2. Aufgabe: Fester Zeittakt

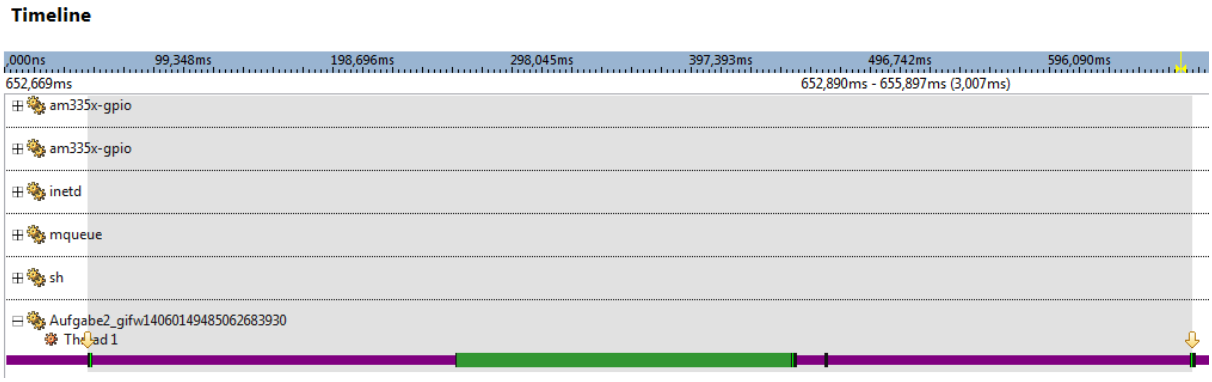
Aufgabenrealisierung a)

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.

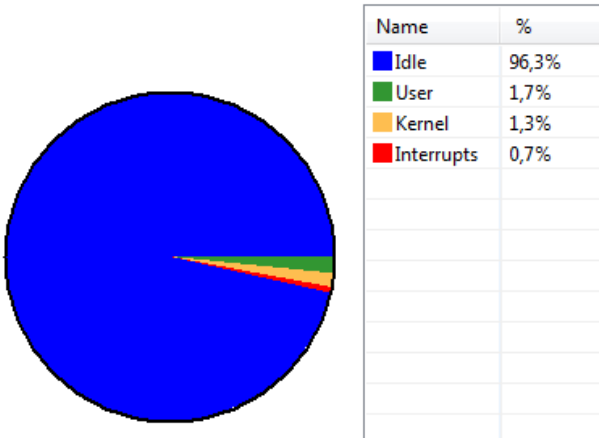


Außerdem vergleichen wir die Schleifendurchläufe mit der Gesamtzeit.
Gemessene Zeit: `4:500000000`
`Bei 4500 Durchläufen`

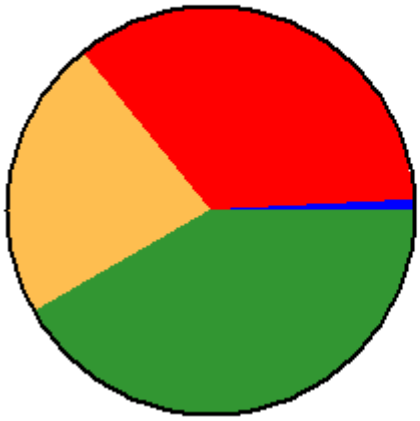
`Aufgabe b)`

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



Name	%
Idle	0,8%
User	41,8%
Kernel	22,4%
Interrupts	35,0%

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

```c
#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);
}
```