Лекция 10

- Нарушение порядка выполнения операций при оптимизации и кэшировании
- Атомарные функции
- Модели упорядочивания памяти

Изменение порядка выполнения инструкций (REORDERING)

```
int turn=0, flagReady[2]={0,0};

void* thread0( ){
 int i=0;
 for(; i < 100; i++){
```

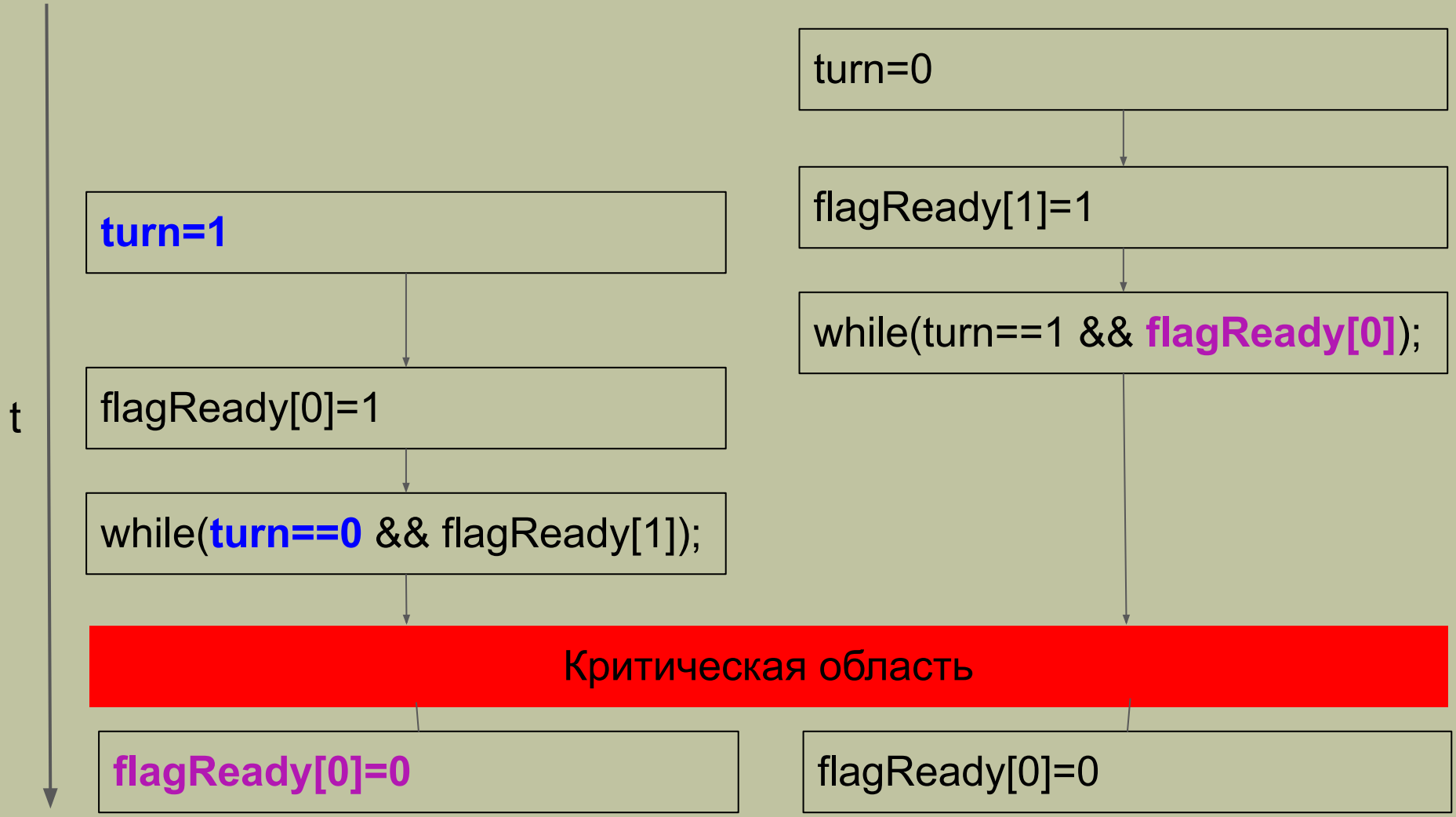
`flagReady[0]=1;` `turn=1;`	`turn=1;` `flagReady[0]=1;`

```
while(turn==1 && flagReady[1]);
   sh++; //критическая область
   flagReady[0]=0;
   usleep(1); //некритическая
 }
}
```

```
void* thread1(){
 int i=0;
 for(;i<100;i++){
```

`flagReady[1]=1;` `turn=0;`	`turn=0;` `flagReady[1]=1;`

```
while(turn==0 && flagReady[0]);
   sh+=2; //критическая область
   flagReady[1]=0; //некритическая
   usleep(1); //sleep(1);
 }
}
```


t
turn=1
flagReady[0]=1
while(turn==0 && flagReady[1]);
turn=0
flagReady[1]=1
while(turn==1 && flagReady[0]);
Критическая область
flagReady[0]=0
flagReady[0]=0

```
#define _GNU_SOURCE
int main(){
  pthread_t th_id[2];
  cpu_set_t cpuset;
  int j;

  pthread_create(&th_id[0], NULL, &my_thread0, NULL);
  pthread_create(&th_id[1], NULL, &my_thread1, NULL);

  CPU_ZERO(&cpuset);
  for (j = 0; j < 1; j++)
    CPU_SET(j, &cpuset);

  pthread_setaffinity_np(th_id[0], sizeof(cpu_set_t), &cpuset);
  pthread_setaffinity_np(th_id[1], sizeof(cpu_set_t), &cpuset);
```

……

```
~Lecture10> ./lab10a
2985
~/Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
2993
~/Lecture10> ./lab10a
2996
```

```
CPU_ZERO(&cpuset);
 for (j = 0; j < 2; j++)
   CPU_SET(j, &cpuset);
```

```
~Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
3000
~/Lecture10> ./lab10a
3000
```

```
CPU_ZERO(&cpuset);
 for (j = 0; j < 1; j++)
   CPU_SET(j, &cpuset);
```

```
~Lecture10> taskset --cpu-list 0-1 ./lab10a
```

Атомарные операции и модели памяти

```
void EnterCriticalRegion(int threadId){
  int x=1;
  int y=1-threadId;
__atomic_store(&readyFlags[threadId], &x, __ATOMIC_SEQ_CST);
__atomic_store(&turn, &y, __ATOMIC_SEQ_CST );
  while (turn == (1- threadId) && readyFlags [1-threadId]);
}

void LeaveCriticalRegion(int threadId){
  int x=0;
__atomic_store(&readyFlags[threadId], &x, __ATOMIC_SEQ_CST);
}
```

```
void EnterCriticalRegion(int threadId){
  int x=1;
  int y=1-threadId;
__atomic_store(&readyFlags[threadId], &x, __ATOMIC_RELAXED);
__atomic_store(&turn, &y, __ATOMIC_RELAXED );
  while (turn == (1- threadId) && readyFlags [1-threadId]);
}

void LeaveCriticalRegion(int threadId){
  int x=0;
__atomic_store(&readyFlags[threadId], &x, __ATOMIC_RELAXED);
}
```

Встроенные аппаратные функции компилятора *gcc*

```
type __atomic_load_n (type *ptr, int memorder)
void __atomic_load (type *ptr, type *ret, int memorder)
void __atomic_store_n (type *ptr, type val, int memorder)
void __atomic_store (type *ptr, type *val, int memorder)
type __atomic_exchange_n (type *ptr, type val, int  memorder)
void __atomic_exchange (type *ptr, type *val, type *ret, int  memorder)
```

```
type __atomic_add_fetch (type *ptr, type val, int memorder)
type __atomic_sub_fetch (type *ptr, type val, int memorder)
type __atomic_and_fetch (type *ptr, type val, int memorder)
type __atomic_xor_fetch (type *ptr, type val, int memorder)
type __atomic_or_fetch (type *ptr, type val, int memorder)
type __atomic_nand_fetch (type *ptr, type val, int  memorder)
```

Модели упорядочивание памяти

__ATOMIC_RELAXED Никаких ограничений на порядок выполнения.

__ATOMIC_ACQUIRE /__ATOMIC_RELEASE Обеспечивается выполнение всех операций перед сохранением/загрузкой данных

__ATOMIC_ACQ_REL Комбинированный эффект __ATOMIC_ACQUIRE и __ATOMIC_RELEASE.

__ATOMIC_SEQ_CST (согласованная последовательность) Сохраняет полный порядок операций выполняемых с опцией __ATOMIC_SEQ_CST.

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

int sh=0;
void* my_thread0( ){
  int i=0;
  for(; i < 1000; i++){
    //  EnterCriticalRegion(0);
     __sync_fetch_and_add(&sh, 1.0);
    //   LeaveCriticalRegion(0);

    usleep(1);
  }
}
```

```
void* my_thread1(){
  int i=0;

  for(;i<1000;i++){
   //  EnterCriticalRegion(1);
   __atomic_fetch_add(&sh, 2.0, __ATOMIC_RELAXED);
  //   LeaveCriticalRegion(1);
      usleep(1);
  }
}
```

```
int main(){
  pthread_t th_id[2];
  int j;

  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;
}
```

```
gcc lab10c.c -lpthread  -o lab10c
```

Атомарные функции в C++11

```
#include <atomic>
#include <iostream>

std::atomic<int>  sh=0;

void my_thread0( ){
  int i=0;

  for(; i < 1000; i++){
    //EnterCriticalRegion(0);
    sh.fetch_add(1, std::memory_order_relaxed); //критическая область
    //LeaveCriticalRegion(0);
    usleep(1); //некритическая
  }
}
```

```
void my_thread1(){
  int i=0;

  for(;i<1000;i++){
  //  EnterCriticalRegion(1);
   sh.fetch_add(2, std::memory_order_relaxed);  //критическая
  //   LeaveCriticalRegion(1);
     //некритическая область:
     usleep(1); //sleep(1);
  }
}
```

```
int main(){
  std::thread th1(my_thread0);
  std::thread th2(my_thread1);

  th1.join();
  th2.join();

  std::cout<<sh<<std::endl;

  return 0;
}
```

```
g++ -std=c++17 lab10c.cpp -lpthread -o lab10cc
```