ITP 30002-01 Operating System, Spring 2020
**Homework 4**
**Name** : Min Jae Yi (이민재) **Student number** : 21800511 **Email Address** : 21800511@handong.edu

## 1. Introduction

Demo video is on https://www.youtube.com/watch?v=tbUTNilpdC4&feature=youtu.be

The main part of the hw4 can divided into two parts making “ddchck.c” and “ddmon.c”

The requirement or function for the c program can be shown as below.

| |
|---|
| *ddchck.c* |
| 1-1. When mutex lock or mutex unlock happens update the graph of mutex. |
| 1-2. Whenever mutex lock is added check whether it is deadlock |
| 1-3. Back trace where deadlock happened |
| *ddmon.c* |
| 2-1. Deliver the context of the mutex to the ddchck. |

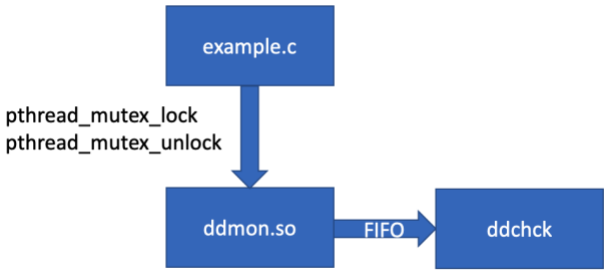
***Table 1 : problem define***

Brief approach for each problem.

| |
|---|
| a. 1-1 can be handled by using FIFO. |
| b. 1-2 can be handled by creating graph |
| c. 1-3 can be handled by popen with addr2line command |
| d. 2-1 can be handled by dynamic linking and FIFO |

***Table 2 : problem approach***

## 2. Approach




***Pic 2 : Overview of the program flow***

Pic 1 shows the overview of the program. By connecting example executable file with ddmon.so dynamic linking happens. ddmon.so refines the pthread_mutex_lock and pthread_mutex_unlock. And every time these functions are used ddchck gets the information of the current mutex context through the FIFO.

Here are more details of the implementations.

1. Dynamic linking

ddmon.c redefines the pthread_mutex_lock and pthread_mutex_unlock. When these functions are executed, ddmon.so informs to the ddchck that the functions has runned. It delivers 3 data t_id, mutex_id and context in to string. The example formula are “thread : %d mutex %d locked”. t_id are known through syscall. mutex and context are known through the function itself. It sends the string through FIFO through the .ddtrace channel.

2. Implementing FIFO

ddmon.so run as sender and ddchck run as receiver. They shared the .ddtrace channel and when new data are sent through the ddmon, ddchck finds the data and decode the string to get information.

3. Making graph by ddchck.

There are three structure of memory that defines the graph.

One is Thread. When new thread is created it creates the structure called Thread. Thread contains thread_id, hold_mutex[] and thread_id represents the thread id and hold_mutex array holds the mutex that is used in the thread.

Another is mutex[10]. It saves the given mutex_id and changes into index 0~9.

Last is edge[10][10]. It is 2d array that holds the relation of edge of the node. When edge[i][j] is 1, it refers that mutex i arrows mutex j.

These that represents the current state of mutex.

### 3. Evaluation

EX 1. ABBA problem

```
s21800511@peace:~/HW4$ LD_PRELOAD="./ddmon.s
thread : 35993 mutex 6295776 locked
thread : 35994 mutex 6295712 locked
thread : 35993 mutex 6295712 locked
thread : 35994 mutex 6295776 locked
```

***Pic 2 ABBA context***

```
ADD(1 0)
0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

***Pic 3 ABBA final mutex table.***

EX 2. Dining philosophers Deadlock

```
s21800511@peace:~/HW4$ LD_PRELOAD="./
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 4 is thinking
thread : 35150 mutex 6295816 locked
Philosopher 3 is thinking
Philosopher 0 is thinking
thread : 35152 mutex 6295856 locked
thread : 35153 mutex 6295896 locked
thread : 35150 mutex 6295856 locked
thread : 35154 mutex 6295936 locked
thread : 35152 mutex 6295896 locked
thread : 35149 mutex 6295776 locked
thread : 35153 mutex 6295936 locked
thread : 35149 mutex 6295816 locked
thread : 35154 mutex 6295776 locked
```

***Pic 4 Dinning philosopher context***

```
thread : 35154 mutex
35154 6295776 locked
ADD(3 4)
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

***Pic 5 Dinning philosopher final mutex table***

```
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

***Pic 6 Dinning philosopher final mutex table2***

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

***Pic 7 Dinning philosopher final mutex table3***

Evaluation shows two kinds of example. Pic 2, Pic 3 shows the result of ABBA and Pic 4, Pic 5, Pic6, Pic 7 shows the result of Dining philosophers. Pic2 context are represented as Pic 3 and Pic 4 context are represented as Pic 5. And Pic 6, Pic 7 are representation of other contexts. If you see the Pic 3, you can find that (0,1) and (1,0) are '1'. You can find out that it is circularly contacted which means Deadlock. And if you look at Pic 5 you can find that (0,1), (1,2), (2,3), (3,4) and (4,0) are '1'. Also, this shows the deadlock. Pic 6 are also one example of deadlock that can appear. You can find (0,1) , (1,2), (2,4), (4,3), (3,0) are '1' and contacted in circular. Pic 7 is the case which deadlock didn't happen. You can see it is are changed as '0'. This is the case when deadlock doesn't happens and free all the edges.

**4. Discussion**

One thing I struggled hard was getting thread id to the ddchck. I tried to use static variable to save the thread id when new thread is added. However, I found it is impossible since static are shared through all thread. Then I found simple way by using syscall.

"pid_t x = syscall(__NR_gettid);"

gives the information of the current thread id.

Another thing I struggled was implementing the graph and edges. I found out that graph can implemented in 2d array. I tried to manage the edges through the 2d array and make graph with 2d array to find the circular connecting nodes. However, I could not implement finding deadlock through the graph.

**5. Conclusion**

This homework teaches the way of using FIFO and implementing dynamic linking. By using the dynamic linking programmers can debug the program during the execution of the program. Also by using FIFO programs can communicate during execution through file (channel).