Лабораторная работа № 5.

Немецков Михаил, ПМ-31

Задание 1.

Необходимые знания

1. Компилирование программ с помощью gcc.
2. Состояние гонки.
3. Критическая секция.
4. POSIX threads: как создавать, как дожидаться завершения.

Скомпилировать mutex.c без использования и с использованием мьютекса. Объяснить разницу в поведении программы.

2) Состояние гонки или гонки опасности является состояние окружающей электроники , программного обеспечения или другой системы , где основное поведение системы зависит от последовательности или времени других неконтролируемых событий. Это становится ошибкой, когда одно или несколько возможных вариантов поведения нежелательны.

Состояние гонки возникает в программном обеспечении, когда правильная работа компьютерной программы зависит от последовательности или времени программных процессов или потоков . Критические состояния гонки вызывают недопустимое выполнение и программные ошибки . Критические состояния гонки часто возникают, когда процессы или потоки зависят от некоторого общего состояния. Операции с общими состояниями выполняются в критических секциях, которые должны быть взаимоисключающими . Несоблюдение этого правила может привести к повреждению общего состояния. Взаимоисключающие операции - это операции, которые нельзя прервать при доступе к какому-либо ресурсу, например к области памяти.

3) В параллельном программировании одновременный доступ к общим ресурсам может привести к неожиданному или ошибочному поведению, поэтому части программы, в которых осуществляется доступ к общему ресурсу, должны быть защищены способами, исключающими одновременный доступ. Этот защищенный раздел является критическим разделом или критической областью. Он не может выполняться одновременно более чем одним процессом. Обычно критическая секция обращается к совместно используемому ресурсу, такому как структура данных , периферийное устройство или сетевое соединение, которые не будут работать правильно в контексте множественных одновременных доступов.

Пример

Процесс A:

```
// Процесс A
.
.
б  =  x  +  5 ;                  // инструкция выполняется в time = Tx
.
```

Процесс B:

```
// Процесс B
.
.
x  =  3  +  z ;                  // инструкция выполняется в time = Tx
.
```

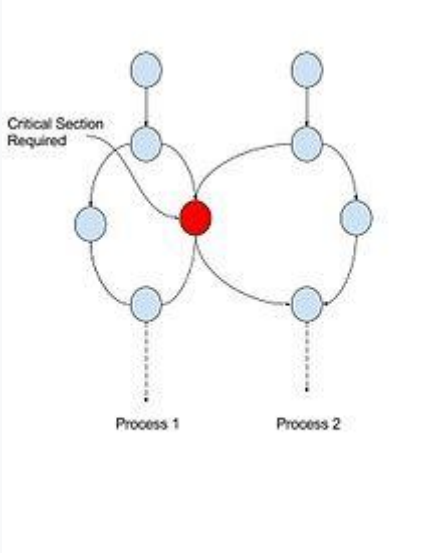

Рис. 1: График, показывающий потребность в критическом участке

В подобных случаях важен критический раздел. В приведенном выше случае, если A необходимо прочитать обновленное значение x , одновременное выполнение процесса A и процесса B может не дать требуемых результатов. Чтобы этого не произошло , переменная x защищена критической секцией. Сначала B получает доступ к разделу. Как только B закончит запись значения, A получит доступ к критическому разделу, и переменная x может быть прочитана.

Компиляция программы

Без мьютекса

~/lab5/src$ ./mutex.exe

doing one thing

counter = 0

doing another thing

counter = 0

doing one thing

doing another thing

counter = 1

counter = 1

doing another thing

counter = 2

doing one thing

counter = 2

doing another thing

counter = 3

doing one thing

counter = 3

doing another thing

counter = 4

doing one thing

counter = 4

doing another thing

counter = 5

doing one thing

counter = 5

doing another thing

counter = 6

doing one thing

counter = 6

doing another thing

counter = 7

doing one thing

counter = 7

doing another thing

counter = 8

doing one thing

counter = 8

doing another thing

counter = 9

doing one thing

counter = 9

doing another thing

counter = 10

doing one thing

counter = 10

doing another thing

counter = 11

doing one thing

counter = 11

doing another thing

counter = 12

doing one thing

counter = 12

doing another thing

counter = 13

doing one thing

counter = 13

doing another thing

counter = 14

doing one thing

counter = 14

doing another thing

counter = 15

doing one thing

counter = 15

doing another thing

counter = 16

doing one thing

counter = 16

doing another thing

counter = 17

doing one thing

counter = 17

doing another thing

counter = 18

doing one thing

counter = 18

doing another thing

counter = 19

doing one thing

counter = 19

doing another thing

counter = 20

doing one thing

counter = 20

doing another thing

counter = 21

doing one thing

counter = 21

doing another thing

counter = 22

doing one thing

counter = 22

doing another thing

counter = 23

doing another thing

counter = 24

doing one thing

counter = 23

doing another thing

counter = 25

doing one thing

counter = 24

doing another thing

counter = 26

doing another thing

counter = 27

doing one thing

counter = 25

doing another thing

counter = 28

doing one thing

counter = 26

doing another thing

counter = 29

doing one thing

counter = 27

doing another thing

counter = 30

doing one thing

counter = 28

doing another thing

counter = 31

doing one thing

counter = 29

doing another thing

counter = 32

doing one thing

counter = 30

doing another thing

counter = 33

doing one thing

counter = 31

doing another thing

counter = 34

doing one thing

counter = 32

doing another thing

counter = 35

doing one thing

counter = 33

doing another thing

counter = 36

doing another thing

counter = 37

doing one thing

counter = 34

doing another thing

counter = 38

doing one thing

counter = 35

doing another thing

counter = 39

doing one thing

counter = 36

doing another thing

counter = 40

doing one thing

counter = 37

doing another thing

counter = 41

doing one thing

counter = 38

doing another thing

counter = 42

doing another thing

counter = 43

doing one thing

counter = 39

doing another thing

counter = 44

doing one thing

counter = 40

doing another thing

counter = 45

doing one thing

counter = 41

doing another thing

counter = 46

doing one thing

counter = 42

doing another thing

counter = 47

doing one thing

counter = 43

doing another thing

counter = 48

doing one thing

counter = 44

doing another thing

counter = 49

doing one thing

counter = 45

doing one thing

counter = 46

doing one thing

counter = 47

doing one thing

counter = 48

doing one thing

counter = 49

All done, counter = 50

Вывод: последовательность выполнения потоков (один за другим, то есть каждый из потоков может иметь одинаковые значения, потому что нет мьютекса, который обеспечивает защиту доступа от других процессов)

Все потоки имеют доступ к данным в один и тот же момент времени!!!

С мьютексом

~/lab5/src$ ./mutex.exe

doing one thing

counter = 0

doing one thing

counter = 1

doing one thing

counter = 2

doing one thing

counter = 3

doing one thing

counter = 4

doing one thing

counter = 5

doing one thing

counter = 6

doing one thing

counter = 7

doing one thing

counter = 8

doing one thing

counter = 9

doing one thing

counter = 10

doing one thing

counter = 11

doing one thing

counter = 12

doing another thing

counter = 13

doing another thing

counter = 14

doing another thing

counter = 15

doing another thing

counter = 16

doing another thing

counter = 17

doing another thing

counter = 18

doing another thing

counter = 19

doing another thing

counter = 20

doing another thing

counter = 21

doing another thing

counter = 22

doing another thing

counter = 23

doing another thing

counter = 24

doing another thing

counter = 25

doing another thing

counter = 26

doing another thing

counter = 27

doing another thing

counter = 28

doing another thing

counter = 29

doing another thing

counter = 30

doing another thing

counter = 31

doing another thing

counter = 32

doing another thing

counter = 33

doing another thing

counter = 34

doing another thing

counter = 35

doing another thing

counter = 36

doing another thing

counter = 37

doing another thing

counter = 38

doing another thing

counter = 39

doing another thing

counter = 40

doing another thing

counter = 41

doing another thing

counter = 42

doing another thing

counter = 43

doing another thing

counter = 44

doing another thing

counter = 45

doing another thing

counter = 46

doing another thing

counter = 47

doing another thing

counter = 48

doing another thing

counter = 49

doing another thing

counter = 50

doing another thing

counter = 51

doing another thing

counter = 52

doing another thing

counter = 53

doing another thing

counter = 54

doing another thing

counter = 55

doing another thing

counter = 56

doing another thing

counter = 57

doing another thing

counter = 58

doing another thing

counter = 59

doing another thing

counter = 60

doing another thing

counter = 61

doing another thing

counter = 62

doing one thing

counter = 63

doing one thing

counter = 64

doing one thing

counter = 65

doing one thing

counter = 66

doing one thing

counter = 67

doing one thing

counter = 68

doing one thing

counter = 69

doing one thing

counter = 70

doing one thing

counter = 71

doing one thing

counter = 72

doing one thing

counter = 73

doing one thing

counter = 74

doing one thing

counter = 75

doing one thing

counter = 76

doing one thing

counter = 77

doing one thing

counter = 78

doing one thing

counter = 79

doing one thing

counter = 80

doing one thing

counter = 81

doing one thing

counter = 82

doing one thing

counter = 83

doing one thing

counter = 84

doing one thing

counter = 85

doing one thing

counter = 86

doing one thing

counter = 87

doing one thing

counter = 88

doing one thing

counter = 89

doing one thing

counter = 90

doing one thing

counter = 91

doing one thing

counter = 92

doing one thing

counter = 93

doing one thing

counter = 94

doing one thing

counter = 95

doing one thing

counter = 96

doing one thing

counter = 97

doing one thing

counter = 98

doing one thing

counter = 99

All done, counter = 100

Вывод: здесь мы не идем по порядку по потокам. Сначала несколько одних, потом несколько других, непоследовательно друг за другом, значения которых не могут совпадать!!!(функция помещает в переменную counter значения. Разные функции не могут поместить одинаковые значения в эту переменную, в этом заключается основная задача мьютекса)

Задание 2.

Написать программу для паралелльного вычисления факториала по модулю `mod` (`k!`), которая будет принимать на вход следующие параметры (пример: `-k 10 --pnum=4 --mod=10`):

1. `k` - число, факториал которого необходимо вычислить.
2. `pnum` - количество потоков.
3. `mod` - модуль факториала

Для синхронизации результатов необходимо использовать мьютексы.

(см.Гитхаб)

```
#include
<pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <getopt.h>
```

```
#include <sys/time.h>


pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;


typedef struct args
{
    uint64_t factorial;
    uint64_t result;
    uint64_t multiplier;
    uint64_t mod;
}args;


void calculations(void* __arg)
{
    pthread_mutex_lock(&mut);
    args* buff = (args*)__arg;
    buff->factorial *= buff->multiplier;
    buff->result *= buff->multiplier;
    buff->result %= buff->mod;
    __arg = (void*)buff;
    pthread_mutex_unlock(&mut);
}




int main(int argc, char* argv[])
{
  uint32_t k = 0;
  uint32_t pnum = 0;
  uint32_t mod = 0;


  static struct option options[] = {{"k", required_argument, 0, 0},
                                    {"pnum", required_argument, 0, 0},
                                    {"mod", required_argument, 0, 0},
                                    {0, 0, 0, 0}};

  while (true) {
    int current_optind = optind ? optind : 1;


    int option_index = 0;
    int c = getopt_long(argc, argv, "", options, &option_index);
```

```
    if (c == -1) break;


    switch (c) {
      case 0:
        switch (option_index) {
          case 0:
            k = atoi(optarg);


            break;
          case 1:
            pnum = atoi(optarg);


            if(pnum == 0)
           {
               printf("Threads number is a positive number\n");
           }


            break;
          case 2:
            mod = atoi(optarg);


            if(mod == 0)
           {
               printf("Module is a positive number.\n");
           }


            break;


          defalut:
            printf("Index %d is out of options\n", option_index);
        }
        break;


      default:
        printf("getopt returned character code 0%o?\n", c);
    }
  }
```

```
  if (optind < argc) {
    printf("Has at least one no option argument\n");
    return 1;
  }


  if (pnum == 0 || mod == 0) {
    printf("Usage: %s --k \"num\" --pnum \"num\" --mod \"num\" \n",
           argv[0]);
    return 1;
  }

  args* arguments = (args*)malloc(sizeof(args));
  arguments->mod = mod;
  arguments->factorial = 1;
  arguments->result = 1;
  pthread_t threads[pnum];


  for(uint64_t i = 0;i < k; i++)
  {
      arguments->multiplier = i + 1;
    if(pthread_create(&threads[i % pnum], NULL, (void*)calculations, (void*)
arguments) != 0)
    {
    perror("pthread_create");
    exit(1);
    }


    if(pthread_join(threads[i % pnum],NULL))
        {
            perror("pthread_join");
            exit(1);
        }
  }


    printf("Factorial %i is equals %i.\n",k,arguments->factorial);
    printf("Factorial %i modulo %i is equals %i.\n",k, mod,arguments-
>result);


    free(arguments);


    return 0;
}
```

~/lab5/src$ ./parallel_factorial.exe --k 3 --pnum=4 --mod=10

Factorial 3 is equals 6.

Factorial 3 modulo 10 is equals 6.

~/lab5/src$ ./parallel_factorial.exe --k 10 --pnum=4 --mod=10

Factorial 10 is equals 3628800.

Factorial 10 modulo 10 is equals 0.

Задание 3.

Необходимые знания

1. Состояние deadlock

Напишите программу для демонстрации состояния deadlock.

Взаимная блокировка (deadlock) — ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение.

Простейший пример взаимной блокировки [править | править код]

Шаг	Процесс 1	Процесс 2
0	Хочет захватить A и B, начинает с A	Хочет захватить A и B, начинает с B
1	Захватывает ресурс A	Захватывает ресурс B
2	Ожидает освобождения ресурса B	Ожидает освобождения ресурса A
3	Взаимная блокировка	

```
#include
<pthread.h>
```

```
#include <stdio.h>
#include <stdlib.h>



int positive = 0;
int negative = 0;
int* empty;
```

```
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;


void increment(void* arg)
{
    printf("Incremented.\n");
    pthread_mutex_lock(&mut);
    positive++;
    pthread_mutex_unlock(&mut);
}




void decrement(void* arg)
{
    printf("Decremented.\n");
    pthread_mutex_lock(&mut);
    negative--;
    increment(empty);
    pthread_mutex_unlock(&mut);
}


int main()
{
pthread_t thr1,thr2;






for(int i = 0; i < 1;i++)
{
    if(pthread_create(&thr1,NULL,(void*)increment,(void*)empty))
    {
        perror("pthread create");
        exit(1);
    }


    if(pthread_create(&thr2,NULL,(void*)decrement,(void*)empty))
    {
        perror("pthread create");
        exit(1);
```

```
        }


        if(pthread_create(&thr2,NULL,(void*)increment,(void*)empty))
        {
            perror("pthread create");
            exit(1);
        }


        if(pthread_create(&thr1,NULL,(void*)decrement,(void*)empty))
        {
            perror("pthread create");
            exit(1);
        }
    }


    pthread_join(thr1,NULL);
    pthread_join(thr2,NULL);




    printf("Positive is %i\n",positive);
    printf("Negative is %i\n",negative);
    return 0;
    }
```

~/lab5/src$ ./deadlock.exe

Incremented.

Decremented.

Incremented.

Decremented.

(первый поток зашел, напечатал инкремент, используем мьютекс, поставили замок, потом посчитали и разблокировали; потом второй поток зашел, напечатал декремент, поставил замок, посчитал, вызвал функцию от инкремента, напечатал инкремент, потом заблочил; затем 1 поток опять заходит в инкремент и все, мьютекс не дает; также и 2 поток(зашел, напечатал декремент и все)