

Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и
информатики

Лабораторная работа №9

Выполнил: студент группы ИП-211

Оганесян Альберт

Лацук Андрей

Проверил:

Профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2024

Задание: протестируйте спин-блокировку используя фрагменты кода лекции 9.

Дополнительное задание: протестировать мьютекс блокировку используя фрагменты кода лекции 9. Сравнить время с спин-блокировкой.

Цель: знакомство с синхронизацией потоков.

1. Используем фрагменты кода из лекции 9, чтобы написать программу:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>

volatile int running = 1;
char sh[6];

void *Thread(void *pParams);

void handle_sigint(int sig)
{
    running = 0;
}

int main(void)
{
    pthread_t thread_id;
    signal(SIGINT, handle_sigint);
    pthread_create(&thread_id, NULL, &Thread, NULL);

    while (running)
    {
        printf("%s", sh);
        fflush(stdout);
    }

    pthread_cancel(thread_id);
    pthread_join(thread_id, NULL);

    return 0;
}

void *Thread(void *pParams)
```

```

{
    int counter = 0;
    while (running)
    {
        if (counter % 2)
        {
            strcpy(sh, "Hello\n");
        }
        else
        {
            strcpy(sh, "Bye_u\n");
        }
        counter++;
    }
    return NULL;
}

```

2. Запустим программу (Рис. 2.1). Основная проблема в том, что потоки работают одновременно, из-за чего основной поток по несколько раз выводит то же значение **sh**.

```

Bye_u
Hello
Hello
Hello
Bye_u
Hello
Bye_u
Bye_u
Hello
Bye_u
Hello
Hello
Hello
Bye_u
Bye_u
Bye_u
Hello
Bye_u
Hello

```

Рис. 2.1 Часть вывода программы

3. Теперь добавим в код спин-блокировку и алгоритм Питерсона, чтобы программа выводила обновленное значение только после измерения и добавим измерения времени:

```

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

```

```
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <sys/times.h>

#define MAX_COUNT 100000

volatile int running = 1;
int turn = 1;
char sh[6];
pthread_spinlock_t spinlock;

void *Thread(void *pParams);
void handle_sigint(int sig)
{
    running = 0;
}

int main(void)
{
    pthread_t thread_id;
    struct tms start_time, end_time;
    clock_t real_start_time, real_end_time;

    pthread_spin_init(&spinlock, PTHREAD_PROCESS_PRIVATE);

    signal(SIGINT, handle_sigint);
    pthread_create(&thread_id, NULL, &Thread, NULL);

    real_start_time = times(&start_time);

    while (running)
    {
        pthread_spin_lock(&spinlock);
        if (turn == 0)
        {
            printf("%s", sh);
            fflush(stdout);
            turn = 1;
        }
    }
}
```

```

        pthread_spin_unlock(&spinlock);
    }

    pthread_cancel(thread_id);
    pthread_join(thread_id, NULL);

    pthread_spin_destroy(&spinlock);

    real_end_time = times(&end_time);

    double user_time = (double)(end_time.tms_utime -
start_time.tms_utime) / sysconf(_SC_CLK_TCK);
    double system_time = (double)(end_time.tms_stime -
start_time.tms_stime) / sysconf(_SC_CLK_TCK);

    printf("User CPU time: %.6f seconds\n", user_time);
    printf("System CPU time: %.6f seconds\n", system_time);

    return 0;
}

void *Thread(void *pParams)
{
    int counter = 0;
    while (counter < MAX_COUNT)
    {
        pthread_spin_lock(&spinlock);
        if (turn == 1)
        {
            if (counter % 2)
            {
                strcpy(sh, "Hello\n");
            }
            else
            {
                strcpy(sh, "Bye_u\n");
            }
            counter++;
            turn = 0;
        }
        pthread_spin_unlock(&spinlock);
    }
}

```

```
    }  
    running = 0;  
    return NULL;  
}
```

4. Сделаем то же самое с Mutex блокировкой:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <signal.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/times.h>  
  
#define MAX_COUNT 100000  
  
volatile int running = 1;  
int turn = 1;  
char sh[6];  
pthread_mutex_t mutex;  
  
void *Thread(void *pParams);  
void handle_sigint(int sig)  
{  
    running = 0;  
}  
  
int main(void)  
{  
    pthread_t thread_id;  
    struct tms start_time, end_time;  
    clock_t real_start_time, real_end_time;  
  
    pthread_mutex_init(&mutex, NULL);  
  
    signal(SIGINT, handle_sigint);  
    pthread_create(&thread_id, NULL, &Thread, NULL);  
  
    real_start_time = times(&start_time);
```

```

while (running)
{
    pthread_mutex_lock(&mutex);
    if (turn == 0)
    {
        printf("%s", sh);
        fflush(stdout);
        turn = 1;
    }
    pthread_mutex_unlock(&mutex);
}

pthread_cancel(thread_id);
pthread_join(thread_id, NULL);

pthread_mutex_destroy(&mutex);

real_end_time = times(&end_time);

double user_time = (double)(end_time.tms_utime -
start_time.tms_utime) / sysconf(_SC_CLK_TCK);
double system_time = (double)(end_time.tms_stime -
start_time.tms_stime) / sysconf(_SC_CLK_TCK);

printf("User CPU time: %.6f seconds\n", user_time);
printf("System CPU time: %.6f seconds\n", system_time);

return 0;
}

void *Thread(void *pParams)
{
    int counter = 0;
    while (counter < MAX_COUNT)
    {
        pthread_mutex_lock(&mutex);
        if (turn == 1)
        {
            if (counter % 2)
            {
                strcpy(sh, "Hello\n");
            }
        }
    }
}

```

```

    }
    else
    {
        strcpy(sh, "Bye_u\n");
    }
    counter++;
    turn = 0;
}
pthread_mutex_unlock(&mutex);
}
running = 0;
return NULL;
}

```

5. Запустим обе программы на значение counter до 100000 сравним время Mutex:

```

Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
User CPU time: 0.950000 seconds
System CPU time: 2.420000 seconds

```

Spin-lock:

```

Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
Bye_u
Hello
User CPU time: 2.240000 seconds
System CPU time: 0.380000 seconds

```

Вывод: Мы научились работать с синхронизацией потоков и сравнили время выполнения mutex и spin-lock. Mutex затрачивает больше системного времени, а Spin-lock пользовательского.