Лабораторная 9

Задание:протестируйте спин-блокировку используя фрагменты кода лекции 9.
Цель:знакомство с синхронизацией потоков.

Спин-блокировка — это блокировка, которая заставляет поток, пытающийся получить её, просто ждать в цикле («спин»), многократно проверяя, доступна ли блокировка.

Фрагменты кода из лекции 9

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
int sh = 0;
int turn = 0;
void* my_thread0() {
  int i = 0;
  for (; i < 100; i++) {
    while (turn);  // spinlock
    sh++;          // критическая область
    turn = 1;
    usleep(1);  // некритическая область
  }
}
void* my_thread1() {
  int i = 0;
  for (; i < 100; i++) {
    while (!turn);
    sh += 2;
    turn = 0;
    usleep(100);
  }
}
int main() {
  pthread_t th_id[2];
  pthread_create(&th_id[0], NULL, &my_thread0, NULL);
  pthread_create(&th_id[1], NULL, &my_thread1, NULL);
  pthread_join(th_id[0], NULL);
  pthread_join(th_id[1], NULL);

  printf("%i\n", sh);

  return 0;
}
```

В коде создаются два потока, которые используют простую спин-блокировку для синхронизации доступа к общей переменной `sh`. Каждый поток выполняет определённые операции в критической области, где доступ к переменной `sh` должен быть синхронизирован.

Спин-блокировка:
Каждый поток использует цикл while для ожидания своей очереди.

Функции потоков:
my_thread0: увеличивает значение sh на 1 в каждой итерации цикла, когда у него есть доступ
my_thread1: увеличивает значение sh на 2 в каждой итерации цикла, когда у него есть доступ

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
int sh = 0;
// int turn=0;
pthread_spinlock_t lock;

void* my_thread0() {
  int i = 0;
  for (; i < 100; i++) {
    // while(turn);
    pthread_spin_lock(&lock);
    sh++;
    // turn=1;
    pthread_spin_unlock(&lock);
    usleep(1);
  }
}
void* my_thread1() {
  int i = 0;
  for (; i < 100; i++) {
    // while(!turn);
    pthread_spin_lock(&lock);
    sh += 2;
    // turn=0;
    pthread_spin_unlock(&lock);
    usleep(1);
  }
}
int main() {
  pthread_t th_id[2];
  pthread_spin_init(&lock, PTHREAD_PROCESS_PRIVATE);
  pthread_create(&th_id[0], NULL, &my_thread0, NULL);
```

```
  pthread_create(&th_id[1], NULL, &my_thread1, NULL);
  pthread_join(th_id[0], NULL);
  pthread_join(th_id[1], NULL);

  pthread_spin_destroy(&lock);
  printf("%i\n", sh);
  return 0;
}
```

Использование pthread_spinlock_t:
спин-блокировка: Использует активное ожидание, где поток постоянно проверяет состояние блокировки. Это может привести к высокой нагрузке на процессор.
спин-блокировка POSIX Threads: Может включать механизмы обратного ожидания (backoff), позволяя потоку временно приостанавливать попытки захвата блокировки, что снижает нагрузку на CPU.

Фрагменты кода к
лабораторной 9

```
char sh[6];
void* Thread( void* pParams );
int main( void ){
  pthread_t thread_id;
  pthread_create(&thread_id, NULL, &Thread, NULL);
  .....
  while( 1 ) printf(“%s\n”, sh);
  .....
}
void* Thread( void* pParams ){
  int counter = 0;
  while ( 1 ){
    if(counter%2){
      sh[0]='H';sh[1]='e';sh[2]='l';sh[3]='l';sh[4]='o';sh[5]='\0';
    }
    else{
      sh[0]='B';sh[1]='y';sh[2]='e';sh[3]='_';sh[4]='u';sh[5]='\0';
    }
    counter++;
  }
  return NULL;
}
```

В текущей реализации программы отсутствует синхронизация между потоками, что может привести к состояниям гонки. Основной поток может читать данные из массива sh, пока другой поток их изменяет, что может привести к некорректному выводу.

Для решения проблемы синхронизации можно использовать спин-блокировку.

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

char sh[6];
pthread_mutex_t mutex;

void* Thread(void* pParams) {
  int counter = 0;
  while (1) {
    pthread_mutex_lock(&mutex);
    if (counter % 2) {
      sh[0] = 'H';
      sh[1] = 'e';
      sh[2] = 'l';
      sh[3] = 'l';
      sh[4] = 'o';
      sh[5] = '\0';
    } else {
      sh[0] = 'B';
      sh[1] = 'y';
      sh[2] = 'e';
      sh[3] = '_';
      sh[4] = 'u';
      sh[5] = '\0';
    }
    pthread_mutex_unlock(&mutex);
    counter++;
    usleep(100000);
  }
  return NULL;
}

int main(void) {
  pthread_t thread_id;
  pthread_mutex_init(&mutex, NULL);
  pthread_create(&thread_id, NULL, &Thread, NULL);

  while (1) {
    pthread_mutex_lock(&mutex);
    printf("%s\n", sh);
    pthread_mutex_unlock(&mutex);
    usleep(100000);
```

```
  }

  pthread_join(thread_id, NULL);
  pthread_mutex_destroy(&mutex);
  return 0;
}
```