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

```c
#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;
	}
}
```