C Grundlagen - Threads

Michael Strassberger
saremox@linux.com

Proseminar C Grundlagen
Fachbereich Informatik
Fakultaet fuer Mathematik, Informatik und Naturwissenschaften
Universitaet Hamburg

3. Juli 2014

Table of Contents

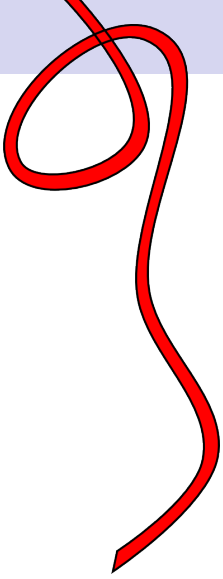
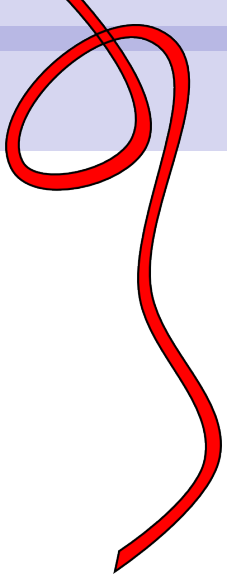


Table of Contents



Processing speed is limited

Physical Limits

- Electrical signal speed (RS)
- Cooling the heat

Workarounds from Hardware manufactures

- Multiple CPU Cores ←
- Advanced IS
- MMX etc

Types of computing problems

Problems that can be seperated

- Compression
  - LempelZivMarkov chain algorithm
- Simulations
  - Weather
  - Chemical reactions
  - Satelite movements
- Sort algorithm
- Arithmetic

Problems that can‘t be seperated

- File I/O
- Network I/O
- Hardware Access
  - Sound
  - HDD
- Graphical user Interface
  - GTK
  - QT

Airthmetic Example

$$\sum_{i=0}^{1000}(i^2 + 5 \cdot i + 5)^{10}$$

Part #1

$$\text{res1} = \sum_{i=0}^{500}(i^2 + 5 \cdot i + 5)^{10}$$

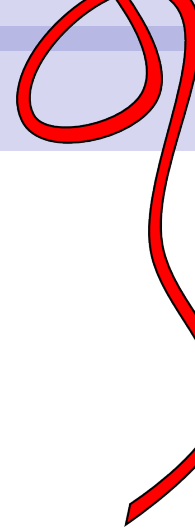
Part #2

$$\text{res2} = \sum_{i=501}^{1000}(i^2 + 5 \cdot i + 5)^{10}$$

Join parts together

$$\text{int result} = res1 + res2;$$

Table of Contents



What are threads

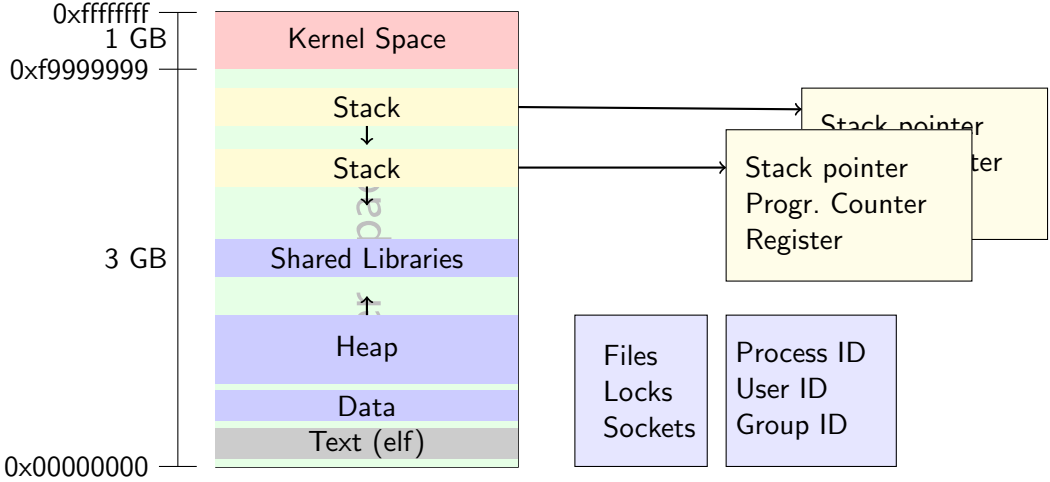
Definition

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. [2]

Properties of threads

- Small memory footprint
- OS can map threads to diffrent CPU‘s

How Unix processes are organized



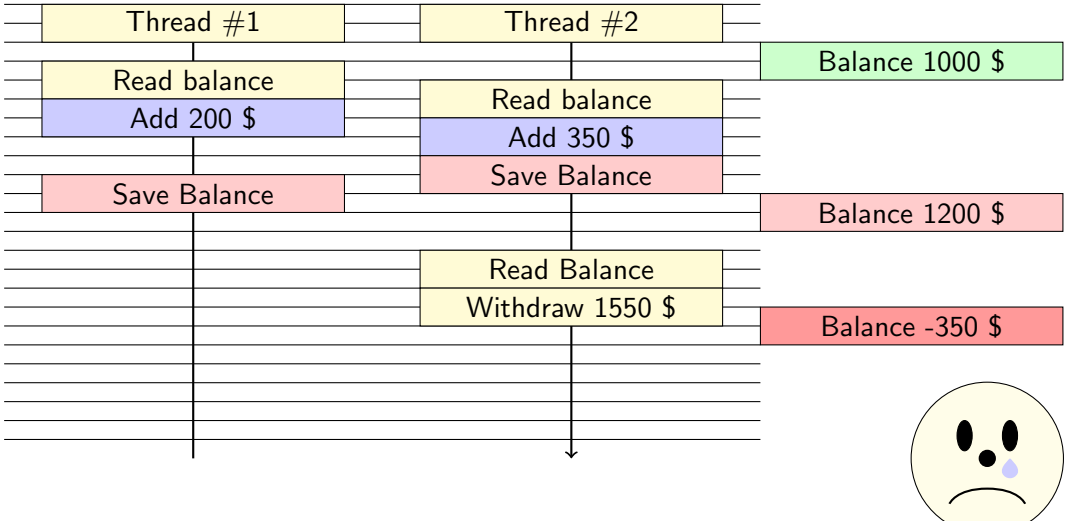
[1]

With great power comes great responsibillity

Problems that occur by using threads

- Race Condition
- Communication between threads
- Dead locks
- Live locks

Race Condition

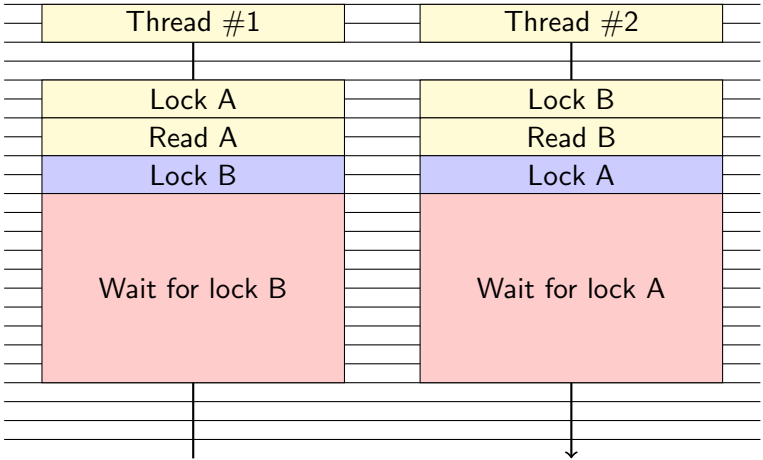


But there is hope

Locks / Mutex

- primary implementation for thread syncronisation
- mutexes can prevent race conditions
- It‘s up to the programmer to ensure locking and unlocking mutexes

But wait...

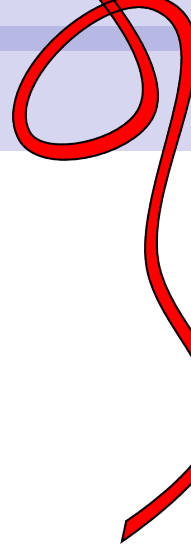

Thread #1
Thread #2
Lock A
Lock B
Read A
Read B
Lock B
Lock A
Wait for lock B
Wait for lock A

Summary

What we now know about threads

- Threads have a small memory footprint
- We‘ve to be careful using threads
  - Consistence of shared data
  - Syncronisation

Table of Contents



Overview

Libraries

- pthread ←
- OpenMP ←
- c11
- glib
- Qt-Threads
- many more...

What is PThread

PThread

- In History each hardware vendor had developed his own implementation of threads
- Pthread is a standardized programming interface
  - defined for unix in IEEE POSIX 1003.1c standard (1995)
- Hardware vendors began to offer pthread implementations

Compiling

gcc

gcc -std=c11 -lpthread ¡file¿ -o ¡output¿

Makefile

```
1 Account : account.o
2   cc -lpthread -O3 account.o -o Account
3 account.o : main.c
4   cc -std=c11 -c -lpthread -lm -O3 main.c -o account.o
```

Quick Overview

Data Types for pthread

```
1 pthread_t thread
```

Important function calls

```
1 pthread_create (thread,attributes,start_routine,argument)
2 pthread_exit (status)
3 pthread_cancel (thread)
```

pthread Example

Hello World

```
1  void * sayhello(void * arg)
2  {
3    long threadID = (long) arg;
4    fprintf(stdout,"Hello from Thread #%d\n",threadID);
5    return 0;
6  }
7  int main(int argc, char* argv[])
8  {
9    pthread_t thread[20];
10   for(long threadid = 1; threadid < 20; threadid++)
11     pthread_create(&thread[threadid], NULL,
12                    sayhello,(void *)threadid);
13   pthread_exit(NULL);
14 }
```

Output

Hello World

```
1  Hello from Thread #4
2  Hello from Thread #5
3  Hello from Thread #6
4  Hello from Thread #7
5  Hello from Thread #9
6  Hello from Thread #10
7  Hello from Thread #8
8  Hello from Thread #11
9  Hello from Thread #12
10 Hello from Thread #13
11 Hello from Thread #14
12 Hello from Thread #19
13 Hello from Thread #18
14 Hello from Thread #16
15 Hello from Thread #17
```

passing Arguments & Joining

Job Structure

```
1 struct job {
2     int start;
3     int end;
4     unsigned long long int result;
5 };
```

void * function(void *ptr) {

```
1     struct job *myJob = (struct job*) ptr;
2     for(int i = myJob->start ; i <= myJob->end ; i++)
3     {
4         myJob->result += (5*(i*i) + 5);
5     }
```

passing Arguments & Joining

int main() {

```
1     struct job job1 = {0,500,0};
2     struct job job2 = {501,1000,0};
3     pthread_t worker1;
4     pthread_t worker2;
5
6     pthread_create(&worker1,NULL,function,(void *) &job1);
7     pthread_create(&worker2,NULL,function,(void *) &job2);
8     pthread_join(worker1,NULL);
9     pthread_join(worker2,NULL);
10    fprintf(stdout,"Result: %llu + %llu = %llu \n",job1.result,job2.result,(
          job1.result+job2.result));
```

Mutex

Data Types for pthread

```
1 pthread_mutex_t mutex
```

Create and Destroy

```
1 pthread_mutex_init (mutex,attr)
2 pthread_mutex_destroy (mutex)
```

Locking

```
1 pthread_mutex_lock (mutex)
2 pthread_mutex_trylock (mutex)
3 pthread_mutex_unlock (mutex)
```

int transfer(struct account *from, struct account *to, int ammount)

```
1     pthread_mutex_lock(&from->lock);
2     if(from->balance > ammount) {
3         if(pthread_mutex_trylock(&to->lock) == 0) {
4             from->balance -= ammount;
5             to->balance   += ammount;
6             pthread_mutex_unlock(&to->lock);
7         } else {
8             pthread_mutex_unlock(&from->lock);
9             return transfer(from,to,ammount);
10        }
11        pthread_mutex_unlock(&from->lock);
12        fprintf(stdout,"| %s\t | %d\t |\n",to->name,to->balance);
13        return 0;
14    } else {
15        pthread_mutex_unlock(&from->lock);
16        return -1;
17    }
```

OpenMP

What is OpenMP

- API for using multiple threads on MPM
- Supports C / C++ and Fortran
- Is a set of compile directives
- Easy to use

compiling

gcc -std=c11 -fopenmp ¡file¿ -o ¡outputfile¿

Example

Hello World

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   #pragma omp parallel
6     printf("Hello, world.\n");
7   return 0;
8 }
```

Table of Contents

Summary

What we‘ve learned

- With threads we can access multiple CPU‘s
- Corecctly used they offer more Performance on MPM
- We have to syncronize access to shared data
- Take care of Race Conditions, Dead locks and Live Locks

Thank you for your attention

Table of Contents

Literature


Blaise Barney.
Posix threads programming.
https://computing.llnl.gov/tutorials/pthreads/.


Many.
Thread (computing).
http://en.wikipedia.org/wiki/Thread_(computing).