


Ferdinand-Universität in Hagen Department of Mathematics and Computer Science



**Very Brief Introduction to Shared Memory Programming with POSIX Threads**  
Course 01727 Parallel Programming

Parallelism and VLSI Group  
Prof. Dr. Jörg Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

## POSIX Threads

- Pthreads are a library for C programming
- Contain functions to start and terminate threads to coordinate threads to regulate access to shared data structures
- Note: as a library, they rely on underlying OS and hardware!

Slide 2 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

## Starting a Thread I

- Thread is started with a particular function  
Called func must have parameter and ret values void\*  
Exception: first thread is started with main()
- Thread terminates when called function terminates or by pthread\_exit(void \*retval)
- Threads started one by one
- Threads represented by data structure of type pthread\_t

Slide 3 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

## Starting a Thread II

- Example

```
#include <pthread.h>
int main(int argc, char *argv[]){
    int *ptr;
    pthread_t thr;

    pthread_create(&thr, NULL, foo, (void*)ptr);
    ...
    pthread_join(&thr, NULL);
    return 0;
}
```

Slide 4 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

## Starting a Thread III

- `void *foo(void *vp){  
 int i = (int)vp;;  
 ...  
}`
- `void *foo(void *vp){  
 Userdefinedstructtype *ptr;  
 ptr=(Userdefinedstructtype*)vp;  
 ...  
}`

Slide 5 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Universität in Hagen Department of Mathematics and Computer Science

## Access to Shared Data I

- Globally defined variables visible to all threads
- Locally defined variables visible to the thread executing the function
- But all data in shared memory  
publish address of data: all threads can access
- Take care: typically no protection between thread data  
thread1 can even write to thread2 stack frame

Slide 6 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Access to Shared Data II

- Example

```
int *globalptr = NULL;

void *foo1(void *ptr1){
  int i = 15;
  globalptr = &i; // globalptr = (int*)malloc(sizeof(int));
  ... }

void *foo2(void *ptr2){
  if(globalptr) *globalptr = 17;
  ... }
```

Slide 7 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Access to Shared Data III

- Left variant extremely dangerous!
- if foo1 terminates then foo2 writes somewhere if globalptr is not manually reset to NULL value
- Right variant is ok as long as garbage collection is ok

Slide 8 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Coordinating Shared Access I

- What to do if several threads need to write shared var
- If they simply write: ok if write order does not play a role
- If they read and write: encapsulate
- Example: access to taskpool threads maintain list of tasks to be performed if thread is idle, gets a task and performs it

Slide 9 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Coordinating Shared Access II

- While(!workdone){
 task = gettask(Pooldescr);
 performtask(task);
 }
- Tasktype gettask(Pool p){
 task = p.queue[p.index]; p.index++;
 return task;
 }

Slide 10 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Coordinating Shared Access III

- Shared access to workdone ok, as long as only 1 writer
- p.index read (twice) and written
- If two threads enter gettask concurrently, both may read old value (e.g. 5), increment, write value 6
- Correct value would be 7
- Protect access to p.index by critical section only 1 thread can be within critical section

Slide 11 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

Ferdinand-Humboldt-Universität in Hagen Department of Mathematics and Computer Science

## Coordinating Shared Access IV

```
pthread_mutex_t mutex; // global variable definition

pthread_mutex_init(&mutex,NULL); // init in main()

// in gettask
pthread_mutex_lock(&mutex);
task = p.queue[p.index]; p.index++;
pthread_mutex_unlock(&mutex);
```

Slide 12 Course 01727 Parallel Programming Parallelism and VLSI Group Prof. Dr. J. Keller

## Coordinating Shared Access V

- If trying to lock mutex that is locked thread is blocked until mutex unlocked again
- Must rely on implementation to be fair i.e. no request to lock mutex may overtake another  
Otherwise: starvation of thread possible
- Beware of deadlocks i.e. situations where mutex is not unlocked sometimes
- Example: if-statement in critical section, unlock in then part



## Coordinating Shared Access VI

- Must also rely on implementation for efficiency
- Time to lock / unlock mutex or synchronize threads varies widely between different platforms
- Note: mutex that all threads access serializes threads  
Make critical section as short as possible

```
pthread_mutex_lock(&mutex);
tmpindex = p.index++;
pthread_mutex_unlock(&mutex);
task = p.queue[tmpindex];
```



## Coordinating Shared Access VII

- When programming on this level of abstraction can minimize serialization, but not avoid
- Better: avoid mutex and similar constructs use higher-level data structures that are lock-free
- Example: NOBLE

