CS302 Lab8 Report

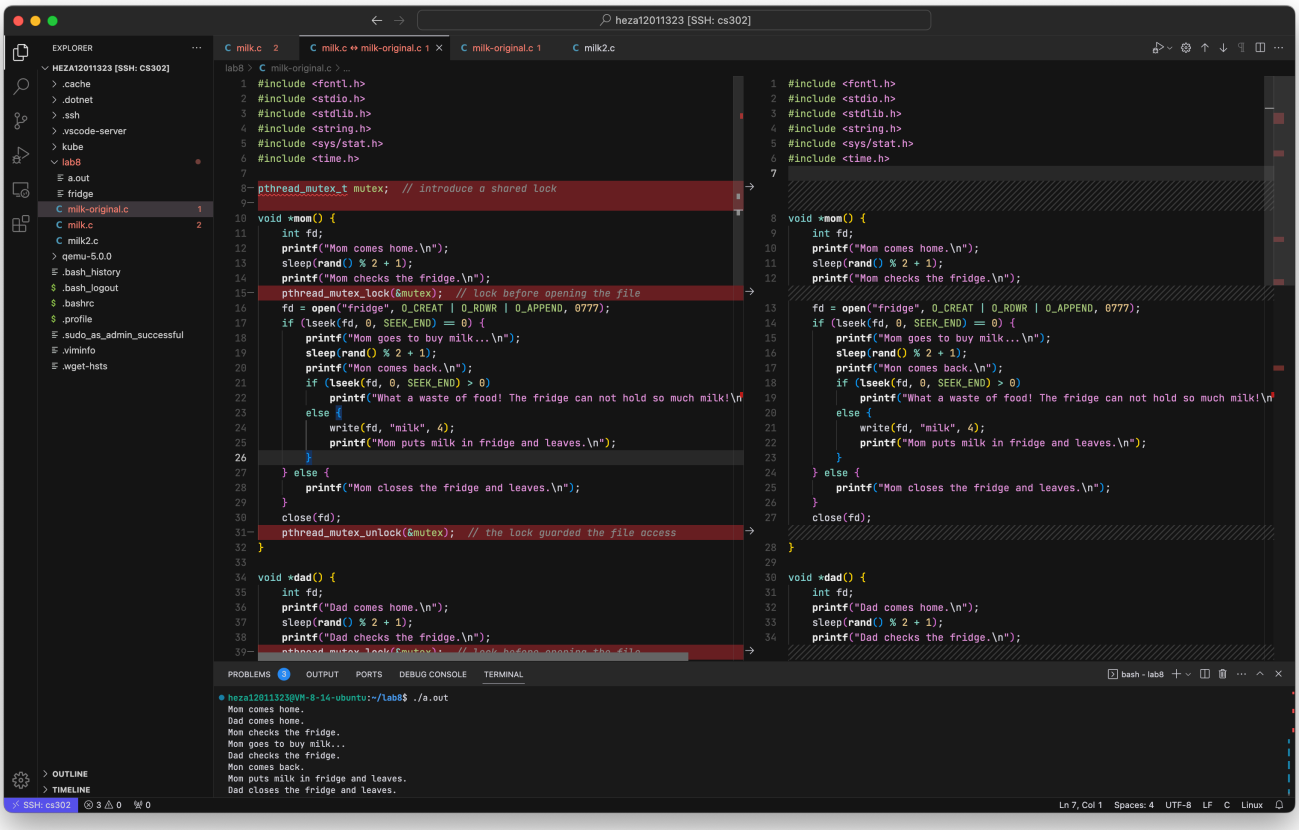
> 何泽安 12011323

> 2023.4.23

1. Mutex

Introducing a global shared mutex lock, and using it to protect every file-access operation, can resolve the potential risks.

Showing the 5 changes:



```
heza12011323@VM-8-14-ubuntu:~/lab8$ gcc milk-original.c -pthread -w && ./a.out    # before
Mom comes home.
Dad comes home.
Mom checks the fridge.
Mom goes to buy milk...
Dad checks the fridge.
Dad goes to buy milk...
Mon comes back.
Mom puts milk in fridge and leaves.
Dad comes back.
What a waste of food! The fridge can not hold so much milk!
heza12011323@VM-8-14-ubuntu:~/lab8$ gcc milk.c -pthread -w && ./a.out    # after
Dad comes home.
Mom comes home.
Dad checks the fridge.
Dad goes to buy milk...
Mom checks the fridge.
Dad comes back.
Dad puts milk in fridge and leaves.
Mom closes the fridge and leaves.
```

2. Condition variable

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <time.h>

#define MAX_MILK 5

pthread_mutex_t mutex;
pthread_cond_t cond;
int milk = 0;

int check_fridge() {
    return milk;
}

int take_milk() {
    if (milk < 1) return;
    milk--;
}

int buy_milk() {
    milk = MAX_MILK;
}

void supply(const char *name) {
    while (1) {
        sleep(rand() % 2 + 1);
        pthread_mutex_lock(&mutex);
```

```
        while (check_fridge() > 0) {
            pthread_cond_wait(&cond, &mutex);
        }
        printf("%s filled the fridge with 5\n", name);
        buy_milk();
        pthread_mutex_unlock(&mutex);
    }
}

void consume(const char *name) {
    while (1) {
        sleep(rand() % 2 + 1);
        pthread_mutex_lock(&mutex);
        if (check_fridge() == 0) {
            pthread_cond_signal(&cond);
        } else {
            printf("%s took 1 milk\n", name);
            take_milk();
        }
        pthread_mutex_unlock(&mutex);
    }
}

void *mom() {
    supply("mom");
}

void *sis() {
    supply("sis");
}

void *dad() {
    consume("dad");
}

void *you() {
    consume("you");
}

int main(int argc, char *argv[]) {
    srand(time(0));
    pthread_t producethread1, producethread2, consumethread1, consumethread2;
    pthread_create(&consumethread1, NULL, mom, NULL);
    pthread_create(&consumethread2, NULL, sis, NULL);
    pthread_create(&producethread1, NULL, dad, NULL);
    pthread_create(&producethread2, NULL, you, NULL);
    pthread_join(producethread1, NULL);
    pthread_join(consumethread1, NULL);
    pthread_join(producethread2, NULL);
```

```
    pthread_join(consumethread2, NULL);
    return 0;
}
```