USP PROJECT REPORT

Submitted By:

Muskaan Goel[ENG18CS0177]
Muskaan Sinha[ENG18CS0178]
Meghana Shree M[ENG18CS0165]
Lysetti Lakshmi Poojitha[ENG18CS0150]

of

BACHELOR OF TECHNOLOGY

in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

at

DAYANANDA SAGAR UNIVERSITY

SCHOOL OF ENGINEERING, BANGALORE-560068

6TH SEMESTER

(Course Code: 16CS)

DAYANANDA SAGAR UNIVERSITY

CERTIFICATE

This is to certify that the CDSS Project report entitled **“Automatic Thread Synchronisation”** being submitted to the Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 6th semester B.Tech C.S.E of this university during the academic year.

2020-2021.

Date:___________

____

Signature of the Faculty in Charge

____

Signature of the Chairman

DECLARATION

We Muskaan Goel[ENG18CS0177],Muskaan Sinha[ENG18CS0178],Meghana Shree M[ENG18CS0165],Lysetti Lakshmi Poojitha[ENG18CS0150]students of 6th semester B.Tech in Computer Science and Engineering, Dayananda Sagar University, Bengaluru, hereby declare that titled

"Automatic Thread Synchronisation"

submitted to the Dayananda Sagar University during the academic year 20202021, is a record of an original work done by me under the guidance of **Dr.Shradha Naik** , Assistant Professor, Department of computer science engineering, Dayananda Sagar University, Bengaluru. This project work is submitted in partial fulfilment for the award of the degree of Bachelor of Technology in Computer Science. The result embodied in this thesis has not been submitted to any other university or institute for the award of any degree.

Team Members,

Muskaan Goel[ENG18CS0177]
Muskaan Sinha[ENG18CS0178]
Meghana Shree M[ENG18CS0165]
Lysetti Lakshmi Poojitha[ENG18CS0150]

Abstract

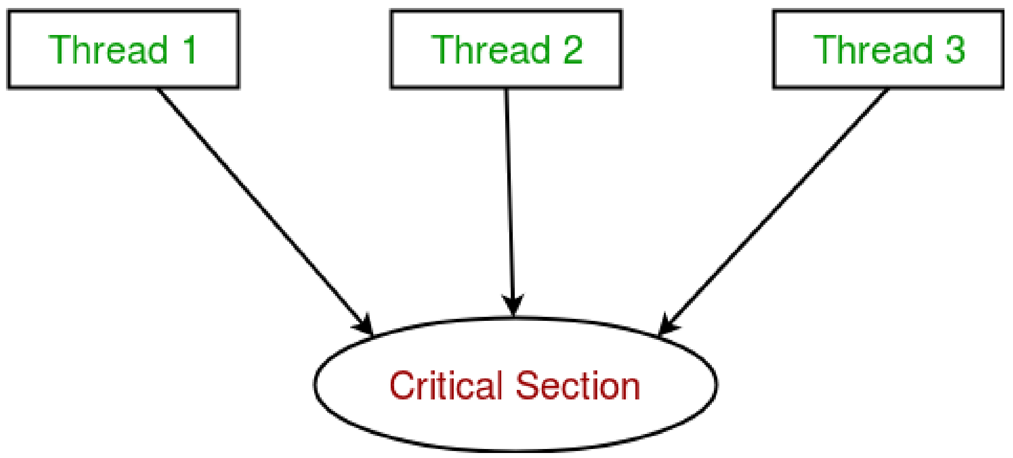
In a multithreaded environment, each thread has its own local thread stack and registers. If multiple threads access the same resource for read and write, the value may not be the correct value. For example, let's say our application contains two threads, one thread for reading content from the file and another thread writing the content to the file. If the write thread tries to write and the read thread tries to read the same data, the data might become corrupted. In this situation, we want to lock the file access. The thread synchronization has two stages. Signaled and non-signaled.

The signaled state allows objects to access and modify data. The non-signaled state does allow accessing or modifying the data in the thread local stack.

Effective compilation of packet processing applications onto the Intel IXP network processors requires, among other things, the automatic use of multiple threads on one or more processing elements, and the automatic introduction of synchronization as required to correctly enforce dependencies between such threads. We describe the program transformation that is used in the Intel Auto-partitioning C Compiler for IXP to automatically multithread/multi-process a program for the IXP. This transformation consists of steps that introduce inter-thread signaling to enforce dependencies, optimize the placement of such signaling, reduce the number of signals in use to the number available in hardware, and transform the initialization code for correct execution in the multithreaded version. Experimental results show that our method provides impressive speedup for six PPSes (Packet Processing Stages) in the widely used NPF IP forwarding benchmarks. For most packet processing stages, our algorithms can achieve almost linear performance improvement after automatic multithreading transformation. The automatic multi-processing transformation help further boost the speedup of two PPSe

I. Introduction

Thread synchronization may be defined as a method with the help of which we can be assured that two or more concurrent threads are not simultaneously accessing the program segment known as critical section.On the other hand, as we know that critical section is the part of the program where the shared resource is accessed.



Thread synchronization is a mechanism which ensures that two or more concurrent threads do not simultaneously execute some particular program segment known as the critical section.

Critical section refers to the parts of the program where the shared resource is accessed.

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. As a result, the values of variables may be unpredictable and vary depending on the timings of context switches of the processes.

II. Problem Statement

In a multithreaded environment, each thread has its own local thread stack and registers. If multiple threads access the same resource for read and write, the value may not be the correct value.

For example, let's say our application contains two threads, one thread for reading content from the file and another thread writing the content to the file. If the write thread tries to write and the read thread tries to read the same data, the data might become corrupted. In this situation, we want to lock the file access. The thread synchronization has two stages. Signaled and non-signaled.

The signaled state allows objects to access and modify data. The non-signaled state does allow accessing or modifying the data in the thread local stack.

III. Objective

The need for synchronization does not arise merely in multi-processor systems but for any kind of concurrent processes; even in single processor systems. Mentioned below are some of the main needs for synchronization:

Forks and Joins: When a job arrives at a fork point, it is split into N sub-jobs which are then serviced by n tasks. After being serviced, each sub-job waits until all other sub-jobs are done processing. Then, they are joined again and leave the system. Thus, parallel programming requires synchronization as all the parallel processes wait for several other processes to occur.

Producer-Consumer: In a producer-consumer relationship, the consumer process is dependent on the producer process till the necessary data has been produced.

Exclusive use resources: When multiple processes are dependent on a resource and they need to access it at the same time, the operating system needs to ensure that only one processor accesses it at a given point in time. This reduces concurrency.

IV. Methodology

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the Lock() method, which returns the new lock.

The acquire(blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.

If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The release() method of the new lock object is used to release the lock when it is no longer required.

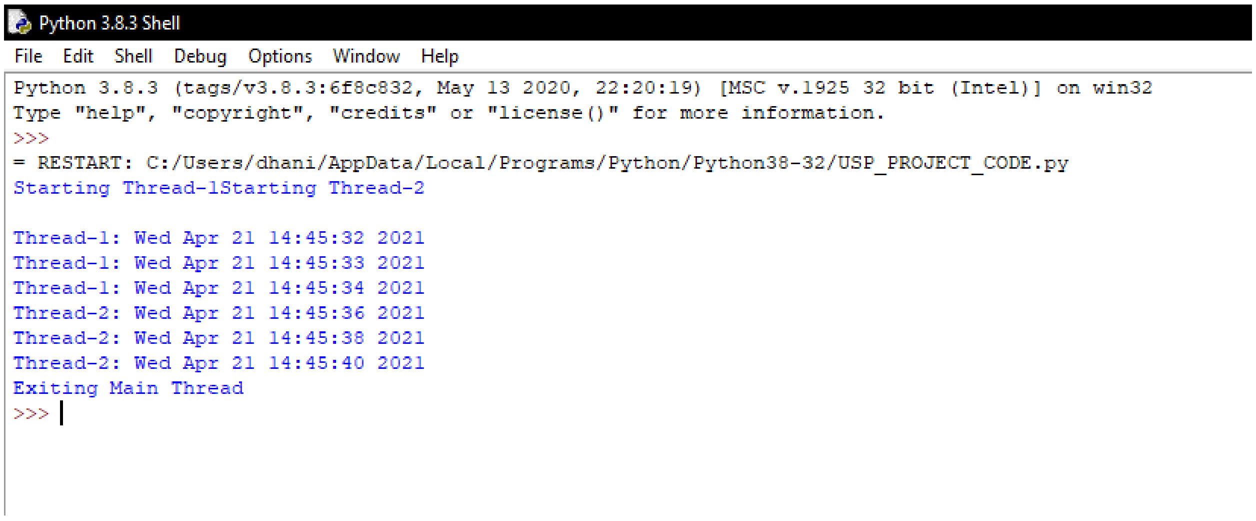
V. Software and hardware Requirement

SOFTWARE-Windows supporting IDLE Python 3.8 (64 bit)

HARDWARE-

- ► Intel Core i5 or i7 processor.
- ► Full HD resolution, ideally 1920×1080.
- ► 8GB of RAM.

VI. Results



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/dhani/AppData/Local/Programs/Python/Python38-32/USP_PROJECT_CODE.py
Starting Thread-1Starting Thread-2

Thread-1: Wed Apr 21 14:45:32 2021
Thread-1: Wed Apr 21 14:45:33 2021
Thread-1: Wed Apr 21 14:45:34 2021
Thread-2: Wed Apr 21 14:45:36 2021
Thread-2: Wed Apr 21 14:45:38 2021
Thread-2: Wed Apr 21 14:45:40 2021
Exiting Main Thread
>>>
```

VII. Conclusion

Thread Synchronization is used to access the shared resources in a multithreaded environment. The programmer decides the situation for when to use the synchronization object efficiently. The MFC Thread Synchronization classes internally call the Win32 API functions. The MFC Thread Synchronization classes wrap many of the functionalities from the Windows environment.

VIII. References

https://en.wikipedia.org/wiki/Synchronization_(computer_science)

https://www.sciencedirect.com/topics/computer-science/synchronize-thread

https://supportline.microfocus.com/documentation/books/nx30books/mtsync.html

https://www.geeksforgeeks.org/multithreading-in-python-set-2-synchronization/