Aufgabe 3:

**Machen Sie mehrere Messungen. Sind die Ergebnisse ausrechend konstant?**

Messungen bei waste\_msecs mit 2000 als Parameter das gleiche Ergebnis:

Messung mit 2000ms:

thread priority: 255

Verbrauchte Zeit s: 2,0

Verbrauchte Zeit s: 2,0

Verbrauchte Zeit s: 2,0

Verbrauchte Zeit s: 2,0

Verbrauchte Zeit s: 2,0

Messung mit 3000ms:

thread priority: 255

Verbrauchte Zeit s: 3,0

Verbrauchte Zeit s: 3,0

Verbrauchte Zeit s: 3,0

Verbrauchte Zeit s: 3,0

Verbrauchte Zeit s: 3,0

Auffällig ist, dass die Messergebnisse abhängig sind von der Thread-Priorität (je höher die Priorität, umso „schneller“ läuft waste\_msecs). Wenn waste\_msecs allerdings mit niedriger Priorität gestartet wird, wird die Abweichung zunehmend größer (das Programm läuft „länger“).

Messung mit 2000ms:

thread priority: 10

Verbrauchte Zeit s: 2,90

Verbrauchte Zeit s: 2,90

Verbrauchte Zeit s: 2,37

Verbrauchte Zeit s: 2,14

Verbrauchte Zeit s: 2,10

Messung mit 3000ms:

thread priority: 10

Verbrauchte Zeit s: 3,40

Verbrauchte Zeit s: 3,14

Verbrauchte Zeit s: 3,41

Verbrauchte Zeit s: 3,19

Verbrauchte Zeit s: 3,42

**Sourcecode**

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <time.h>

**#include** <pthread.h>

**#include** <errno.h>

**void** **measurement**(**void**\*);

**void** **waste\_msecs**(**unsigned** **int** msecs);

**int** **main**(**int** argc, **char** \*argv[]) {

**int** ret;

**struct** sched\_param param;

pthread\_attr\_t attr;

**pthread\_attr\_init**(&attr);

**pthread\_attr\_setinheritsched**(&attr, PTHREAD\_EXPLICIT\_SCHED);

**pthread\_attr\_getschedparam**(&attr, &param);

param.sched\_priority = 10;

ret = **pthread\_attr\_setschedparam**(&attr, &param);

**if** (ret != EOK)

{

**fprintf**(stderr, "pthread\_attr\_setschedparam: %s\n", strerror(ret));

**return** EXIT\_FAILURE;

}

pthread\_t thread;

ret = **pthread\_create**(&thread, &attr, &measurement, NULL);

**if** (ret != EOK)

{

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

**return** EXIT\_FAILURE;

}

ret = **pthread\_join**(thread, NULL);

**if** (ret != EOK)

{

**fprintf**(stderr, "pthread\_join: %s\n", strerror(ret));

**return** EXIT\_FAILURE;

}

**return** EXIT\_SUCCESS;

}

**void** **measurement**(**void**\* arg)

{

**int** ret;

**struct** timespec startTime;

**struct** timespec endTime;

**struct** sched\_param param;

ret = **pthread\_getschedparam**(**pthread\_self**(), NULL, &param );

**if** (ret != EOK)

{

**fprintf**(stderr, "pthread\_getschedparam: %s\n", strerror(ret));

**exit**(EXIT\_FAILURE);

}

**printf**("thread priority: %d\n", param.sched\_curpriority);

**int** i;

**for** (i = 0; i < 5; i++)

{

ret = **clock\_gettime**(CLOCK\_REALTIME, &startTime);

**if** (ret != EOK)

{

**fprintf**(stderr, "error time: %s\n", strerror(ret));

**exit**(EXIT\_FAILURE);

}

waste\_msecs(3000);

ret = **clock\_gettime**(CLOCK\_REALTIME, &endTime);

**if** (ret != EOK)

{

**fprintf**(stderr, "error time: %s\n", strerror(ret));

**exit**(EXIT\_FAILURE);

}

**unsigned** **long** BILLION = 1000000000L;

**long** seconds = endTime.tv\_sec - startTime.tv\_sec;

**long** nanos = endTime.tv\_nsec - startTime.tv\_nsec;

**if** (nanos < 0)

{

nanos+=BILLION;

seconds--;

}

**printf**("Verbrauchte Zeit s: %lu,%lu\r\n", seconds, nanos);

}

}

**void** **waste\_msecs**(**unsigned** **int** msecs)

{

**unsigned** **int** i = 0;

**unsigned** **int** max = msecs \* 99843;

**int** tmp;

**for** (i = 0; i < max; i++)

{

tmp+=1;

}

}