ITP 30002-02 Operating System, Spring 2020

Homework 4

Eunjun Jang 21800633, jangej1031@naver.com, Demo: https://youtu.be/-5GqbfoI67A

1. Introduction

The deadlock problem, where multiple threads cannot release a lock and fall into an infinite wait at the same time, is a common problem in multithreaded programs. There are many deadlock detection algorithms. In this paper, I will cover how to implement resource deadlock detector using cyclic deadlock monitoring algorithm.

To implement this, I used run-time interposition technique to get locking and unlocking data. Moreover, I used a named pipe which is FIFO in Linux OS to check cycle. An adjacency matrix is used to represent a lock graph. It uses a simple stack structure to store the thread's locked mutex data.

In evaluation section, I will show you some deadlock situation and prove my implementation is work fine.

At the discussion, I will discuss about improving its performance and DLL injection.

2. Approach

In this section, I will describe how it was implemented. Implementation can divide into two part. First part is ddmon which is wrapping mutex lock and unlock function in pthread and pass the data through FIFO. Second part is ddchck which is run-time resource deadlock detection part.

2.1. Wrapping Pthread

First, I will talk about ddmon part. The ddmon is wrapping of pthread API. The ddmon wraps pthread_mutex_lock and pthread_mutex_unlock. Overall structure of routine is like below. First, it calls dlsym to find original function. After that, it gets thread ID and mutex's memory address data and sends these to ddchck. Simply calling pthread_self can get current thread ID. After all previous work is done, it calls original pthread function. This execution order is important. This is because data must be transmitted before the locking mutex which already locked so that it can be detected before falling into deadlock.

2.2. Communication Through FIFO

FIFO, named pipe in Linux OS as I already described before, used to communicate ddchck and ddmon. There is some convection to pass and get a message. When mutex is locked, message will be like this:

```
lock_<thread ID>_<mutex address>\n
<filename>_[hex address]\n
```

Underscore represented a one space. Second line is result of backtrace function. Unlocking message is like this:

```
unlock_<thread ID>_<mutex address>\n
```

Unlike locking message, unlocking message is simply one-line message. Because there is no need to pass call stack data. FIFO Reader which is ddchck can get a message from FIFO to process message.

2.3. Deadlock Detection with Lock Graph

Thread ID and mutex address is unique key to distinguish each other but these cannot use to index. Basically, in ddchck, all threads and mutexes are represented in index. It means there are two arrays to contain each thread ID and mutex address. When ddchck receive the message, ddchck find the thread ID from array first. If there is no such value, it inserts empty space of array and that array index becomes internal ID of the thread. This is same as mutex. There are 10 history stacks for each thread to save own mutex lock history. Lock graph G is represented by adjacency matrix. The weight of edge is just one which do not care its weight. G[i][j] is an edge way from node i to j. Now, I will talk about Locking routine. The routine first check thread ID. If there no such thread ID, it pushes thread ID to empty space of an array. After that, find the mutex address. This is same as thread ID case. After declaration index is done, it checks that thread lock history. Lock history buffer is not empty, edge will establish like this:

G[top history mutex index][current mutex index] = 1;

Otherwise, it just appends mutex index to its history buffer. After the edge insertion, it checks cycle on the G. Cycle check routine is simply made with DFS algorithm. If it finds the cycle in the graph G, there is a deadlock. So, it opens addr2line with popen using the hexadecimal address obtained from the FIFO and the file name received as a command line input. And it reads one-line string using fscanf with popen return file pointer.

Unlocking is very simple. Unlocking routine simply clear all zero an mutex index of row and column of adjacency matrix G. After that, it removes from its history buffer.

3. Evaluation

There are three test code for checking implementation is work fine. First test is detecting self-loop which means thread lock same mutex twice. Second test is ab-bc-ca. One thread lock in a-b order, other thread lock in b-c order and last thread lock in c-a order. They have a dependency with each thread to lock second mutex. Last test is abba with sleep. Last test is done with no deadlock. All tests are done in Peace server. Runtime interposition is done with changing LD_PRELOAD.

```
-----------------
ddchck> pipe message: lock 139941823719168 0x6010a0
ddchck> create node!
ddchck> l_idx: 0
-----------------
ddchck> pipe message: lock 139941815326464 0x6010e0
ddchck> create node!
ddchck> l_idx: 1
-----------------
ddchck> pipe message: lock 139941806933760 0x601120
ddchck> create node!
ddchck> l_idx: 2
-----------------
ddchck> pipe message: lock 139941823719168 0x6010a0
ddchck> l_idx: 0
ddchck> edge (0, 0) added
ddchck> Deadlock detected
at:/home/s21800633/OS_PA/HW4/test1.c:21
<0, > cycle
-----------------
ddchck> pipe message: lock 139941815326464 0x6010e0
ddchck> l_idx: 1
ddchck> edge (1, 1) added
ddchck> Deadlock detected
at:/home/s21800633/OS_PA/HW4/test1.c:32
<1, > cycle
-----------------
ddchck> pipe message: lock 139941806933760 0x601120
ddchck> l_idx: 2
ddchck> edge (2, 2) added
ddchck> Deadlock detected
at:/home/s21800633/OS_PA/HW4/test1.c:43
<2, > cycle
```

Figure 1. Test1

First three messages are about creating node. After three messages appears, it locks again. So, there are falling into deadlock. The ddchck successfully detects self-loop deadlock.

```
-----------------
ddchck> pipe message: lock 140219160696576 0x6010e0
ddchck> create node!
ddchck> l_idx: 0
-----------------
ddchck> pipe message: lock 140219152303872 0x601120
ddchck> create node!
ddchck> l_idx: 1
-----------------
ddchck> pipe message: lock 140219169089280 0x6010a0
ddchck> create node!
ddchck> l_idx: 2
-----------------
ddchck> pipe message: lock 140219152303872 0x6010a0
ddchck> l_idx: 2
ddchck> edge (1, 2) added
-----------------
ddchck> pipe message: lock 140219160696576 0x601120
ddchck> l_idx: 1
ddchck> edge (0, 1) added
-----------------
ddchck> pipe message: lock 140219169089280 0x6010e0
ddchck> l_idx: 0
ddchck> edge (2, 0) added
ddchck> Deadlock detected
at:/home/s21800633/OS_PA/HW4/test2.c:21
<0, 1, 2, > cycle
```

Figure 2. Test2

After creating nodes, edge (1, 2), edge (0, 1) and edge (2, 0) is done. This is deadlock because they have cycle. The ddchck successfully detects ab-bc-ca deadlock.

```
-----------------
ddchck> pipe message: lock 139835893131008 0x6010a0
ddchck> create node!
ddchck> l_idx: 0
-----------------
ddchck> pipe message: lock 139835893131008 0x6010e0
ddchck> create node!
ddchck> l_idx: 1
ddchck> edge (0, 1) added
-----------------
ddchck> pipe message: unlock 139835893131008 0x6010e0
ddchck> delete node!
-----------------
ddchck> pipe message: unlock 139835893131008 0x6010a0
ddchck> delete node!
-----------------
ddchck> pipe message: lock 139835905656576 0x6010e0
ddchck> create node!
ddchck> l_idx: 0
-----------------
ddchck> pipe message: lock 139835905656576 0x6010a0
ddchck> create node!
ddchck> edge (0, 1) added
-----------------
ddchck> pipe message: unlock 139835905656576 0x6010a0
ddchck> delete node!
-----------------
ddchck> pipe message: unlock 139835905656576 0x6010e0
ddchck> delete node!
```

Figure 3. Test3

First four messages is done with same thread. This is no deadlock. After four messages, other thread lock and unlock like abba. This also work fine with no deadlock. The ddchck's unlock is successfully work.

4. Discussion

In this section, I will talk about improving performance of ddchck and DLL Injection.

4.1. Performance Enhancement

Since the algorithm now uses the adjacency matrix, it goes back to the O(V^2) algorithm when searching and deleting. To increase the performance, first change the graph representation that represents the lock to the Adj List. Then it can be Ω(V+E). In addition, it takes O(V) time using linear search as an algorithm to retrieve the index of the thread and the index of the mutex, and this can be improved to O(1) by changing to a hash map.

4.2. DLL Injection

It seems to work like DLL Injection to let the user load the shared library created by the user and execute the user made code. I heard that DLL injection is used when hacking PC game clients. Using DLL Injection, it is possible to tamper with the client and manipulate the game money or score at will by intercepting it when sending the desired packet to the server or writing a value of a specific memory. I also found out that it is Code Injection. It seems to be the principle of embedding code directly into the process memory. Allocating thread memory to execute arbitrary code to the process, putting code into the memory, and executing the thread. I was able to understand a bit by looking at the examples, but most of them were more difficult than I thought as I was tearing down assembly and memory on Windows operating system. I will study x86 assembly later. It is difficult to understand because it is different from the MIPS learned in computer architecture.

5. Conclusion

In this paper, I talked about ddmon and ddchck which is run-time resource deadlock detector. This work is very meaningful to me. This work is very excited. I want to do more work practical homework.