**Extra Lab 3 – Race Condition in PySide6 (With and Without Fixes)**

**Objective:**

This example demonstrates **both cases**:

1. **With a race condition (incorrect behavior)**
2. **Fixed using QMutex (correct behavior)**

**Scenario 1: Race Condition Present (Incorrect Behavior)**

In this scenario, **two threads update a shared counter and a log list** without synchronization, leading to unpredictable results.

import sys

import time

import random

from PySide6.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QTextEdit

from PySide6.QtCore import Qt, QThread, Signal

# Shared variables (causing race conditions)

counter = 0

log\_messages = []

class WorkerThread(QThread):

update\_signal = Signal(int, str) # Signal to update counter and log

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def run(self):

global counter, log\_messages

for \_ in range(5):

temp = counter # Read the counter value

time.sleep(random.uniform(0.1, 0.5)) # Simulate processing delay

counter = temp + 1 # Increment (RACE CONDITION!)

log\_messages.append(f"{self.name} incremented counter to {counter}")

self.update\_signal.emit(counter, log\_messages[-1])

class MainWindow(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Race Condition Demo - Incorrect")

self.setGeometry(200, 200, 400, 300)

self.counter\_label = QLabel("Counter: 0")

self.counter\_label.setAlignment(Qt.AlignCenter)

self.log\_text = QTextEdit()

self.log\_text.setReadOnly(True)

self.start\_button = QPushButton("Start Threads")

self.start\_button.clicked.connect(self.start\_threads)

layout = QVBoxLayout()

layout.addWidget(self.counter\_label)

layout.addWidget(self.log\_text)

layout.addWidget(self.start\_button)

self.setLayout(layout)

def start\_threads(self):

"""Start two threads that modify shared resources"""

self.thread1 = WorkerThread("Thread-1")

self.thread2 = WorkerThread("Thread-2")

self.thread1.update\_signal.connect(self.update\_ui)

self.thread2.update\_signal.connect(self.update\_ui)

self.thread1.start()

self.thread2.start()

def update\_ui(self, count, log\_msg):

"""Update UI with counter and log messages"""

self.counter\_label.setText(f"Counter: {count}")

self.log\_text.append(log\_msg)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Expected Issues (Race Condition)**

* **Counter skips values** (e.g., 1 → 1 → 3 instead of 1 → 2 → 3).
* **Logs may be out of order**.
* **Threads overwrite each other’s updates**.

**Scenario 2: Fixed Using QMutex**

Here, we use **QMutex** to **prevent multiple threads from updating shared resources at the same time**.

import sys

import time

import random

from PySide6.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QVBoxLayout, QTextEdit

from PySide6.QtCore import Qt, QThread, Signal, QMutex

# Shared variables

counter = 0

log\_messages = []

mutex = QMutex() # Mutex to prevent race condition

class WorkerThread(QThread):

update\_signal = Signal(int, str)

def \_\_init\_\_(self, name):

super().\_\_init\_\_()

self.name = name

def run(self):

global counter, log\_messages

for \_ in range(5):

time.sleep(random.uniform(0.1, 0.5)) # Simulate processing delay

mutex.lock() # Lock before modifying shared resources

temp = counter

counter = temp + 1

log\_messages.append(f"{self.name} incremented counter to {counter}")

mutex.unlock() # Unlock after update

self.update\_signal.emit(counter, log\_messages[-1])

class MainWindow(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Race Condition Demo - Fixed")

self.setGeometry(200, 200, 400, 300)

self.counter\_label = QLabel("Counter: 0")

self.counter\_label.setAlignment(Qt.AlignCenter)

self.log\_text = QTextEdit()

self.log\_text.setReadOnly(True)

self.start\_button = QPushButton("Start Threads")

self.start\_button.clicked.connect(self.start\_threads)

layout = QVBoxLayout()

layout.addWidget(self.counter\_label)

layout.addWidget(self.log\_text)

layout.addWidget(self.start\_button)

self.setLayout(layout)

def start\_threads(self):

"""Start two threads that modify shared resources safely"""

self.thread1 = WorkerThread("Thread-1")

self.thread2 = WorkerThread("Thread-2")

self.thread1.update\_signal.connect(self.update\_ui)

self.thread2.update\_signal.connect(self.update\_ui)

self.thread1.start()

self.thread2.start()

def update\_ui(self, count, log\_msg):

"""Update UI with counter and log messages"""

self.counter\_label.setText(f"Counter: {count}")

self.log\_text.append(log\_msg)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Fixes Implemented**

1. **mutex.lock() & mutex.unlock()**
   * Prevents **multiple threads from modifying counter or log\_messages at the same time**.
2. **Ensures sequential updates**
   * Every increment happens correctly (1 → 2 → 3 → 4).
   * No missing or duplicate updates.
3. **Thread-safe logging**
   * Messages are logged in the correct order.

**Comparison: Before vs After Fix**

| **Feature** | **Before Fix (Race Condition)** | **After Fix (Using QMutex)** |
| --- | --- | --- |
| Counter Updates | Skips or duplicates values | Updates sequentially |
| Log Order | Messages may be out of order | Log messages appear correctly |
| UI Stability | Possible flickering/crashes | Smooth updates |
| Data Corruption | Counter may not be accurate | Correct values every time |

**Final Takeaways**

* Race conditions occur when multiple threads modify shared resources without coordination.
* Using QMutex ensures only one thread modifies the resource at a time.
* This fix is crucial in applications handling shared resources like counters, logs, files, or databases.