**Aufgabe 1: Posix Threads**

**Robin Wismeth, Simon Neureither**

**Erzeugen und Beenden von Threads:**

Erzeugt werden Threads mit pthread\_create. Dieser Funktion können Thread Parameter mitgegeben werden, die Dinge wie den Detached/Joinable Zustand aber auch andere Zustände regeln.

**Problematik DETACHED/JOINABLE:**

Während …DETACHED einen Thread in einen DETACHED Zustand erstellt, erstellt …JOINABLE einen Thread in einen Zustand in dem ein anderer Thread auf diesen warten kann.

Standardmäßig sind Threads …JOINABLE.

**Übergabe von Parametern über “generische Pointer (void \*)“:**

Der Voidpointer wird als Platzhalter für verschiedenste Datentypen verwendet. Dadurch ist es möglich eine generische Funktion in C zu formulieren.

**Umgang mit Returncodes:**

Bei Systemfunktionen sollte der Returncode überprüft werden. Es gibt zwei verschiedene Arten diese Returncodes zu überprüfen:

1. Funktion gibt -1 bei Fehler zurück (Fehler kann über perror zurückgegeben werden, nicht threadsafe)

if (<func(...)>)

{

perror("func failed");

<Fehlerbehandlung>

}

1. Funktion gibt Errorcode direkt zurück. (strerror liefert den entsprechenden Fehlertext zurück, da der Fehlercode nicht zentral gespeichert wird ist diese Art der Fehlerbehandlung threadsafe.)

int ret;

if (ret = <func(...)>)

{

fprintf(stderr, "func failed: %s", strerror(ret));

<Fehlerbehandlung>

}

**Sourcecode**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

void\* task(void\* arg)

{

sleep(\*((int\*)arg));

printf("Einen Text %u\n", pthread\_self());

return (void\*)pthread\_self();

}

int main( void )

{

pthread\_attr\_t attr;

pthread\_attr\_init( &attr );

pthread\_t thread1;

pthread\_t thread2;

// Zeiten, die unsere Tasks warten sollen.

int waitThread1 = 5;

int waitThread2 = 2;

// Starte Thread 1,2

int ret = pthread\_create( &thread1, &attr, &task, &waitThread1 );

if (ret != EOK)

{

fprintf(stderr, "pthread\_create: %s", strerror(ret));

return EXIT\_FAILURE;

}

ret = pthread\_create( &thread2, &attr, &task, &waitThread2 );

if (ret != EOK)

{

fprintf(stderr, "pthread\_create: %s", strerror(ret));

return EXIT\_FAILURE;

}

pthread\_t taskID1;

pthread\_t taskID2;

// Warte auf Thread 1,2

ret = pthread\_join(thread2, (void\*)&taskID2);

if (ret != EOK)

{

printf("pthread\_join: %s", strerror(ret));

return EXIT\_FAILURE;

}

ret = pthread\_join(thread1, (void\*)&taskID1);

if (ret != EOK)

{

printf("pthread\_join: %s", strerror(ret));

return EXIT\_FAILURE;

}

// Überprüfe Thread Ids.

if (taskID1 != thread1)

{

printf("Error thread ids do not match: %u %u\n", thread1, taskID1);

return EXIT\_FAILURE;

}

if (taskID2 != thread2)

{

printf("Error thread ids do not match: %u %u\n", thread2, taskID2);

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}