t):equipe(equipe),↩
        lapInterval(t),abortRequested(false){
11      count++;
12      //Recomendado pela Documentação, marca para deleção após ↩
            término
13      setAutoDelete(true);
14  }
15
16  Vehicle::~Vehicle(){
17      count--;
18      emit sendMsg(""); // mensagem em branco apenas para emitir ↩
            um sinal na destruição
19  }
20
21  void Vehicle::requestAbort(){
22      abortRequested = true;
23      finish();
```

```
24 }
25
26 Moto::Moto(QString equipe,int t):Vehicle(equipe,t){}
27
28 void Moto::init(){
29     emit  sendMsg(QString("Equipe de Moto ").append(equipe).↩
          append( " criada\n"));
30 }
31
32 Carro::Carro(QString equipe, int t):Vehicle(equipe,t){}
33
34 void Carro::init(){
35     emit sendMsg(QString("Equipe de Carro ").append(equipe).↩
          append( " criada\n"));
36 }
37
38 void Moto::finish(){
39     char s[] = "Equipe de Moto %s finalizou o treino.          ↩
          \n";
40     sprintf(s,s,equipe.toStdString().c_str());
41     emit sendMsg(s);
42 }
43
44 void Moto::run(){
45
46     for(int i=0;i<MAX_LAPS;i++){
47         if(abortRequested) return;
48         //tenta adquirir o semáforo, bloqueia se não der
49         sem.acquire(1);
50         char s[] = "Moto da Equipe %s treinando na volta %d ↩
                       \n";
51         sprintf(s,s,equipe.toStdString().c_str(),i);
52         emit sendMsg(s);
53         //Para melhorar o tempo de resposta, vamos fazer vários↩
               delays pequenos em vez de um grande
54         for(int t=0;t<lapInterval;t+=100){
55             if(abortRequested) {
56                 sem.release(1);
57                 return;
58             }
59             else QThread::msleep(lapInterval>100?lapInterval↩
                   /100:lapInterval);
60         }
61         sem.release(1);
62     }
63     finish();
64 }
65
66 void Carro::finish(){
```

```
67      char s[] = "Equipe de Carro %s finalizou o treino. ↩
                    \n";
68      sprintf(s,s,equipe.toStdString().c_str());
69      emit sendMsg(s);
70  }
71
72  void Carro::run(){
73
74      for(int i=0;i<MAX_LAPS;i++){
75          if(abortRequested) return;
76          sem.acquire(1);
77          char s[] = "Carro da Equipe %s treinando na volta %d ↩
                       \n";
78          sprintf(s,s,equipe.toStdString().c_str(),i);
79          emit sendMsg(s);
80          //Para melhorar o tempo de resposta, vamos fazer vários↩
                delays pequenos em vez de um grande
81          for(int t=0;t<lapInterval;t+=100){
82              if(abortRequested) {
83                  sem.release(1);
84                  return;
85              }
86              else QThread::msleep(lapInterval>100?lapInterval↩
                    /100:lapInterval);
87          }
88          sem.release(1);
89      }
90      finish();
91  }
```

mainwindow.h

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include <QtWidgets>
6  #include <vector>
7  #include "vehicle.h"
8
9  class MainWindow : public QMainWindow
10 {
11     Q_OBJECT
12
13 public:
14     MainWindow(QWidget *parent = nullptr);
15     ~MainWindow();
```

```
16      void vehicleMsg(QString msg);
17  private:
18      QLabel *lbDisplay, *lbCount;
19      QLineEdit *inIdCarro, *inRestCarro, *inIdMoto, *inRestMoto;
20      QPushButton *btnCriaCarro, *btnCriaMoto, *btnStopAll;
21      QString strCount;
22
23
24      std::vector<Vehicle *> vehicles;
25
26      void criarMoto();
27      void criarCarro();
28      void stopAll();
29      void updateVehicleCount();
30
31
32  };
33  #endif // MAINWINDOW_H
```

mainwindow.cpp

Este arquivo contém a implementação da classe da janela principal. Os Widgets são declarados e instanciados no construtor, permitindo que a janela seja construída somente com código.

```
1   #include "mainwindow.h"
2
3   MainWindow::MainWindow(QWidget *parent)
4       : QMainWindow(parent)
5   {
6       //O Qt exige que uma QMainWindow tenha pelo menos um Widget↩
            Central;
7       QWidget *centralWidget = new QWidget();
8
9       //É ao central widget que devemos adicionar o layout
10      QHBoxLayout *hBox = new QHBoxLayout(centralWidget);
11      // centralWidget->setLayout(hBox);
12
13      this->setCentralWidget(centralWidget);
14      lbDisplay = new QLabel("Aguardando início do treino...\n");
15      lbDisplay->setAlignment(Qt::AlignTop);
16      hBox->addWidget(lbDisplay);
17      //"Esticar" o último widget adicionado
18      hBox->addStretch(1);
19
20      QVBoxLayout *vBox = new QVBoxLayout();
21      hBox->addLayout(vBox);
22      vBox->setAlignment(Qt::AlignTop);
```

```
23      vBox->setSpacing(20);
24
25      QString strId = "ID da Equipe";
26      QString strRest = "Descanso (ms)";
27      QString strCriar = "Criar";
28      QString strStopAll = "Finalizar Todos os Treinos";
29      strCount = "Total de Equipes Criadas: ";
30      QString strLimpar = "Limpar Texto";
31      QGroupBox *boxCarro = new QGroupBox("Equipe de Carro");
32      QVBoxLayout *v = new QVBoxLayout();
33      boxCarro->setLayout(v);
34
35      v->addWidget(btnCriaCarro = new QPushButton(strCriar));
36      v->addWidget(new QLabel(strId));
37      v->addWidget(inIdCarro=new QLineEdit());
38      v->addWidget(new QLabel(strRest));
39      v->addWidget(inRestCarro=new QLineEdit());
40
41      QGroupBox *boxMoto = new QGroupBox("Equipe de Moto");
42      //Como os ponteiros são passados por valor, é seguro ↩
            reutilizar v
43      //Isto é, passamos para as funções anteriores só valor do ↩
            endereço de memória apontado pela variável
44      v = new QVBoxLayout();
45      boxMoto->setLayout(v);
46
47      v->addWidget(btnCriaMoto = new QPushButton(strCriar));
48      v->addWidget(new QLabel(strId));
49      v->addWidget(inIdMoto=new QLineEdit());
50      v->addWidget(new QLabel(strRest));
51      v->addWidget(inRestMoto=new QLineEdit());
52
53      vBox->setMargin(20);
54      vBox->addWidget(boxCarro);
55      vBox->addWidget(boxMoto);
56      vBox->addWidget(lbCount = new QLabel(strCount));
57      vBox->addWidget(btnStopAll = new QPushButton(strStopAll));
58      QPushButton *btnLimpar = new QPushButton(strLimpar);
59      vBox->addWidget(btnLimpar);
60
61      connect(btnStopAll,&QPushButton::clicked,this,&MainWindow::↩
            stopAll);
62      connect(btnCriaMoto,&QPushButton::clicked,this,&MainWindow↩
            ::criarMoto);
63      connect(btnCriaCarro,&QPushButton::clicked,this,&MainWindow↩
            ::criarCarro);
64      connect(btnLimpar, &QPushButton::clicked,this,[&](){this->↩
            lbDisplay->setText("");});
65
```

```
66      inIdMoto->setText("1");
67      inIdCarro->setText("1");
68      inRestMoto->setText("10000");
69      inRestCarro->setText("10000");
70
71  }
72
73  void MainWindow::updateVehicleCount(){
74      QString s="";
75      s.append(strCount).append(QString::number(Vehicle::getCount↩
            ()));
76      lbCount->setText(s);
77  }
78
79  void MainWindow::criarCarro(){
80     // QMessageBox::information(this,"carro","click");
81
82      Carro *m =new Carro(inIdCarro->text(),inRestCarro->text().↩
            toInt());
83      connect(m,&Vehicle::sendMsg,this,&MainWindow::vehicleMsg);
84      vehicles.push_back(m);
85      m->init();
86
87      QThreadPool::globalInstance()->start(m);
88      inIdCarro->setText(QString::number(inIdCarro->text().toInt↩
            ()+1));
89      updateVehicleCount();
90
91  }
92  void MainWindow::criarMoto(){
93
94      Moto *m =new Moto(inIdMoto->text(),inRestMoto->text().toInt↩
            ());
95      connect(m,&Vehicle::sendMsg,this,&MainWindow::vehicleMsg);
96      vehicles.push_back(m);
97      m->init();
98
99       QThreadPool::globalInstance()->start(m);
100      inIdMoto->setText(QString::number(inIdMoto->text().toInt()↩
             +1));
101      updateVehicleCount();
102
103 }
104 void MainWindow::stopAll(){
105     for(Vehicle *v:vehicles)
106         if(v!=nullptr) v->requestAbort();
107     vehicles.clear();
108 }
109
```

```
110
111 void MainWindow::vehicleMsg(QString msg){
112     lbDisplay->setText(lbDisplay->text().append(msg));
113     updateVehicleCount();
114 }
115
116 MainWindow::~MainWindow()
117 {
118 }
```

4 Execução

Tarefa 1

O teste da aplicação da Tarefa 1 pode ser visto nas figuras 1 e 2. A aplicação funciona como esperado: são criados 10 veículos, eles efetuam os treinos seguindo os requerimentos, e o programa termina.

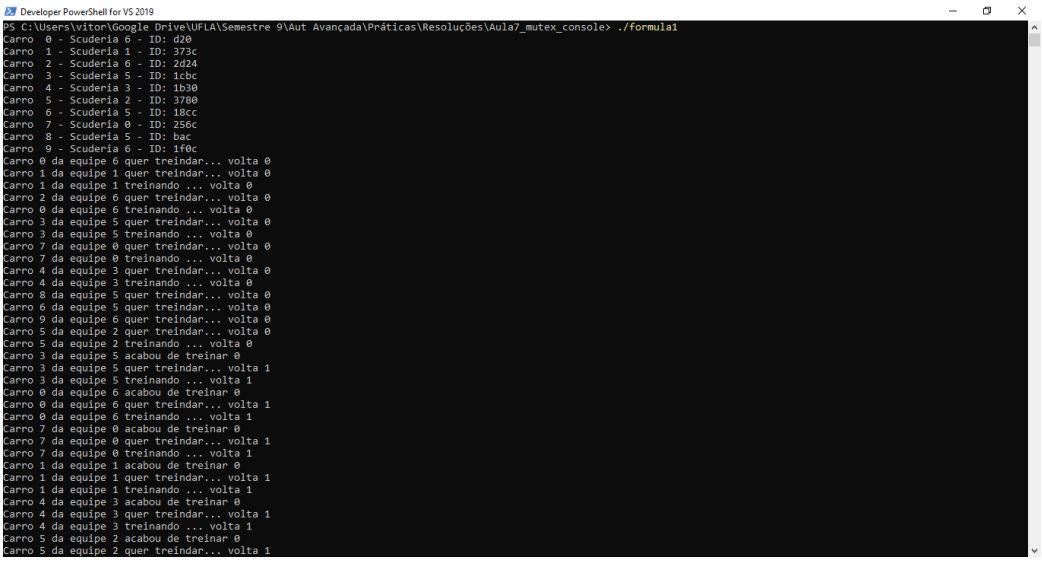

Figura 1: Teste do Programa da Tarefa 1 - Parte Superior

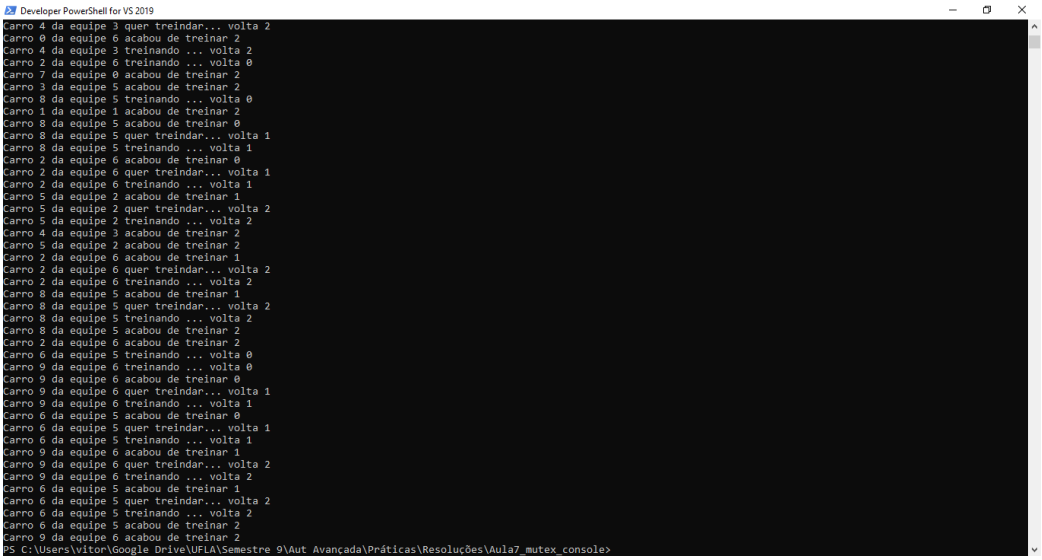

Figura 2: Teste do Programa da Tarefa 1 - Parte Superior

Tarefa 2

Aqui é apresentado o teste da aplicação com GUI. A figura 3 mostra o estado inicial do programa.

As figuras 4 e 5 mostram testes com tempos de descanso de 10s e 5s por volta, respectivamente.

Pode-se notar, especialmente na figura 4 que nunca há mais de 5 veículos treinando.

A figura 6 mostra o comportamento do botão ”Finalizar Todos os Treinos”.

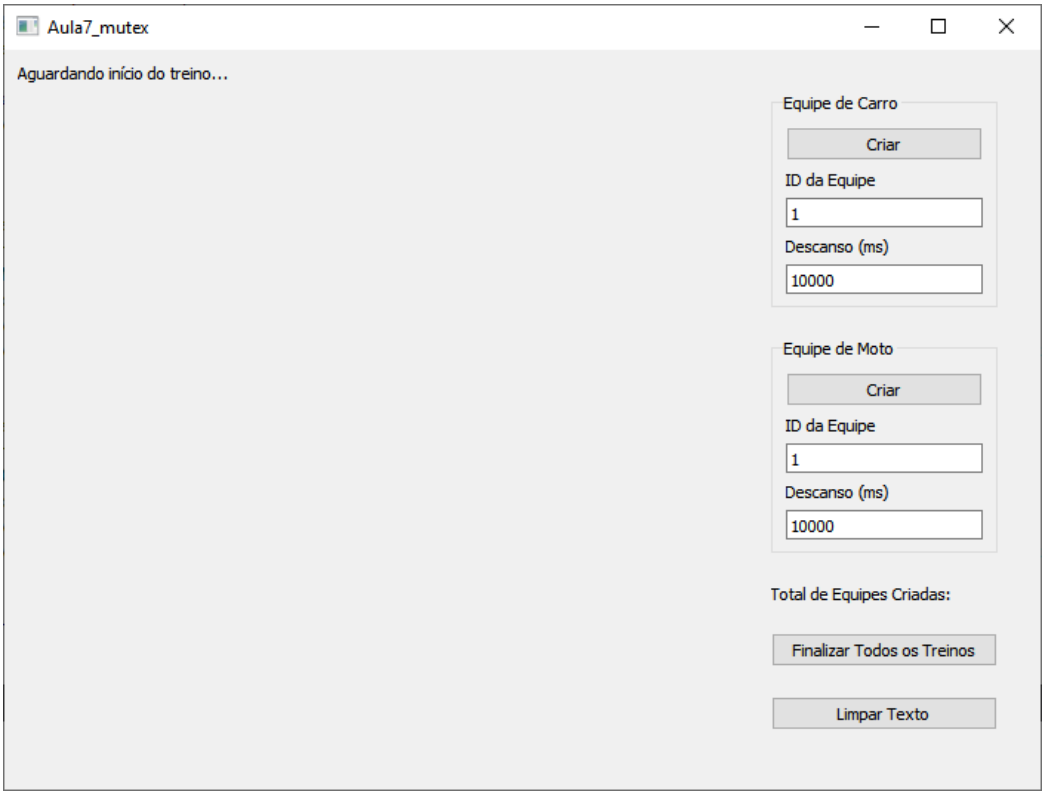

Figura 3: Janela inicial do Programa da Tarefa 2

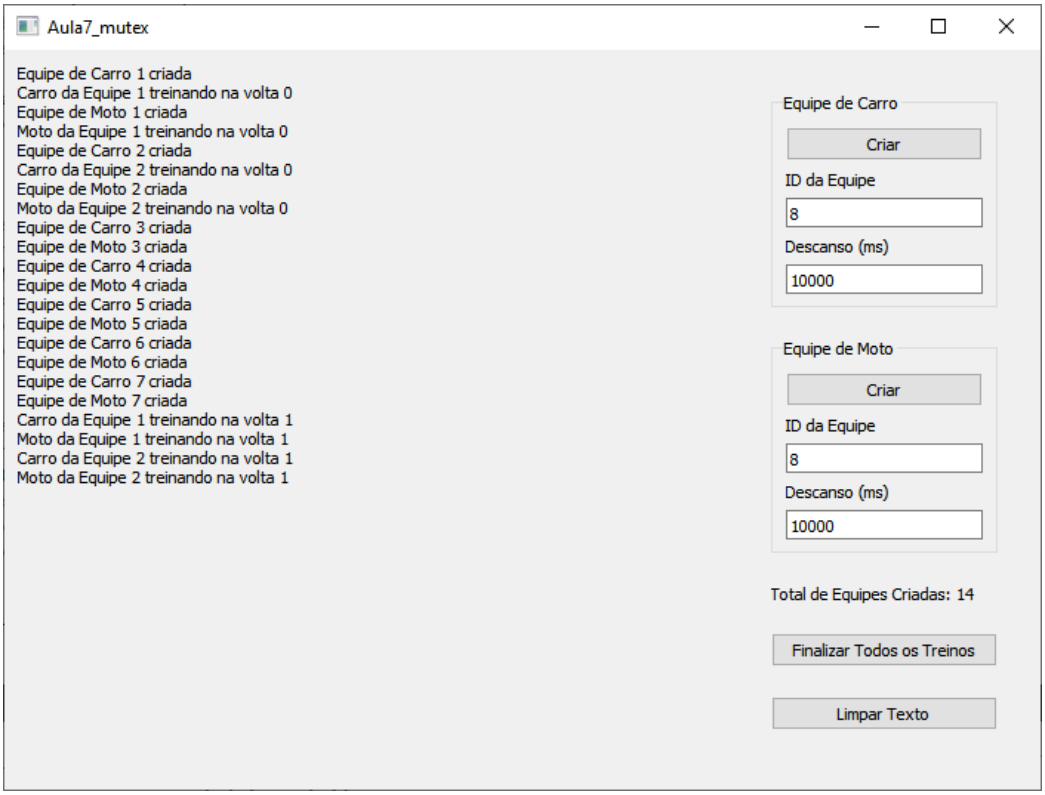

Figura 4: Teste com 10s por volta

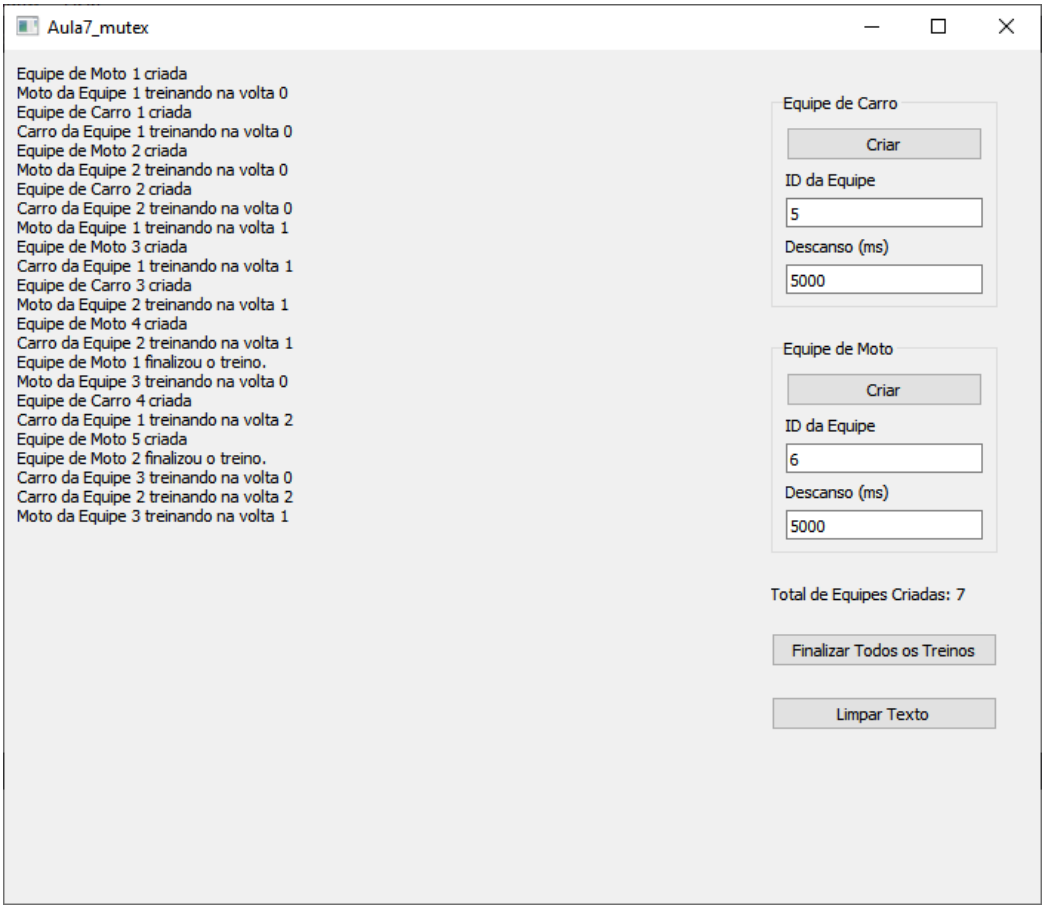

Figura 5: Teste com 5s por volta

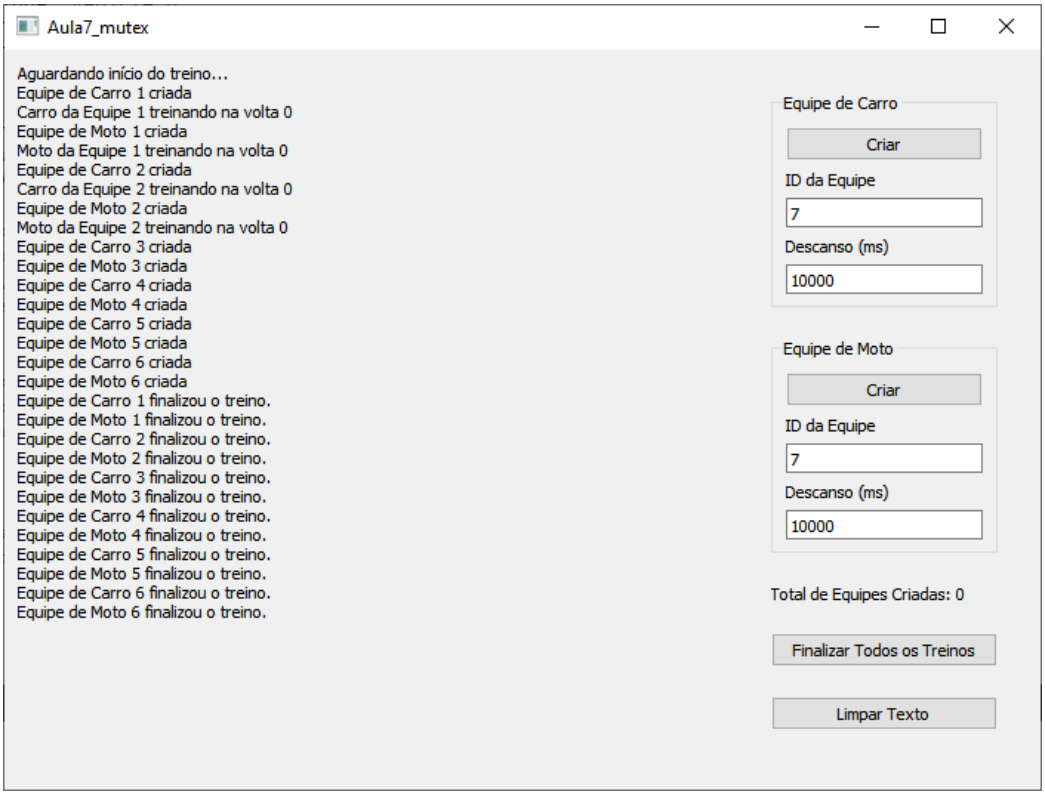

Figura 6: Finalização Forçada dos Treinos

Referências

[1] *Difference between binary semaphore and mutex*. URL: https://stackoverflow.com/questions/62814/difference-between-binary-semaphore-and-mutex.

[2] *Multithreading Technologies in Qt*. URL: https://doc.qt.io/qt-5/threads-technologies.html.

[3] Wikipédia. *Semaphore (programming)*. URL: https://en.wikipedia.org/wiki/Semaphore_(programming).