Paralelné programovanie
Monitory

doc. Ing. Michal Čerňanský, PhD.
FIIT STU Bratislava

Posix Threads

- > gcc monitor.c -lpthread -o monitor
- > ./monitor

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

const int no_threads = 10;
int counter = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *thread_fnc(void *arg) {
    while (1) {
        pthread_mutex_lock(&mutex);
        while (counter % no_threads != (long long)arg)
            pthread_cond_wait(&cond, &mutex);
        usleep(100000);
        counter++;
        printf("T%lld\n", (long long)arg);
        pthread_cond_broadcast(&cond);
        pthread_mutex_unlock(&mutex);
        if (counter > 100) return NULL;
    }
}

int main(void) {
    pthread_t threads[no_threads];
    for (long long i=0; i<no_threads; i++)
        pthread_create(&threads[i], NULL, &thread_fnc, (void*) i);

    for (int i=0; i<no_threads; i++)
        pthread_join(threads[i], NULL);

    printf("Counter is %d\n", counter);
    exit(EXIT_SUCCESS);
}
```

C# Threads (Mono)

- > mcs monitor.cs
- > mono monitor.exe

```
using System;
using System.Threading;

public class Program
{
    private static int Counter = 0;
    private static System.Object lockObject = new System.Object();
    private const int noOfThreads = 10;

    private static void ThreadFunction(Object m) {
        while (true) {
            Monitor.Enter(lockObject);
            try {
                while (Counter%noOfThreads != (int)m)
                    Monitor.Wait(lockObject);
                Thread.Sleep(100);
                Counter += 1;
                Console.WriteLine("{0} ",Thread.CurrentThread.Name);
                Monitor.PulseAll(lockObject);
                if (Counter > 100) return;
            } finally {
                Monitor.Exit(lockObject);
            }
        }
    }

    public static void Main(string[] args) {
        Thread[] threads = new Thread[noOfThreads];
        for (int i=0; i<noOfThreads; i++) {
            threads[i] = new Thread(ThreadFunction );
            threads[i].Name = string.Format("T{0}", i);
            threads[i].Start(i);
        }

        for (int i=0; i<noOfThreads; i++)
            threads[i].Join();

        Console.WriteLine("Counter is {0}", Counter);
    }
}
```

C# Threads (Mono)
Task-based Asynchronous Programming

- > mcs monitor.cs
- > mono monitor.exe

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

public class Program
{
    private static int Counter = 0;
    private static System.Object lockObject = new System.Object();
    private const int noOfThreads = 10;

    public static void Main()
    {
        List<Task> tasks = new List<Task>();
        for (int ti=0; ti<noOfThreads; ti++)
            tasks.Add(Task.Factory.StartNew( (object m) => {
                while (true) {
                    Monitor.Enter(lockObject);
                    while (Counter%noOfThreads != (int)m)
                        Monitor.Wait(lockObject);
                    Thread.Sleep(100);
                    Counter += 1;
                    Console.WriteLine("T{0} ", m);
                    Monitor.PulseAll(lockObject);
                    Monitor.Exit(lockObject);
                    if (Counter > 100) return;
                }
            }, (ti) ));

        Task.WaitAll(tasks.ToArray());

        Console.WriteLine("Counter is {0}", Counter);
        Console.ReadKey();
    }
}
```

Java Threads

- > javac Monitor.java
- > java Monitor

```
public class Monitor {

    private static int counter = 0;
    private static int noOfThreads = 10;
    private static Object monitor = new Object();

    static class MyThread extends Thread {
        public int m;
        public void run() {
            try {
                while (true) {
                    synchronized (monitor) {
                        while (counter%noOfThreads != m) monitor.wait();
                        sleep(100);
                        counter += 1;
                        System.out.println(getName());
                        monitor.notifyAll();
                        if (counter > 100) return;
                    }
                }
            }
            catch (InterruptedException e) {
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {

        MyThread[] threads = new MyThread[noOfThreads];
        for (int i=0; i<noOfThreads; i++) {
            threads[i] = new MyThread();
            threads[i].m = i;
            threads[i].setName("T"+i);
            threads[i].start();
        }

        for (int i=0; i<noOfThreads; i++)
            threads[i].join();

        System.out.println("Counter is " + counter);
    }
```

C++ 11 Threads

- > g++ -std=c++11 monitor.cpp -o monitor
- > ./monitor

```cpp
#include <iostream>
#include <vector>
#include <thread>

static const int no_threads = 10;

int main() {

    std::thread threads[no_threads];
    std::mutex lock;
    std::condition_variable cond;
    int counter = 0;

    for (int i=0; i<no_threads; i++) {
        threads[i] = std::thread([&,i] {
            while(true) {
                std::unique_lock<std::mutex> locker(lock);

                while (counter % no_threads != i) cond.wait(locker);
                std::this_thread::sleep_for(std::chrono::milliseconds(100));
                counter += 1;
                std::cout << "T" << i << std::endl;
                cond.notify_all();
                if (counter > 100) return;
            }
       });
    }

    for (auto&& t : threads)
        t.join();

    std::cout << "Counter is " << counter << std::endl;

    return 0;
}
```

Go Threads

- > go build monitor.go
- > ./monitor

```go
package main

import (
    "fmt"
    "sync"
    "time"
)

const noThreads = 10

var counter int = 0

var mutex *sync.Mutex
var cond *sync.Cond
var wg sync.WaitGroup

func threadFnc(mycnt int) {
    for {
        mutex.Lock()
        for counter%noThreads != mycnt {
            cond.Wait()
        }

        time.Sleep(10 * time.Millisecond)

        counter++
        fmt.Printf("%d\n", mycnt)

        cond.Broadcast()
        mutex.Unlock()

        if counter > 100 {
            wg.Done()
            return
        }
    }
}

func main() {
    mutex = &sync.Mutex{}
    cond = sync.NewCond(mutex)

    for i := 0; i < noThreads; i++ {
        go threadFnc(i)
        wg.Add(1)
    }

    wg.Wait()
    fmt.Printf("Counter is %d\n", counter)
}
```

Python Threads

- > python monitor.py

```python
import threading
import time

no_threads = 10

cond = threading.Condition()
counter = 0

def thread_fnc(my_cnt):
    global counter

    while True:
        cond.acquire()
        while counter % no_threads != my_cnt:
            cond.wait()

        time.sleep(0.1)
        counter += 1

        print(my_cnt)

        cond.notifyAll()
        cond.release()

        if counter > 100:
            break


threads = list()
for i in range(no_threads):
    thread = threading.Thread(target=thread_fnc, args=(i,))
    threads.append(thread)
    thread.start()

for index, thread in enumerate(threads):
    thread.join()

print("Counter is ", counter)
```