2. Aufgabe: Fester Zeittakt

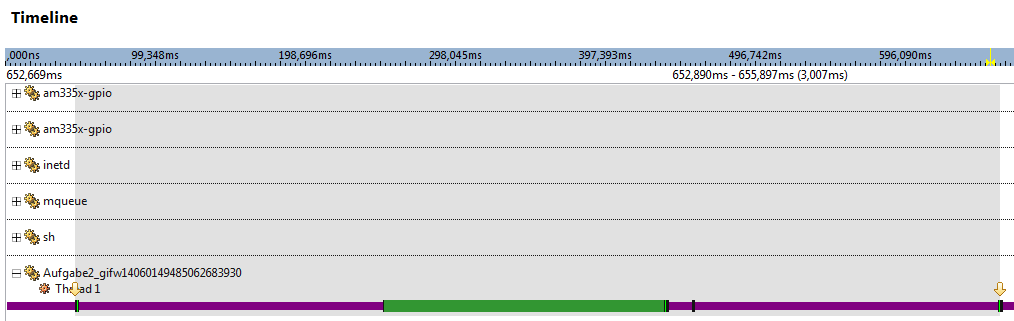
**Aufgabenrealisierung a)**

Der Takt von 1ms wurde mittels einer Schleife realisiert. Zu Beginn haben wir die Zeit gemessen und warten dann in jedem Schleifendurchlauf Anfangszeit + 1 ms – „Schleifenoverhead“ .

**Nachweis zur Aufgabe a)**

Auf zwei Arten lässt sich die korrekte Funktion des Programms nachweisen:

Durch Kernel Logging kann überprüft werden wann der Thread aktiv ist.



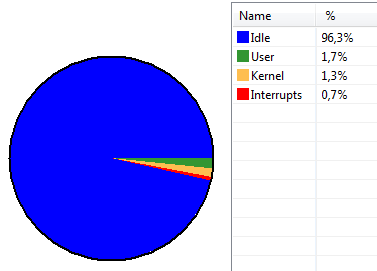
Außerdem vergleichen wir die Schleifendurchläufe mit der Gesamtzeit.

Gemessene Zeit: 4:500000000

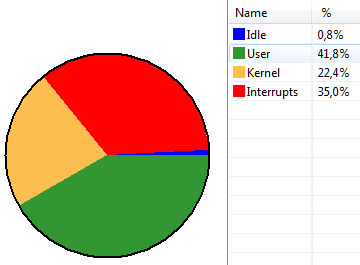
Bei 4500 Durchläufen

**Aufgabe b)**

Auflösung der Systemuhr: 1ms (default)

Der größte Zeit der CPU-Zeit wird im Idle-Zustand verbracht, da das Programm nur kurzzeitig die Schleife durchläuft und dann wieder in den blockiert Zustand übergeht.

Auflösung der Systemuhr: 0,01ms



Anders als in den anderen Messungen verbringt die CPU mehr Zeit Interrupts zu behandeln, da die Zeit zwischen den Interrupts wesentlich geringer ist und so deutlich mehr Interrupts behandelt werden müssen.

Das obere Diagramm spiegelt den Programmablauf besser wieder, da das Programm in einem Schleifendurchlauf nur wenige Instruktionen ausführt und dann wieder in den Idle Zustand übergeht.

Minimaler, gerade noch sinnvoller Wert für die Zykluszeit:

* Anteil an Idle und User Zeit sollte so groß wie möglich sein (Systemverwaltung sollte vergleichsweise niedrig sein)
* Davon sollte der User Anteil so groß wie möglich sein (Rechenleistung des Prozessors sollte so gut wie möglich ausgenutzt sein)

Wie verändert eine Modifikation des Systemtakts die Ergebnisse aus a)?

Bei hohem Systemtakt ändert sich nichts am Programmablauf, allerdings wird viel Zeit für die Systemverwaltung benötigt.

Bei zu niedrigem Systemtakt (Zykluszeit > Timer) können die Timer-Events nicht rechtzeitig behandelt werden, weswegen der korrekte Programmablauf nicht möglich ist.

**Sourcecode**

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <time.h>

**#include** <sys/neutrino.h>

**#include** <errno.h>

**#define** BILLION 1000000000L

**int** **clockMeasure**();

**void** **changeSystemTick**(**unsigned** **int** microsecs);

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

changeSystemTick(10000);

**int** ret = clockMeasure();

**if** (ret == EXIT\_FAILURE) **return** ret;

// Reset auf default Wert.

changeSystemTick(1000);

**return** EXIT\_SUCCESS;

}

**int** **clockMeasure**()

{

**int** ret;

**struct** timespec startTime;

**struct** timespec timeOld;

**struct** timespec timeNew;

**int** duration = 0;

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

**if** (ret != 0)

{

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

**return** EXIT\_FAILURE;

}

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

**if** (ret != 0)

{

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

**return** EXIT\_FAILURE;

}

**while** (duration++ < 4500)

{

timeOld.tv\_nsec+= 1000000;

**if** (timeOld.tv\_nsec >= BILLION)

{

timeOld.tv\_sec++;

timeOld.tv\_nsec=timeOld.tv\_nsec - BILLION;

}

ret = clock\_nanosleep(CLOCK\_REALTIME, TIMER\_ABSTIME, &timeOld, NULL);

**if** (ret != 0)

{

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

**return** EXIT\_FAILURE;

}

}

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

**if** (ret == -1)

{

perror("clock\_gettime");

**return** EXIT\_FAILURE;

}

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

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

**if** (nanos < 0)

{

nanos+=BILLION;

seconds--;

}

printf("%lu:%lu\r\n", seconds, nanos);

}

**void** **changeSystemTick**(**unsigned** **int** microsecs)

{

**struct** timespec res;

**int** ret = clock\_getres(CLOCK\_REALTIME, &res);

**if** (ret == -1)

{

perror("clock\_getres");

exit(EXIT\_FAILURE);

}

printf("Old: %lu\n", res.tv\_nsec);

**struct** \_clockperiod NewClockPeriod;

NewClockPeriod.nsec = microsecs \* 1000;

NewClockPeriod.fract = 0;

ret = ClockPeriod\_r(CLOCK\_REALTIME, &NewClockPeriod, NULL, 0);

**if** (ret != EOK)

{

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

exit(EXIT\_FAILURE);

}

ret = clock\_getres(CLOCK\_REALTIME, &res);

**if** (ret == -1)

{

perror("clock\_getres");

exit(EXIT\_FAILURE);

}

printf("New: %lu\n", res.tv\_nsec);

}