Thread barrier

Определение: *Барьер* — это счётчик, который заставляет все треды ждать определённого события. Каждая треда вычитает из этого «счётчика» единицу, и как только значение барьера станет равным 0, все треды продолжают работать.

Синтаксис работы с posix thread барьерами в C++

Как и раньше, для корректной компиляции и запуска программы `bar.c` необходимо сделать 2 шага:

```
gcc bar.c -lpthread -o bar
./bar
```

В начале программы надо объявить глобальную переменную

```
    pthread_barrier_t b;
```

В начале работы `main` необходимо *инициализировать* барьер, указав количество потоков, необходимое, чтобы его «проломить»:

```
  pthread_barrier_init(&b, NULL, T);
```

Внутри каждого потока запускаем функцию ожидания барьера:

```
  pthread_barrier_wait(&b);
```

Пример 1

Данная программа запускает `T` потоков, которые

1. Выводят строку `first`
2. Дожидаются глобальной синхронизации
3. Выводят строку `second`

Код 1: bar.c

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

#define T 10

pthread_barrier_t b;

void * func (void * arg)
{
  printf("first\n");
  pthread_barrier_wait(&b);
  printf("second\n");
  return NULL;
}

int main(int argc, char ** argv)
{
  pthread_t t[T];
  int i;
  void * st;

  pthread_barrier_init(&b, NULL, T);
  for (i = 0; i < T; ++i)
  {
    pthread_create(&t[i], NULL, func, NULL);
  }
  for (i = 0; i < T; ++i)
  {
    pthread_join(t[i], &st);
  }
  return 0;
}
```

Семафоры

На практике синхронизация тысячи потоков может занять длительное время.

Когда мы решаем задачу теплопроводности, необходимо согласовывать только *соседние* потоки.

Вопрос: Как ускорить процесс, используя только локальную синхронизацию?

Это можно сделать несколькими способами

1. Используя механизм критической секции `mutex`, рассмотренный на предыдущем семинаре.
2. Условные переменные (conditional variable, который работает в паре с mutex). Об этом рассказывалось на 3 курсе.
3. Семафоры. Об этом пойдёт речь ниже.

Определение: *Семафор* — это счётчик. Программа ждёт до тех пор, пока семафор не примет положительное значение, затем вычитает из него единицу.

Другими словами, если значение семафора больше нуля, его значение уменьшается на 1, и работа программы продолжается. Если значение равно нулю, то программа ждёт, пока его значение не станет больше нуля, затем из его значения вычитается единичка, и работа программы продолжается.

```
#include <semaphore.h>

sem_t s;//Семафор может быть глобальной переменной

sem_init(&s, 0, 0)//Инициализация семафора
//2 аргумент: флаг
//3 аргумент: начальное значение
sem_wait(&s)//Вычесть единицу в соответствии с написанным выше
sem_post(&s)//Увеличить значение семафора s на единицу
```

Флаг указывает, будет ли данный семафор доступен в других процессах операционной системы. Для задачи теплопроводности это не нужно.

Пример 2

Код 2: bar.c

```c
#include <stdio.h>
#include <unistd.h>//Для функции sleep()
#include <pthread.h>
#include <semaphore.h>

#define T 10

sem_t s;

void * func (void * arg)
{
  printf("first\n");
  sem_wait(&s);
  printf("second\n");
  sem_post(&s);
  return NULL;
}

int main(int argc, char ** argv)
{
  pthread_t t[T];
  int i;
  void * st;

  sem_init(&s, 0, 0);

  for (i = 0; i < T; ++i)
  {
    pthread_create(&t[i], NULL, func, NULL);
  }
  sleep(1);
  for (i = 0; i < T; ++i)
  {
    sem_post(&s);
    pthread_join(t[i], &st);
  }
  return 0;
}
```

Дополнительная задача

Найти интеграл с заданной точностью ε

$$\int_0^1 \frac{1}{x} \sin \frac{1}{x} dx$$

(диффузная балансировка, интеграл с заданной точностью, метод Рунге)