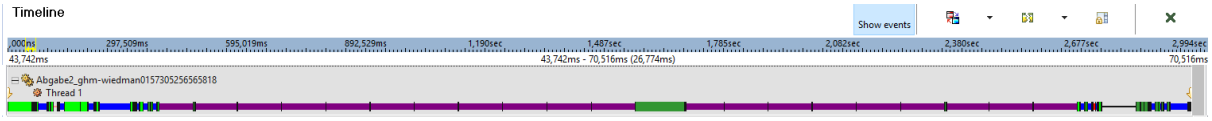
Abgabe 2

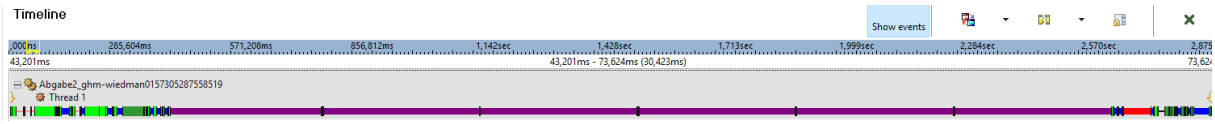
```c
/*
 * This program calculates the time required to
 * execute the program specified as its first argument.
 * The time is printed in seconds, on standard out.
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <sys/neutrino.h>

#define MILLISECONDS_PER_SECOND  1E6
#define TIME_TO_WAIT 1000000 //1ms

struct timespec rqtp, start, stop;

/*
 * Prints current frequency of system tick
 * Changes system tick to nanosecs
 * Prints new frequency of system tick
 * System tick indicates the frequency how oft the cpu is interrupted from the clock
 * Minimal value is 10 microseconds
 */
int changeSystemTick(unsigned int nanosecs) {

  struct _clockperiod new, old;

  if (ClockPeriod(CLOCK_REALTIME, NULL, &old, 0) != 0) {
    perror("clock period");
    return EXIT_FAILURE;
  }

  printf("old fract: %ld, old nsec: %ld \n", old.fract, old.nsec);

  new = old;
  new.nsec = nanosecs;

  if (ClockPeriod(CLOCK_REALTIME, &new, NULL, 0) != 0) {
    perror("clock period");
    return EXIT_FAILURE;
  }

  if (ClockPeriod(CLOCK_REALTIME, NULL, &old, 0) != 0) {
    perror("clock period");
    return EXIT_FAILURE;
  }

  printf("new fract: %ld, new nsec: %ld \n", old.fract, old.nsec);

  return EXIT_SUCCESS;
}

```

```c
/*
 * Simulates a cycle of TIME_TO_WAIT
 * Get current time with clock_gettime
 * For initialization setStart = 1
 * To simulate an ongoing cycle setStart = 0 for every iteration
 * Add TIME_TO_WAIT to the start time
 * With clock_nanosleep the programm sleeps until TIME_TO_WAIT is over
 */
int takt(int setStart) {
  int s;
  int ms;
  int err;

  if (clock_gettime(CLOCK_REALTIME, &start) == -1) {
    perror("clock gettime");
    return EXIT_FAILURE;
  }

  if (setStart == 1) {
    rqtp = start;
  }

  rqtp.tv_nsec += TIME_TO_WAIT;

  err = clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &rqtp, NULL);
  if (err != 0) {
    printf("clock nanosleep: %d \n", strerror(err));
  }

  if (clock_gettime(CLOCK_REALTIME, &stop) == -1) {
    perror("clock gettime");
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}

/*
 * Set system tick
 * Simulate ongoing cycle of 1ms
 * Print the waited miliseconds, check continuity of the results
 */
int main(int argc, char** argv) {
  changeSystemTick(1 * MILLISECONDS_PER_SECOND);
  int i = 0;
  takt(1);

  while(i < 20){
    takt(0);
    i++;
  }
  int s,ms;
  s = (stop.tv_sec - start.tv_sec) * 1000;
  ms = (stop.tv_nsec - start.tv_nsec) / MILLISECONDS_PER_SECOND;

```

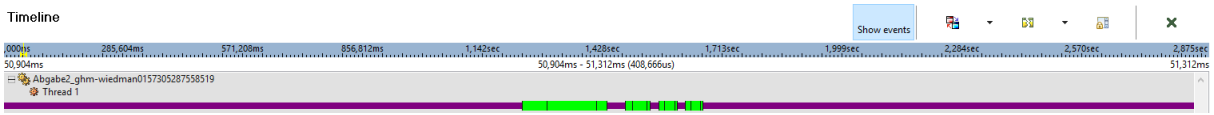
```
108   printf("start: %ld, n %ld \n", start.tv_sec, start.tv_nsec);
109   printf("stop: %ld, n %ld \n", stop.tv_sec, stop.tv_nsec);
110   printf("Waited miliseconds: %d\n", s + ms);
111   return EXIT_SUCCESS;
112 }
```



Bei Überprüfung des Codes mit dem Kernel Event Tracer ist zu sehen, dass bei einer Durchführung mit System Takt mit >=1ms durch den simulierten Takt immer 1ms geschlafen wird.



Wenn der System Takt <1ms wird der simulierte Takt dem System Takt angepasst, da nur alle zb 4ms ein Interrupt durch den Sheduler passiert und das Programm nur bei einem Interrupt aufwachen kann.



Da nur eine ms geschlafen hätte werden sollen, liegt die Aufwachszeit der nächsten x Iterationen in der Vergangenheit. Das heißt das Programm geht nur kurz in den sleep, wo dies dann bemerkt wird und er wieder aufwacht.