

# Relatório da Prática 6

Vítor Barbosa

4 de agosto de 2020

## 1 Introdução

Essa prática é uma introdução às Threads. Faremos 3 programas: um exemplo de threads em C++ usando a biblioteca padrão; um exemplo com a API do Windows (Win32); finalmente, um aplicativo com interface gráfica (GUI).

## 2 Código

### 2.1 Threads com a Standard Library

O C++ conta, desde a versão C++11 com suporte nativo à threads pela *std::thread*. Na opinião do autor, a *std::thread* possui uma API muito mais clara e simples que versão Win32.

Contudo, é interessante notar que aparentemente não há um método na biblioteca padrão para forçar o término de uma única thread, o que pode ser feito com Win32.

Nosso programa conta com uma classe *BabyThread* que implementa uma thread de exemplo, além do arquivo *main* que instancia várias *BabyThreads* com tempos de execução aleatórios.

**babythread.h**

---

```
1 #ifndef BABYTHREAD_H
2 #define BABYTHREAD_H
3
4 #include <iostream>
5 #include <thread>
6
7 class BabyThread{
8     public:
```

```

9      // Uma variável só pode ser definida aqui se for static ←
      const int ou um enum
10     // Para outras constantes, utilize a lista
11     static const int MAX_THREAD_COUNT = 20;
12     BabyThread();
13     std::thread spawn (int tId);
14
15     private:
16     void showMsg(int tId);
17 };
18 #endif

```

---

## babythread.cpp

---

```

1  #include "babythread.h"
2  #include <chrono>
3
4  using namespace std;
5  BabyThread::BabyThread(){}
6
7  thread BabyThread::spawn(int tId){ // Spawn é chamado ao rodar ←
      a thread
8      return thread([&]() {showMsg(tId);});
9  }
10
11 void BabyThread::showMsg(int tId){ // Dorme um tempo aleatório ←
      e exibe mensagem no fim
12     this_thread::sleep_for(chrono::milliseconds(rand()%5000));
13     cout<<"Esta eh a thread "<<tId<<"\n";
14 }

```

---

## main.cpp

---

```

1  #include "babythread.h"
2
3  using namespace std;
4
5  int main (int argc, char **argv){
6      BabyThread *thr = new BabyThread();
7
8      thread *threads = new thread[thr->MAX_THREAD_COUNT];
9
10     for(int i=0;i<thr->MAX_THREAD_COUNT;i++)
11         threads[i] = thr->spawn(i);
12
13     for(int i=0;i<thr->MAX_THREAD_COUNT;i++)

```

```

14         threads[i].join();
15
16     return 0;
17 }

```

---

## 2.2 Threads com a Api do Windows

O código aqui foi provido pelo professor. Ele instancia uma thread que executa e periodicamente exibe um caractere . na tela por 10 segundos.

---

```

1 #define WIN32_LEAN_AND_MEAN // Só os .h mais básicos serão ←
   incluídos.
2 #include <iostream>
3 #include <windows.h>
4 #include <process.h>
5 typedef unsigned (WINAPI *PBEGINTHREADEX_THREADFUNC)(LPVOID ←
   lpThreadParameter);
6 typedef unsigned *PBEGINTHREADEX_THREADID;
7 // This ThreadObject is created by a thread that wants to start←
   another
8 // thread. All public member functions except ThreadFunc() are ←
   called by
9 // that original thread. The virtual function ThreadMemberFunc←
   () is the
10 // start of the new thread.
11 class ThreadObject
12 {
13 public:
14     ThreadObject(); // Construtor
15     void StartThread();
16     void WaitForExit();
17     static DWORD WINAPI ThreadFunc(LPVOID param);
18     void SetKillThread(bool kill);
19     bool GetKillThread();
20 protected:
21     virtual DWORD ThreadMemberFunc();
22     HANDLE m_hThread; // Handle para thread criada
23     DWORD m_ThreadId; // Identificador da thread
24     bool KillThread;
25 };
26 ThreadObject::ThreadObject() // Inicializa membros privados da ←
   classe
27 {
28     this->m_hThread = NULL;
29     this->m_ThreadId = 0;
30     this->KillThread = false;
31 }

```

```

32 void ThreadObject::StartThread()
33 {
34     m_hThread = (HANDLE)_beginthreadex(NULL, 0,
35     (PBEGINTHREADEX_THREADFUNC) ThreadObject::ThreadFunc,
36     (LPVOID)this, // passa pointer para objeto como parâmetro
37     0, (PBEGINTHREADEX_THREADID) &m_ThreadId );
38     if (m_hThread) {
39         std::cout<< "Thread launched\n";
40     }
41 }
42 void ThreadObject::WaitForExit() // Espera fim da thread
43 {
44     WaitForSingleObject(m_hThread, INFINITE);
45     CloseHandle(m_hThread);
46 }
47 void ThreadObject::SetKillThread(bool kill) {
48     this->KillThread = kill;
49 }
50 bool ThreadObject::GetKillThread() {
51     return this->KillThread;
52 }
53 // This is a static member function. Unlike C static functions, ←
54 // you only
55 // place the static declaration on the function declaration in ←
56 // the class, not
57 // on its implementation. Static member functions have no "this ←
58 // " pointer, but
59 // do have access rights.
60 DWORD WINAPI ThreadObject::ThreadFunc(LPVOID param) {
61     // Use the param as the address of the object
62     ThreadObject* pto = (ThreadObject*)param;
63     // Call the member function. Since we have a proper object ←
64     // pointer,
65     // even virtual functions will be called properly.
66     return pto->ThreadMemberFunc();
67 }
68 // This above function ThreadObject::ThreadFunc() calls this ←
69 // function after the
70 // thread starts up.
71 DWORD ThreadObject::ThreadMemberFunc()
72 // Função que desempenhará as funções da thread
73 {
74     // Do something useful ...
75     while(!this->GetKillThread())
76     {
77         std::cout << ".";
78         Sleep(1000);
79     }
80     return 0;

```

```

76 }
77 void main()
78 {
79 ThreadObject obj;
80 obj.StartThread(); // Cria thread
81 Sleep(10000);
82 obj.SetKillThread(true);
83 obj.WaitForExit(); // Espera fim da thread
84 }

```

## 2.3 Threads com Interface Gráfica

O programa cria 2 threads, que podem ser pausadas, reativadas ou terminadas. Cada thread retorna um texto a cada 3 segundos de sua execução. Primeiro, foi criado o formulário da janela como usual. Ele pode ser visto na figura 1.

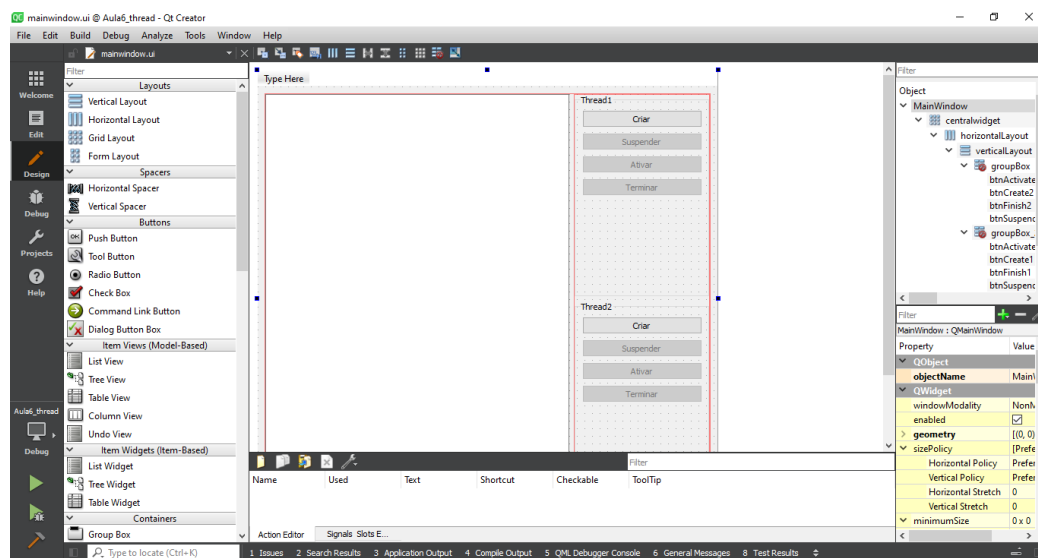


Figura 1: Criação da Janela

O código fonte ficou dividido entre as classes MainWindow e TextThread, além do arquivo main.

### textthread.h

```

1 #ifndef TEXTTHREAD_H
2 #define TEXTTHREAD_H
3 #include <QThread>

```

```

4
5
6 class TextThread : public QThread
7 {
8     Q_OBJECT
9
10 public:
11     TextThread(int threadId);
12     void run() override;
13     //Funções inline, já que são bem simples
14     void resume(){pauseRequested = false;}
15     void pause(){pauseRequested = true;}
16
17 signals:
18     void resultReady(const QString &s); //emite atualizações ←
        periódicas da execução
19     void finished();
20
21 private:
22     int id;
23     bool pauseRequested;
24 };
25
26 #endif // TEXTTHREAD_H

```

---

## textthread.cpp

---

```

1 #include "textthread.h"
2
3 TextThread::TextThread(int threadId):id(threadId)
4 {
5     pauseRequested = false;
6 }
7
8 // A thread retorna uma string a cada 3 segundos. Sua execução ←
    pode ser pausada ou interrompida
9 void TextThread::run(){
10     while(true && !isInterruptionRequested()){
11         msleep(3000);
12         if(!pauseRequested){
13             if(isInterruptionRequested()) return;
14             emit resultReady(QString("Executando loop da thread←
                %1 \n").arg(id));
15         }
16         else msleep(333); // Sem sleep a thread fica em loop ←
            contínuo e usa muita CPU quando em pausa
17     }
18     //emit finished();

```

19 }

---

## mainwindow.h

---

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QMainWindow>
5 #include "textthread.h"
6
7 QT_BEGIN_NAMESPACE
8 namespace Ui { class MainWindow; }
9 QT_END_NAMESPACE
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18
19 private slots:
20     void on_btnCreate1_clicked();
21     void on_btnSuspend1_clicked();
22     void on_btnActivate1_clicked();
23     void on_btnFinish1_clicked();
24     void on_btnCreate2_clicked();
25     void on_btnSuspend2_clicked();
26     void on_btnActivate2_clicked();
27     void on_btnFinish2_clicked();
28
29 private:
30     Ui::MainWindow *ui;
31     TextThread *thread1;
32     TextThread *thread2;
33
34     void handleThreadResult(QString result);
35     void disableGroup(int group);
36 };
37 #endif // MAINWINDOW_H
```

---

## mainwindow.cpp

---

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
```

```

3
4 using namespace std;
5
6 MainWindow::MainWindow(QWidget *parent)
7     : QMainWindow(parent)
8     , ui(new Ui::MainWindow)
9 {
10     ui->setupUi(this);
11     thread1 = new TextThread(1);
12     thread2 = new TextThread(2);
13
14     connect(thread1,&TextThread::resultReady,this,&MainWindow::←
        handleThreadResult);
15     connect(thread2, &TextThread::resultReady,this,&MainWindow←
        ::handleThreadResult);
16
17     //Recomendado pela Documentação do Qt. Marca a Thread para ←
        deleção após a execução
18     connect(thread1,&TextThread::finished,this,&QObject::←
        deleteLater);
19     connect(thread2,&TextThread::finished,this,&QObject::←
        deleteLater);
20     //connect(ui->btnCreate1,&QPushButton::clicked,this,[&]() {←
        this->thread1->start();});
21
22
23 }
24
25 MainWindow::~MainWindow()
26 {
27     delete ui;
28 }
29
30 void MainWindow::handleThreadResult(QString result){
31     QString lastTxt = ui->textEdit->toPlainText();
32     ui->textEdit->setText(lastTxt.append(result));
33 }
34
35 void MainWindow::disableGroup(int groupId){
36     //Armazenamos os botões num array bidimensional para ←
        desabilitar vários de uma vez
37     QPushButton* arr[2][4] = {{(ui->btnCreate1),(ui->←
        btnSuspend1),(ui->btnActivate1),(ui->btnFinish1)},
38                               {(ui->btnCreate2),(ui->←
        btnSuspend2),(ui->←
        btnActivate2),(ui->btnFinish2←
        )}};
39
40     //Desabilita uma coluna inteira do array

```



```

41     for(QPushButton* btn:arr[groupId-1])
42         btn->setEnabled(false);
43 }
44
45 void MainWindow::on_btnCreate1_clicked()
46 {
47     thread1->start();
48     disableGroup(1);
49     ui->btnSuspend1->setEnabled(true);
50     ui->btnFinish1->setEnabled(true);
51 }
52
53 void MainWindow::on_btnSuspend1_clicked()
54 {
55     thread1->pause();
56     disableGroup(1);
57     ui->btnActivate1->setEnabled(true);
58     ui->btnFinish1->setEnabled(true);
59 }
60
61 void MainWindow::on_btnActivate1_clicked()
62 {
63     thread1->resume();
64     disableGroup(1);
65     ui->btnSuspend1->setEnabled(true);
66     ui->btnFinish1->setEnabled(true);
67 }
68
69 void MainWindow::on_btnFinish1_clicked()
70 {
71     thread1->requestInterruption();
72     disableGroup(1);
73     ui->btnCreate1->setEnabled(true);
74 }
75 void MainWindow::on_btnCreate2_clicked()
76 {
77     thread2->start();
78     disableGroup(2);
79     ui->btnSuspend2->setEnabled(true);
80     ui->btnFinish2->setEnabled(true);
81 }
82
83 void MainWindow::on_btnSuspend2_clicked()
84 {
85     thread2->pause();
86     disableGroup(2);
87     ui->btnActivate2->setEnabled(true);
88     ui->btnFinish2->setEnabled(true);
89 }

```

```

90
91 void MainWindow::on_btnActivate2_clicked()
92 {
93     thread2->resume();
94     disableGroup(2);
95     ui->btnSuspend2->setEnabled(true);
96     ui->btnFinish2->setEnabled(true);
97 }
98
99 void MainWindow::on_btnFinish2_clicked()
100 {
101     thread2->requestInterruption();
102     disableGroup(2);
103     ui->btnCreate2->setEnabled(true);
104 }

```

---

## main.cpp

Código gerado pelo Qt Creator, igual ao das práticas anteriores.

## 3 Resultados

### Programa com a Standard Libray

Veja na figura 2 que 20 threads são criadas e retornam em tempos aleatórios.

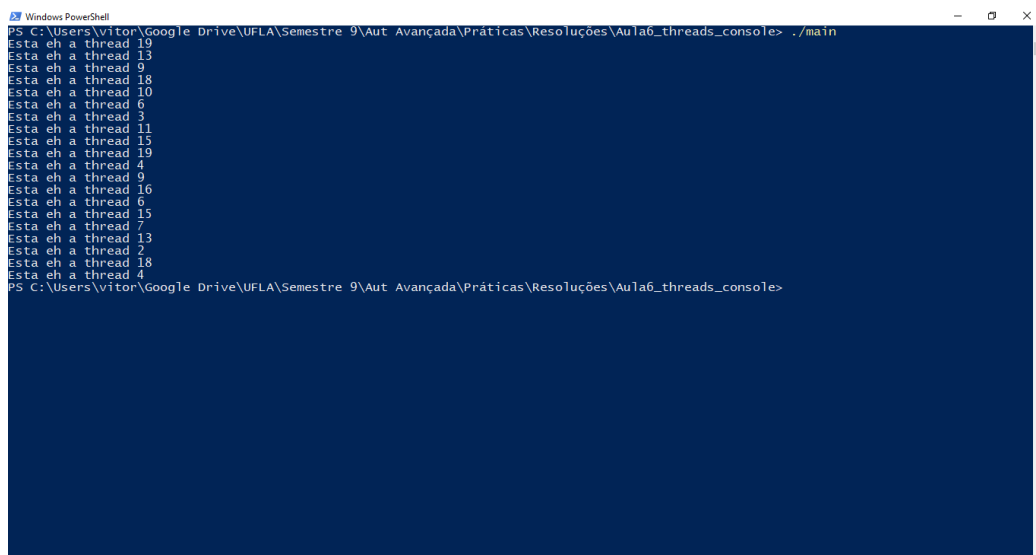
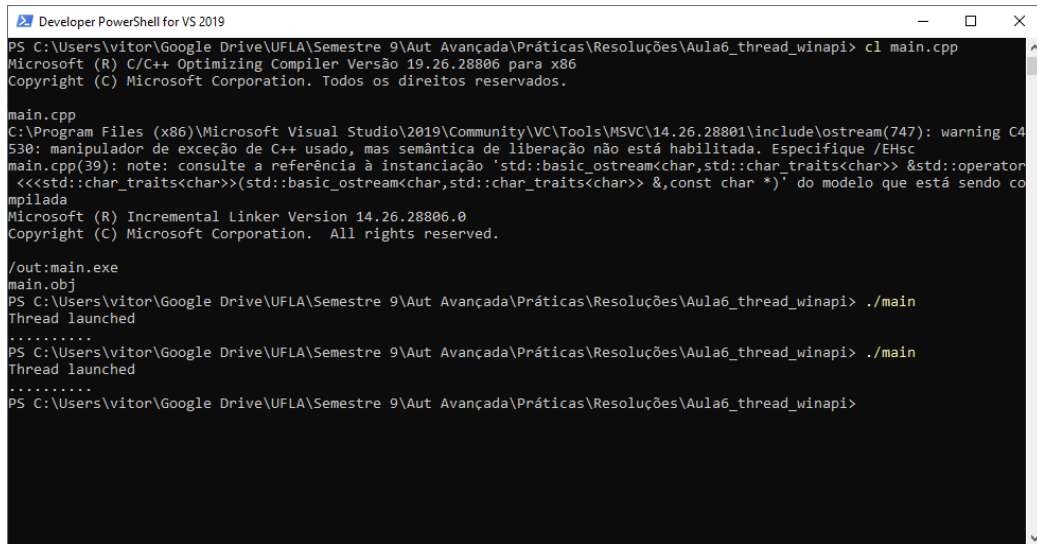


Figura 2: Teste do Programa com StdLib

## Programa com a Api do Windows

Veja na figura 3 que uma thread é criada e executada por 10s, então é forçosamente terminada. No início do console há também a saída do compilador CLang.



```
Developer PowerShell for VS 2019
PS C:\Users\vitor\Google Drive\UFLA\Semestre 9\Aut Avançada\Práticas\Resoluções\Aula6_thread_winapi> cl main.cpp
Microsoft (R) C/C++ Optimizing Compiler Versão 19.26.28806 para x86
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

main.cpp
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.26.28801\include\ostream(747): warning C4
530: manipulador de exceção de C++ usado, mas semântica de liberação não está habilitada. Especifique /EHsc
main.cpp(39): note: consulte a referência à instância 'std::basic_ostream<char,std::char_traits<char>> &std::operator
<<<std::char_traits<char>>(std::basic_ostream<char,std::char_traits<char>> &,const char *)' do modelo que está sendo co
mpilada
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:main.exe
main.obj
PS C:\Users\vitor\Google Drive\UFLA\Semestre 9\Aut Avançada\Práticas\Resoluções\Aula6_thread_winapi> ./main
Thread launched
.....
PS C:\Users\vitor\Google Drive\UFLA\Semestre 9\Aut Avançada\Práticas\Resoluções\Aula6_thread_winapi> ./main
Thread launched
.....
PS C:\Users\vitor\Google Drive\UFLA\Semestre 9\Aut Avançada\Práticas\Resoluções\Aula6_thread_winapi>
```

Figura 3: Teste do Programa com Win32

## Programa com o Qt

No teste do programa com GUI, são criadas duas threads, e seu texto é exibido a cada atualização (de 3 em 3 segundos) na tela, como mostra a figura 4.

Na figura 5, a thread 1 é terminada e a thread 2 é suspensa após algum tempo.

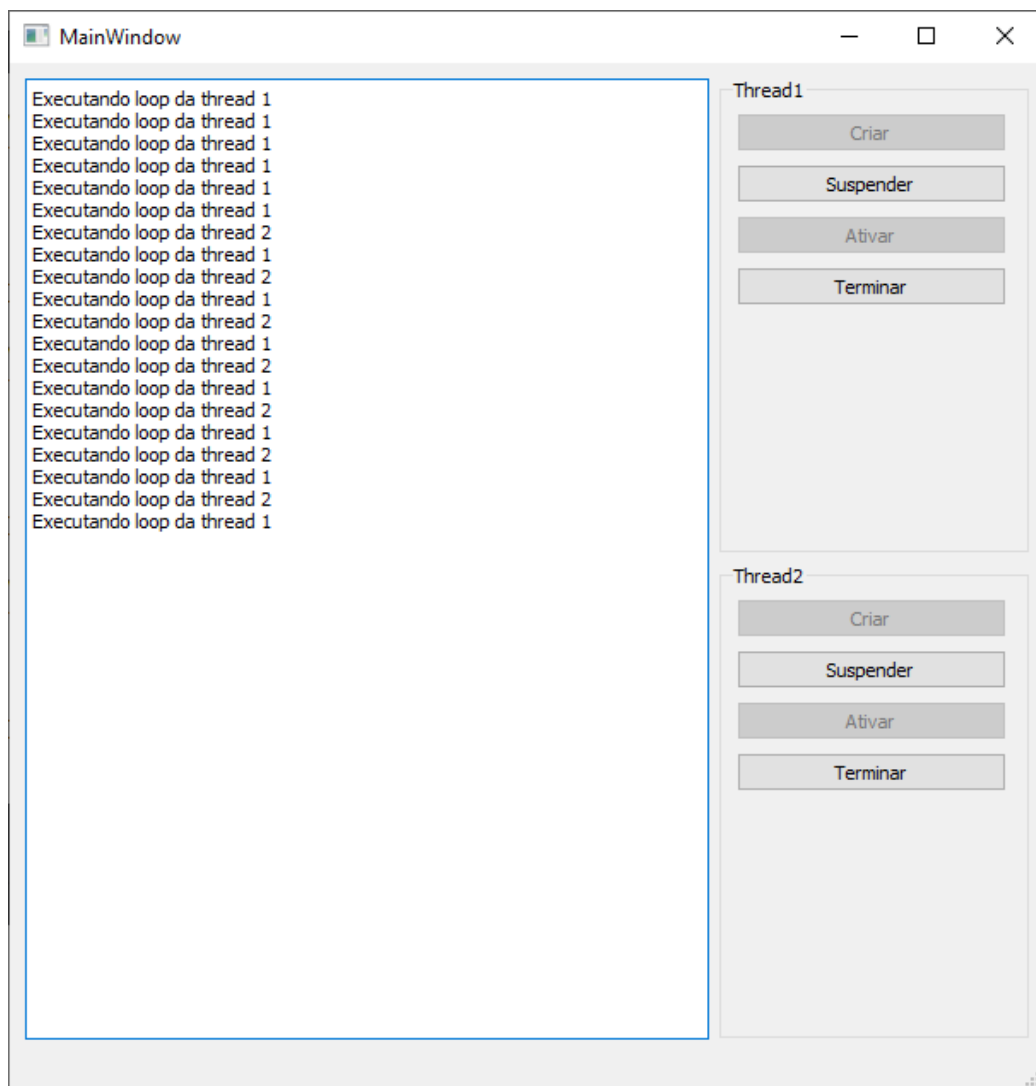


Figura 4: Teste 1 do Programa com Qt

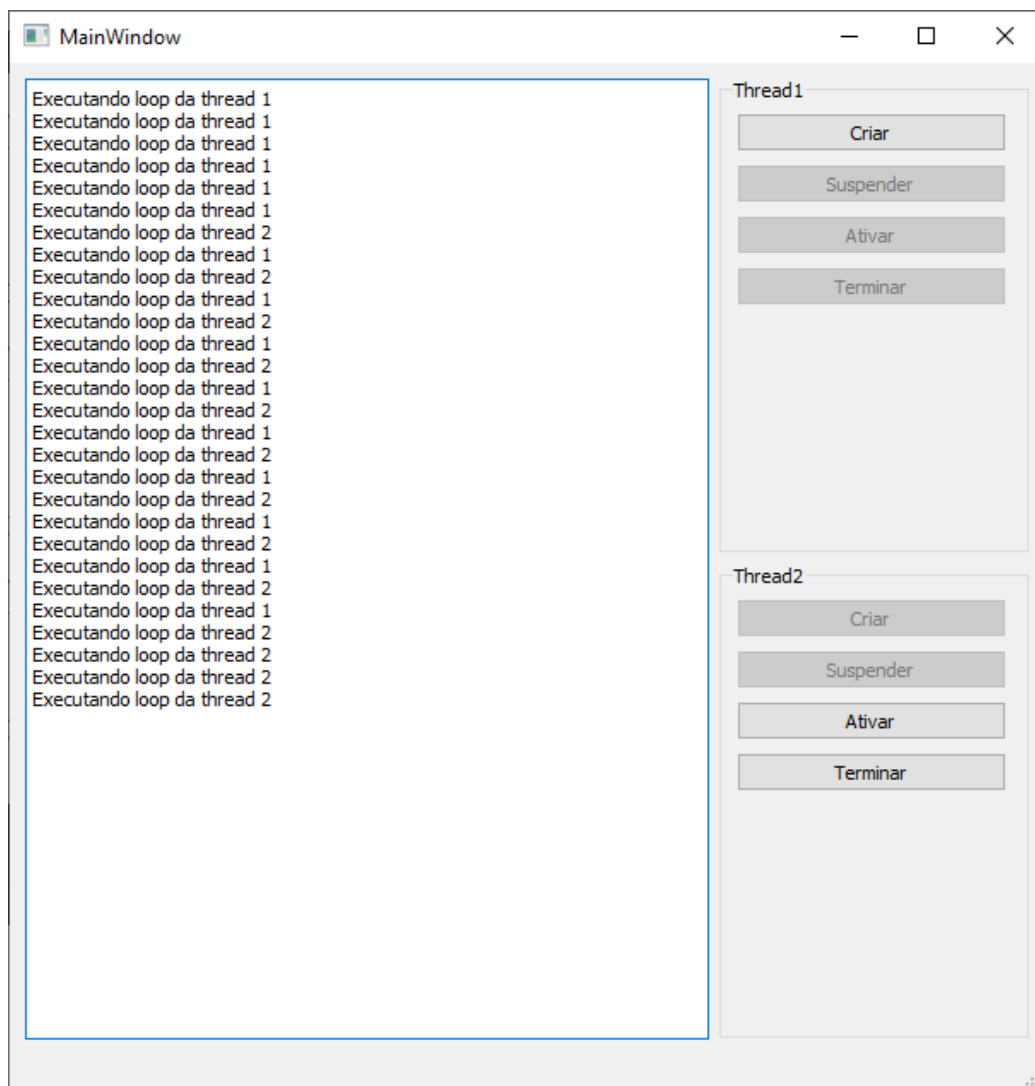


Figura 5: Teste 2 do Programa com Qt