ITP 30002-02 Operating System, Spring 2020

Homework 4

Park Ye Gyeom/21600277/21600277@handong.edu

1. Introduction

The function of the program required in this task is to identify the deadlocks of the target program executed in real time and output the thread_id and memory address of mutex that reason of the deadlocks to the user. To implement this program, ddmon.so (ddmon.c), the dynamic library of the target program, should be implemented and ddchck.c should be implemented to determine whether the target program has deadlocks by creating a graph with the information received through FIFO. To implement Ddmon.so, it is necessary to implement a function that intercepts the pthread_mutex_lock() and pthread_mutex_unlock() provided by pthread.h through the Library Interpositioning and obtains information to draw a graph and passes it to ddchck.c through FIFO. n ddchck.c, draw a graph based on the information received through FIFO and check the presence of the cycle when the edge is added. If a cycle occurs, print the thread_id and mutex_memory_address that caused the cycle and ask the user if it is shut down. To implement this function, thread_node and mutex_node were created and graphed, and the occurrence of cycle was detected through DFS when edge was added.

2. Approach

-Description of FIFO

Before I begin this explanation, I will briefly explain FIFO. FIFO is a named pipe that is one of the IPC mechanisms for exchanging information between two processes, not the parent child relationship process. To accomplish this task, FIFO, called ddtrace, was assumed to be in the same directory as target program and ddchck Ddtrace was used similarly to file descriptor. In Ddmon.c, Dtrace was opened as a way to write, and in ddchck.c, the IPC function is provided while opening as a way to perform read.

- Approach of ddmon.c

Ddmon.c is compiled to ddmon.so via the gcc -shared -fPIC -o ddmon.so ddmon.c -ldl compilation command and executed with a dynamic library target program. And Ddmon.c modifies the existing function through the function

```
pthread_mutex_lock_cp = dlsym(RTLD_NEXT, "pthread_mutex_lock") ;
```

```
pthread_mutex_unlock_cp = dlsym(RTLD_NEXT, "pthread_mutex_unlock") ;
```

Dlsym of pthread_mutex_lock and pthread_mutex_unlock. Thus, when the function pthread_mutex_lock() and pthread_mutex_unlock() are executed in the target program, the function's basic function is maintained and executed as a function of ddmon.c, which implements additional functions. In this executed function, the thread_id and the mutex_memory_address are sent through the write function to the ddchck, and the ddchck checks the deadlock in real time by drawing a graph with the received information.

-Types of information sent by FIFO

There are three information sent from ddmon.c to ddchck.c as follows.

1. The thread id requested for Pthread mutex lock and the information for that mutex.
2. After requesting Pthread mutex lock, information about the thread id that was given the mutex authority and made the lock state, and the corresponding mutex.
3. Information about the thread id and its mutex that performed the release via Pthread mutex unlock.

As shown above, three kinds of information are transmitted through ddtrace. Therefore, a process is needed to implement it sensibly in ddchck through a variable called sign. The format of the information sent is sent to a structure called Package. This package contains thread id, mutex memory address, and sign.

- Approach of ddchck.c

Ddchck reads the information transferred to ddtrace in units of the package structure through the read function. The ddchck categorizes the read-in information through the value of the sign.

- How to share information received through FIFO

1. The sign value is 0
   In this case, the thread requested a mutex lock. At this point, because the mutex is not yet assigned to this thread, create a thread->mutex edge.
2. The sign value is 1
   In this case, the thread is engaged in a critical section after confirming the mutex lock. At this point, because this thread owns the mutex, delete the thread->mutex edge and create the mutex-

>thread edge.

3. The sign value is 2
   In this case, the thread executes the mutex unlock and is outside the critical section. Delete the mutex->thread edge since the lock has been released.

- Structure of the Graph

A Ddchck consists of two types of nodes: a thread node and a mutex node. Thread node can only point to a mutex node, whereas mutex node can only point to a thread node. Therefore, since this graph is a directed graph, it is possible to check the presence of cycle through the concept of DFS concept. Therefore, when the sign value is 0, add thread->mutex edge and check the presence of the cycle through the conform_cycle function.

-Target program

The target program includes abba.c and dinning_deadlock.c, and penil_eraser.c. Evaluate the normal operation of the detector with this target program.

- Pencil_eraser.c is a target program in which ABBA lock occurs. It has been confirmed through several experiments that deadlock occurs once in five times. ABBA lock occurs between mutex(temp) and mutex(pencil) at stop() and get() functions.

3. Evaluation

We implemented ddmon.c and ddchck.c in the above manner and executed abba.c(prof.example) dinning_deadlock.c(prof.example) and pencil_eraser(my.example) as target program. The thread id requested for Pthread mutex lock and the information for that mutex.

1. In case of Deadlock
   The thread id and mutex id that caused deadlock were successfully printed in ddchck. If deadlock occurs in ddchck, the ddchck is not shut down. Other target programs are also implemented to monitor deadlock generation.

```
ALERT!! DEADLOCK DETECT!!
THREAD ID --------> [140147935680256]
MUTEX ADDRESS ----> [0x6020c8]

DO YOU WANT TO RESTART? PLEASE INPUT (Y / N) : y


ALERT!! DEADLOCK DETECT!!
THREAD ID --------> [139645519800064]
MUTEX ADDRESS ----> [0x602128]

DO YOU WANT TO RESTART? PLEASE INPUT (Y / N) : n

        ====================
        ==       BYE!!      ==
        ====================
```

2. Deadlock does not occur
   The ddchck was implemented to wait with nothing printed.

4. Discussion

I'm a little disappointed that I didn't realize the full Lock Graph in this project and I think I've implemented the slightly modified Lock Graph. I want to improve the Lock Graph side by watching other classmates' implemented programs. When I check the cycle, I checked the cycle through DFS, but I am sorry that I have implemented the edge by linear search without taking advantage of the pointer. I want to think more about it after submitting the assignment and make use of the advantage of the pointer. I have implemented Deadlock detector in this project, and I would like to implement deadlock predictor & restore program that can prevent deadlock before falling into deadlock state.

5. Conclusion

Implementing Deadlock detector required learning about the pthread_mutex_lock and pthread_mutex_unlock functions. And the process and study of adding and modifying functions using Library Interpositioning were needed. Through the process of preserving the original function of the function and sending the necessary information to ddchck through FIFO, we learned about the operating principles and concepts of FIFO.

Finally, the lock graph was implemented through the information read by FIFO. The graph examined the cycle using the DFS algorithm. In case of cycle, a program that informs the user that the cycle is dead lock could be implemented. What is unfortunate is that they have not tried many things about backtrace and have only implemented the detector. Still, being able to implement a program that accurately detects deadlock will remain a very good experience.

Demo URL : https://youtu.be/87jjP53xkdg