جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

Lab 7
Process Synchronisation

Objective

- ❑ To learn how to synchronize threads using pthread mutex lock.

Multithreading Concepts

Concurrent access to shared data may result in **data inconsistency**.

Race condition – several processes (or threads) access and manipulate the same data concurrently. The **outcome** depends on the **order** of the **access**.

Critical section – a **code segment** where process (or threads) may change **shared** variables, updating a table, writing a file etc.

Solution – ensure **mutual exclusion**, that is only one process (or thread) access the critical section at any one time.

pthreads Synchronization

- **pthreads** includes support for **Mutual Exclusion primitives**.
- A **mutex** is a lock that is set before using a shared resource and release after using it.
- When the lock is **set**, no other thread can access the locked region.
- The idea is **to lock the critical section of the code before accessing global variables and to unlock as soon as you are done**.

Mutex Declaration

- A global variable of type **pthread_mutex_t** is **required** and it’s defined as the following:

```
pthread_mutex_t     mutex = PTHREAD_MUTEX_INITIALIZER;
```

Mutex States

- A mutex has two possible states:
  - **unlocked** (not owned by any thread),
  - **locked** (owned by one thread).
- A mutex can never be owned by two different threads simultaneously.
- A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.
- **To lock use:**

```
pthread_mutex_lock(&mutex);
```

- **To unlock use:**

```
pthread_mutex_unlock(&mutex);
```

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

Demo

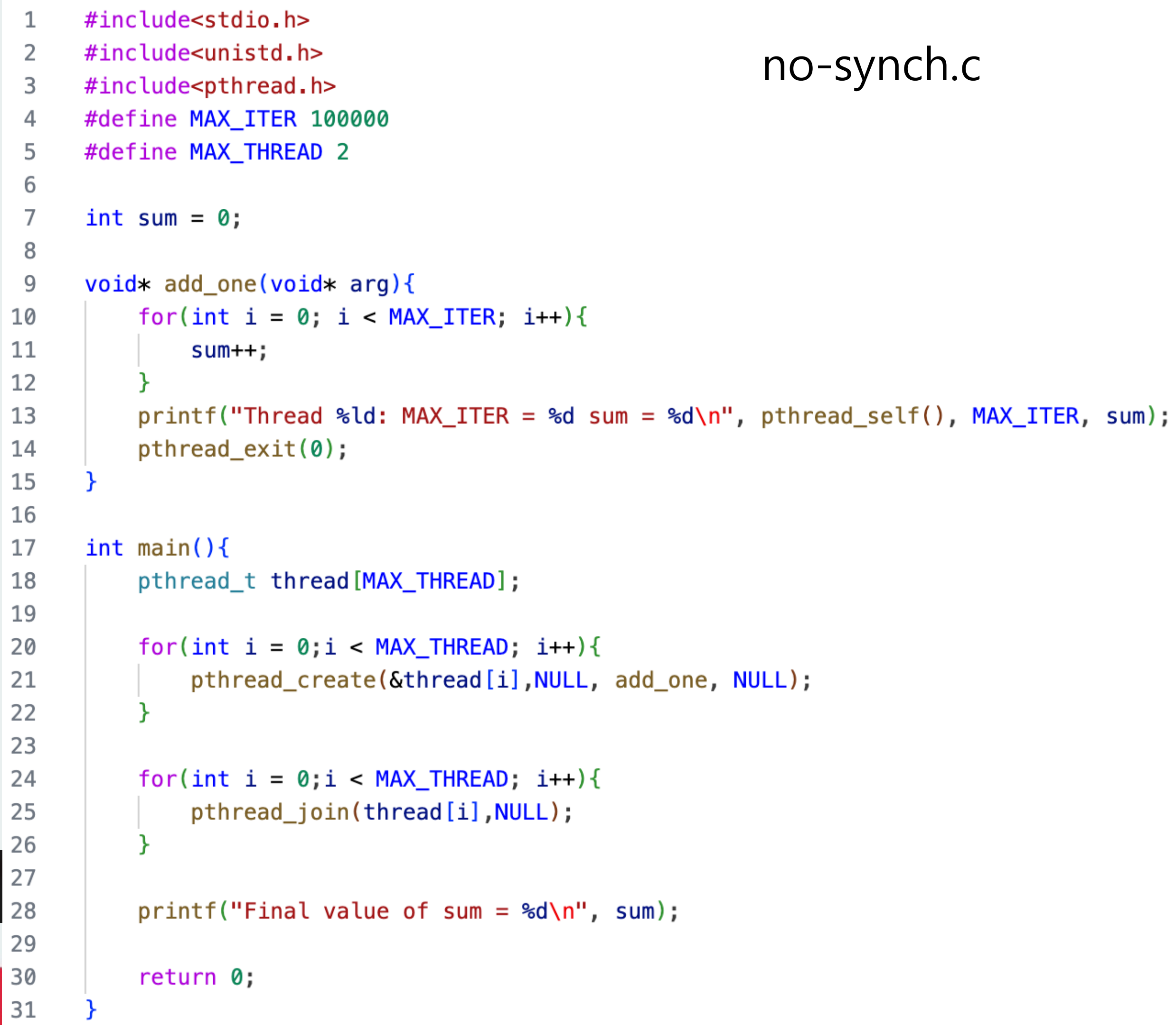
No Synchronization

no-synch.c

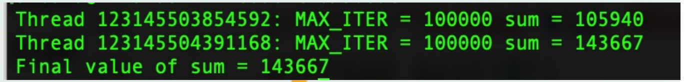
Recall the program that you wrote in Lab 5, which does a summation by 1. Run the program with a large value of MAX_ITER, e.g. 100,000 or 1,000,000

Incorrect result when MAX_ITER is high

Problem => Race Condition



```c
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#define MAX_ITER 100000
#define MAX_THREAD 2

int sum = 0;

void* add_one(void* arg){
    for(int i = 0; i < MAX_ITER; i++){
        sum++;
    }
    printf("Thread %ld: MAX_ITER = %d sum = %d\n", pthread_self(), MAX_ITER, sum);
    pthread_exit(0);
}

int main(){
    pthread_t thread[MAX_THREAD];

    for(int i = 0;i < MAX_THREAD; i++){
        pthread_create(&thread[i],NULL, add_one, NULL);
    }

    for(int i = 0;i < MAX_THREAD; i++){
        pthread_join(thread[i],NULL);
    }

    printf("Final value of sum = %d\n", sum);

    return 0;
}
```



```
Thread 123145503854592: MAX_ITER = 100000 sum = 105940
Thread 123145504391168: MAX_ITER = 100000 sum = 143667
Final value of sum = 143667
```

Solution to race condition

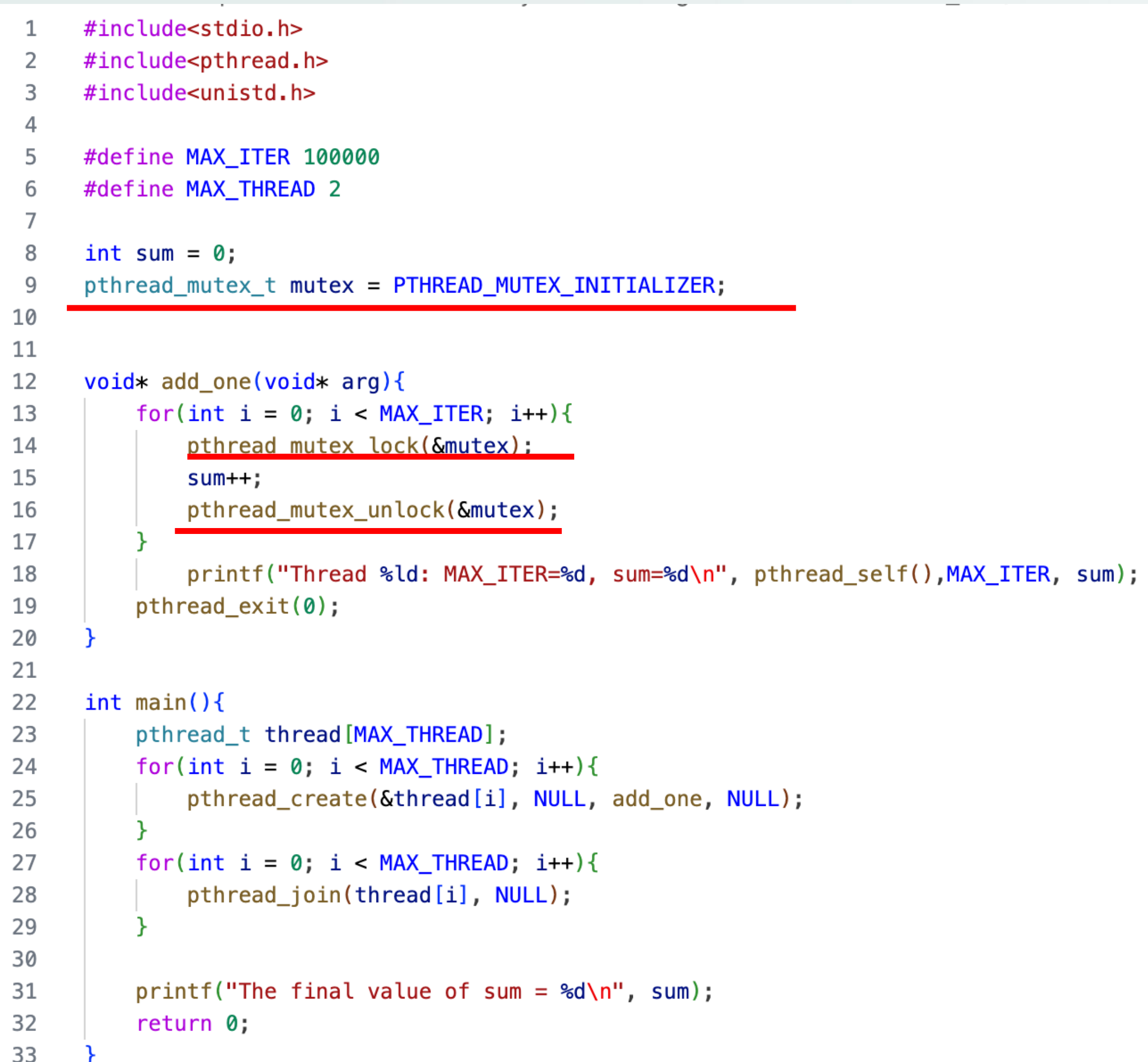
- The execution of threads can be synchronise using the following steps:

1. Declare a variable of type pthread_mutex_t and initialise it.

   E.g. `pthread_mutex_t  m = PTHREAD_MUTEX_INITIALIZER;`

2. Encapsulate the line that causes race condition, that is the critical section, with:

   `pthread_mutex_lock(&m);` and
   `pthread_mutex_unlock(&m);`



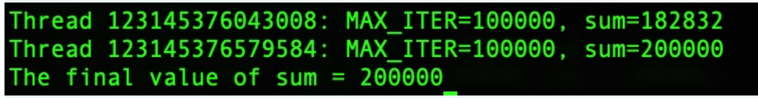
```c
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>

#define MAX_ITER 100000
#define MAX_THREAD 2

int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;


void* add_one(void* arg){
    for(int i = 0; i < MAX_ITER; i++){
        pthread_mutex_lock(&mutex);
        sum++;
        pthread_mutex_unlock(&mutex);
    }
        printf("Thread %ld: MAX_ITER=%d, sum=%d\n", pthread_self(),MAX_ITER, sum);
    pthread_exit(0);
}

int main(){
    pthread_t thread[MAX_THREAD];
    for(int i = 0; i < MAX_THREAD; i++){
        pthread_create(&thread[i], NULL, add_one, NULL);
    }
    for(int i = 0; i < MAX_THREAD; i++){
        pthread_join(thread[i], NULL);
    }

    printf("The final value of sum = %d\n", sum);
    return 0;
}
```

With Synchronisation

To synchronise:

1. Declare a variable of type
   `pthread_mutex_t  m`
2. Encapsulate the line that causes ra
   ce condition with:

`pthread_mutex_lock(&m)` and
`pthread_mutex_unlock(&m)`



```
Thread 123145376043008: MAX_ITER=100000, sum=182832
Thread 123145376579584: MAX_ITER=100000, sum=200000
The final value of sum = 200000
```

Resources:

1. Safety and Speed Issues with Threads. (pthreads, mutex, locks) https://www.youtube.com/watch?v=9axu8CUvOKY

2. What are Race Conditions? https://www.youtube.com/watch?v=FY9livorrJI

3. What is a mutex in C? (pthread_mutex) https://www.youtube.com/watch?v=oq29KUy29iQ