Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовая работа по курсу

«Операционные системы»

Предотвращение взаимоблокировок

Студент: Петрухин Дмитрий Олегович
Группа: М8О-201Б-18
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020

Содержание

Постановка задачи

Написать библиотеку, которая выдает мьютексы по запросу, но с избежанием взаимоблокировки. Предотвращение взаимоблокировки реализовано с методом атаки условия циклического ожидания. Написать демонстрационную программу для демонстрации работы метода.

Общий метод и алгоритм решения

Метод атаки условия циклического ожидания предотвращает взаимоблокировку за счет устранения одного из условий ресурсных взаимоблокировок, а именно условия циклического ожидания. Циклическое ожидание подразумевает наличие кольцевой последовательности двух и более процессов, каждый из которых, возможно, ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.

Суть метода заключается в том, что все ресурсы определенным образом занумерованы и действует одно правило: процессы могут запрашивать ресурсы, когда только пожелают, но все запросы должным быть сделаны в порядке нумерации ресурсов. То есть если процесс запросит ресурс 3, а затем ресурс 1, то будет выдаваться исключение и ресурс 1 не будет выдан процессу. Если же ресурс освободит ресурс 3, то он может без проблем запросить ресурс 1 и получить его. Таким образом, если процесс запросит ресурс с меньшим номером, чем уже имеющийся ресурс, то процесс либо закончит свою работу, либо запросит ресурс с большим номером, чем имеющийся у него ресурс.

В зависимости от задач, для которых будет применим этот метод, очень важна нумерация ресурсов, чтобы программа работала как можно эффективнее.

Мьютекс— примитив синхронизации, обеспечивающий взаимное исключение исполнения критических участков кода. Классический мьютекс отличается от

двоичного семафора наличием эксклюзивного владельца, который и должен его освобождать (т.е. переводить в незаблокированное состояние).

Для реализации библиотеки использовалась стандартная библиотека pthread.h, которая содержит в себе средства синхронизации потоков, а именно мьютексы. В библиотеке реализована обертка над стандартными системными вызывами инициализации, создании и разрушении мьютекса. При запросе блокировки или освобождения передается вектор захваченных ресурсов данным процессом. Мьютекс выдается процессу если номер последнего захваченного ресурса меньше, чем номер запрашиваемого ресурса (При реализации рабочей программы для конкретной задачи номер ресурса должен сопостовляться с мьютексом с соответсвтующим номером), иначе выдается исключение. В случае успешной выдачи мьютекса в вектор добавляется номер полученного ресурса. При освобождении мьютекса из вектора захваченных ресурсов процессом удаляется номер освобожденного ресурса.

Библиотека содержит следующие функции :

1. **mutex_lock()** — захват мьютекса. Обертка над системным вызовом pthread_mutex_lock().
2. **mutex_unlock()** — Освобождение мьютекса. Обертка над системным вызовом pthread_mutex_unlock().
3. **Mutex()** — конструктор мьютекса, инициализируется с помощью системного вызова pthread_mutex_init(&mutex, NULL).
4. **~Mutex()** — деструктор мьютекса, уничтожается с помощью системного вызова pthread_mutex_destroy(&mutex).

Задача для демонстрации работы программы.

Имеется четыре банка. Клиент хочет переслать деньги из одного банка в другой различными сложными путями. К примеру из банка 1 в банк 4, но при этом деньги в банк 4 должны быть пересланы из банка 3. Имеются ограничения по переводу, а именно банк i не переводит деньги банку j если i > j. В случае, если перевод не может быть возможен, банк возвращает деньги клиенту. Необходимо определить сколько по окончанию дня осталось денег на каждом банковском счету и определить сумму, которую вернули клиентам обратно.

Файлы программы

```
#include "Mutex.h"
#include <iostream>
#include <optional>
#include <pthread.h>
#include <vector>
#include <string>

struct thread_params {
	std::vector<int> transfer_scheme;
	int *res;
	int *account_1;
	int *account_2;
	int *account_3;
	int *account_4;
	int money;
	Mutex m1;
	Mutex m2;
	Mutex m3;
	Mutex m4;
};

void *transactions(void *arg){
	thread_params targ = *(thread_params *) arg;
	std::vector<int> &transfer_scheme = targ.transfer_scheme;
	int *bank1 = targ.account_1;
	int *bank2 = targ.account_2;
	int *bank3 = targ.account_3;
	int *bank4 = targ.account_4;
	int *res = targ.res;
	int money = targ.money;
	Mutex m1 = targ.m1;
	Mutex m2 = targ.m2;
	Mutex m3 = targ.m3;
	Mutex m4 = targ.m4;
	std::vector<int> busy_resources;
	std::string b;
	int f, past;
```

```
	int t = 1;
	int now = 0;
	int size = 0;

	while((transfer_scheme.size() != 0) && (t == 1) ) {
		f = transfer_scheme.front();
		transfer_scheme.erase(transfer_scheme.begin());
		switch (f){
			case 1:
				b = m1.mutex_lock(busy_resources);
				if (b == "Ok" && size == 0){
					*bank1 = *bank1 + money;
					break;
				} else if ((b == "Error") || (b == "Possible deadlock") ){
					t = 0;
					break;
				}
			case 2:
				b = m2.mutex_lock(busy_resources);
				if (b == "Ok" && size == 0) {
					*bank2 = *bank2 + money;
					break;
				} else if (b == "Ok" && size != 0) {
					*bank1 = *bank1 - money;//условно это операция
перевода денег с одного банка к другому
					m1.mutex_unlock(busy_resources);
					*bank2 = *bank2 + money;
					break;
				} else if ((b == "Error") || (b == "Possible deadlock") ){
					t = 0;
					break;
				}
			case 3:
				b = m3.mutex_lock(busy_resources);
				if (b == "Ok" && size == 0){
					*bank3 = *bank3 + money;
					break;
				} else if (b == "Ok" && size != 0) {
					switch(now){
						case 1:
							*bank1 = *bank1 - money;
							m1.mutex_unlock(busy_resources);
							*bank3 = *bank3 + money;
							break;
						case 2:
							*bank2 = *bank2 - money;
							m2.mutex_unlock(busy_resources);
							*bank3 = *bank3 + money;
							break;
					}
					break;
				} else if ((b == "Error") || (b == "Possible deadlock") ){
					t = 0;
					break;
				}
			case 4:
				b = m4.mutex_lock(busy_resources);
				if (b == "Ok" && size == 0){
					*bank4 = *bank4 + money;
					break;
```

```
                        } else if(b == "Ok" && size != 0){
                              switch(now){
                                    case 1:
                                          *bank1 = *bank1 - money;
                                          m1.mutex_unlock(busy_resources);
                                          *bank4 = *bank4 + money;
                                          break;
                                    case 2:
                                          *bank2 = *bank2 - money;
                                          m2.mutex_unlock(busy_resources);
                                          *bank4 = *bank4 + money;
                                          break;
                                    case 3:
                                          *bank3 = *bank3 - money;
                                          m3.mutex_unlock(busy_resources);
                                          *bank4 = *bank4 + money;
                                          break;
                              }
                        } else if ((b == "Error") || (b == "Possible deadlock") ){
                              t = 0;
                              break;
                        }
                  }//switch
                  past = now;
                  now = f;
                  size++;

            }


            if(t != 1) {
                  switch(past){
                        case 1:
                              *bank1 = *bank1 - money;
                              m1.mutex_unlock(busy_resources);
                              break;
                        case 2:
                              *bank2 = *bank2 - money;
                              m2.mutex_unlock(busy_resources);
                              break;
                        case 3:
                              *bank3 = *bank3 - money;
                              m3.mutex_unlock(busy_resources);
                              break;
                        case 4:
                              *bank4 = *bank4 - money;
                              m4.mutex_unlock(busy_resources);
                              break;
                  }
                  *res = *res + money;
            }

            return NULL;
      }


      int main(){
            Mutex m1(1), m2(2), m3(3), m4(4);
            int number_clients, i;
            int bank4 = 0, bank3 = 0, bank2 = 0, bank1 = 0;
```

```
	int loose_money = 0;

	std::cout << "Enter the number of clients\n";
	std::cin >> number_clients;

	std::vector<std::optional<int>> results(number_clients);
	std::vector<pthread_t> threads(number_clients);
	std::vector<thread_params> params(number_clients);


	for(i = 0; i < number_clients; i++){
		int num;
		std::cout << "Enter the transfer amount\n";
		std::cin >> params[i].money;

		std::cout << "Select transfer scheme in chronological order:\n";
		std::cout << "1 - Bank 1\n";
		std::cout << "2 - Bank 2\n";
		std::cout << "3 - Bank 3\n";
		std::cout << "4 - Bank 4\n";
		std::cout << "Enter scheme trans\n";


		int value;
		int j = 0;
		while (1) {
		std::cin >> value ;
		if (value == 0) break ;
		params[i].transfer_scheme.push_back(value);
		j++;
		}

		params[i].account_1 = &bank1;
		params[i].account_2 = &bank2;
		params[i].account_3 = &bank3;
		params[i].account_4 = &bank4;
		params[i].res = &loose_money;
		params[i].m1 = m1;
		params[i].m2 = m2;
		params[i].m3 = m3;
		params[i].m4 = m4;
	}
	for(i = 0; i < number_clients; ++i) {
		if(pthread_create(&threads[i], NULL,
			&transactions, &params[i]) != 0){
			std::cout << "Error create\n";
			return 1;
		}
	}

	for(i = 0; i < number_clients; ++i) {
		if(pthread_join(threads[i], NULL) != 0) {
			std::cout << "Error join\n";
			return 1;
		}
	}

	std::cout << "Amount on the Banks accounts\n";
	std::cout << "Bank 1: " << bank1 << "\n";
	std::cout << "Bank 2: " << bank2 << "\n";
```

```
	std::cout << "Bank 3: " << bank3 << "\n";
	std::cout << "Bank 4: " << bank4 << "\n";
	std::cout << "Amount that may have been in Bank accounts:" << loose_money <<
"\n";

	return 0;
}
```

Mutex.h

```
#pragma once
#include <pthread.h>
#include <vector>
#include <stdexcept>
#include <string>

class Mutex {
public:
	Mutex() = default;
	Mutex(int n){
		number = n;
		init_res = pthread_mutex_init(&mutex, NULL);
		if(init_res != 0){
			throw std::runtime_error("Error erno");
		}
	}
	~Mutex(){
		pthread_mutex_destroy(&mutex);
	}
	std::string mutex_lock(std::vector<int> &v);
	std::string mutex_unlock(std::vector<int> &v);

private:
	int number;
	pthread_mutex_t mutex;
	int init_res;
};
```

Mutex.cpp

```
#include "Mutex.h"

using namespace std;

string Mutex::mutex_lock(vector<int>& v){
	if (v.size() == 0 || v.back() < (this->number) ){
		v.push_back(this->number);
		int b = pthread_mutex_lock(&(this->mutex));
		if(b == 0) {
			return "Ok";
		} else {
			return "Error: " + to_string(b);
		}
	} else if (v.back() > (this->number)) {
		return "Possible deadlock";
	}
}
string Mutex::mutex_unlock(vector<int>& v){
	vector<int>::iterator it = v.begin();

	while(it != v.end()){
		if(*it == (this->number)){
			v.erase(it);
```

```
                int b = pthread_mutex_unlock(&(this->mutex));
                if(b == 0) {
                        return "Ok";
                } else {
                        return "Error: " + to_string(b);
                }
            }
            it++;
        }
        if(it == v.end()){
            return "The resource did not block this mutex";
        }
}
```

Демонстрация работы программы

dobb2@dobb2-Laptop:~/Рабочий стол/OS_KP$ g++ main.cpp Mutex.cpp -std=c++1z -lpthread

dobb2@dobb2-Laptop:~/Рабочий стол/OS_KP$ ./a.out

Enter the number of clients

3

Enter the transfer amount

1000

Select transfer scheme in chronological order:

1 - Bank 1

2 - Bank 2

3 - Bank 3

4 - Bank 4

Enter scheme trans

1 2 3 4 0

Enter the transfer amount

300

Select transfer scheme in chronological order:

1 - Bank 1

2 - Bank 2

3 - Bank 3

4 - Bank 4

Enter scheme trans

1 2 4 3 0

Enter the transfer amount

1500

Select transfer scheme in chronological order:

1 - Bank 1

2 - Bank 2

3 - Bank 3

4 - Bank 4

Enter scheme trans

1 3 4 0

Amount on the Banks accounts

Bank 1: 0

Bank 2: 0

Bank 3: 0

Bank 4: 2500

Amount that may have been in Bank accounts:300

Выводы

Курсовая работа, как и весь курс, показывают, как можно использовать многие полезные утилиты и системные вызовы ОС, как устроены операционные системы в целом, а так же где это может применяться. В данном курсовом проекте я использовал некоторые из уже ранее пройденных в курсе тем: потоки и синхронизация потоков.

Метод избежания взаимоблокировки реализован успешно, но для применения на практических задачах он малопригоден, так как в зависимости от задачи нужно еще продумать умную нумерацию ресурсов, что является уже гораздо более сложной задачей, чем реализация метода. Несмотря на отсутствие как таковой умной нумерации, программа работает корректно без возникновения взаимоблокировок.