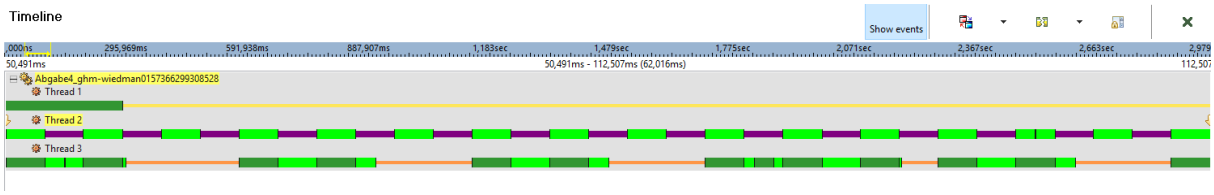
Abgabe 4

Das main-Programm startet zwei Threads. Thread 1 wird alle 4ms aktiv und arbeitet dann für 2ms. Alle drei Takte setzt Thread 1 nach seiner Verarbeitungszeit dann eine Semaphore, auf die Thread 2 wartet. Thread 2 wartet auf eine Semaphore. Sobald diese frei ist, arbeitet er für 3ms.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define BILLION  1000000000L

sem_t semaphore;

void oneSecMethod() {
  volatile int zero = 0, one = 1, sum;
  int i;
  for (i = 0; i < 50000; i++) {
    sum = zero + one;
  }
  for (i = 0; i < 50000; i++) {
    sum = zero + one;
  }
}

void waste_msecs(unsigned int msecs) {
  int i;
  for (i = 0; i < msecs; i++) {
    oneSecMethod();
  }
}
void do_nothing() {
  ;
}

/**
* Zaehlt Zeit um 4ms nach oben.
* Rechnet 2ms.
* Schlaeft bis die anfaenglichen 4ms vorbei sind.
* Alle 3 Durchlaeufe wird eine Semaphore freigegeben.
*/
void* function_thread1(void* arg) {
  struct timespec time;
  int i, err;

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

```

```
49    while (1) {
50
51      time.tv_nsec += 4 * 1000 * 1000;
52
53      if (time.tv_nsec >= BILLION) {
54        time.tv_nsec = time.tv_nsec % BILLION;
55        time.tv_sec += 1;
56      }
57
58      waste_msecs(2);
59
60      err = clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &time, NULL);
61      if (err != 0) {
62        printf("clock nanosleep: %s \n", strerror(err));
63      }
64      if (i == 3) {
65        sem_post(&semaphore);
66        i = 0;
67      }
68      ++i;
69    }
70
71    return NULL;
72  }
73
74  /**
75  * Wartet auf Freigabe einer Semaphore.
76  * Arbeitet dann 3ms.
77  */
78  void* function_thread2(void* arg) {
79    while (1) {
80      while (sem_wait(&semaphore) && (errno == EINTR))
81        ;
82      waste_msecs(3);
83    }
84    return NULL;
85  }
86
87  int main(int argc, char *argv[]) {
88    pthread_t thread_one;
89    pthread_t thread_two;
90    pthread_attr_t attr;
91    pthread_attr_t attr2;
92    int err;
93
94    struct sched_param param;
95
96    param.sched_priority += 12;
97
98    pthread_attr_init(&attr);
99    pthread_attr_init(&attr2);
100
101   pthread_attr_setschedparam(&attr, &param);
102   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
103   pthread_attr_setdetachstate(&attr2, PTHREAD_CREATE_JOINABLE);
```

```
104
105   err = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
106
107   if (err != 0) {
108     printf("pthread_attr_setinheritsched: %s ", strerror(err));
109   }
110
111   sem_init(&semaphore, 0, 1);
112
113   err = pthread_create(&thread_one, &attr, &function_thread1, NULL);
114
115   if (err != 0) {
116     printf("pthread_create: %s ", strerror(err));
117   }
118
119   err = pthread_create(&thread_two, &attr2, &function_thread2, NULL);
120
121   if (err != 0) {
122     printf("pthread_create: %s ", strerror(err));
123   }
124
125   err = pthread_join(thread_one, NULL);
126
127   if (err != 0) {
128     printf("pthread_create: %s ", strerror(err));
129   }
130
131   err = pthread_join(thread_two, NULL);
132
133   if (err != 0) {
134     printf("pthread_create: %s ", strerror(err));
135   }
136
137   return EXIT_SUCCESS;
138 }
```



Thread 1(im Bild Thread2) hat eine höhere Priorität als Thread 2(im Bild Thread3), da sonst Thread 1 seinen 4ms Rhythmus nicht einhalten kann. Thread 1 verdrängt Thread 2 und hält somit seinen Rhythmus ein.