FH OÖ - Hagenberg
embedded systems design

RTO1 UE
WS 2020

Protokoll

Übung 3: Entwicklung eines effizienten RTOS

Erweiterung des Echtzeitbetriebssystems APOS

Simon Steindl S2010567025
Florian Hinterleitner S2010567014

Aufgabe A – Erweiterung prioritätsbasiertes Scheduling

Die Priorisierung der Tasks ist implementiert in einer 'prioSufficient()'-Funktion und erfolgt nach dem Schema:

- hat der nächste Task in der Queue ein höhere, oder zumindest gleiche Prio als der aktuelle, erfolgt auch ein Switch.
- hat der nächste Task in der Queue ein kleinere, Prio als der aktuelle, und der aktuelle geht in den 'SUSPENDED' state, erfolgt auch ein Switch.
- Andernfalls wird der aktuelle Task beibehalten.

Untersuchung der Funktion:

- Hat der Counter eine Prio von 1 (sehr unwichtig), laufen alle Tasks korrekt ab, jedoch erfolg das hochzählen des Counters merklich langsamer, als ohne Priorisierung.
- Hat der Counter eine Prio von 100 (gleich wichtig), laufen alle Tasks korrekt ab, das hochzählen des Counters erfolgt wie gewohnt. Da alle Tasks in diesem Fall die selbe Prio haben, ergibt sich das selbe Verhalten wie ein Round Robin ohne Priorisierung.
- Hat der Counter eine Prio von 200 (sehr wichtig), läuft nur noch dieser Task, seine Priorität lässt die anderen 'nicht zum zug kommen'.

Conclusio: Diese sehr primitiver Prio-Konzept erlaubt typischerweise, einzelne unwichtige Tasks nach unten zu priorisieren, einen einzelnen Task als sehr wichtig einzustufen und nach oben zu priorisieren, erfordert ausgeklügeltere Konzepte. Eventuell würden auch Scheduler-calls aus einem Task hoher Priorität heraus das System wieder stabiler zum laufen bringen, jedoch ist man dadurch auf Kooperativität diese Tasks angewiesen.

Abbildung 1: Prio 1

Aufgabe B – Einbau eines Stack-Checkers

Der Stack-Checker wurd realisiert mit 'defines' zum definieren des End-Markers, zum jeweiligen setzen des Markers pro Task, sowie eines Abfrage-Makros für den Marker. Die eigentliche Validiering erfolgt jeweils beim TaskSwitch mit der Funktion 'verifyStackEnd()'. Diese vergleicht den tatsächlichen Marker und retourniert 'TRUE', wenn dieser ident mit dem Sollwert ist, das Stack-Ende also nicht überschrieben wurde.

Der Overhead ergibt sich aus der Differenz der SysTick-werte vor & nach dem stack-checking, sowie dem Prozessortakt:

- 0x0000a15b - 0x0000BB15 = 0x000019BA = 6586(Dez)
- bei 48MHz Prozessortakt ergibt dies 137.2uS für das Stack-Checking.

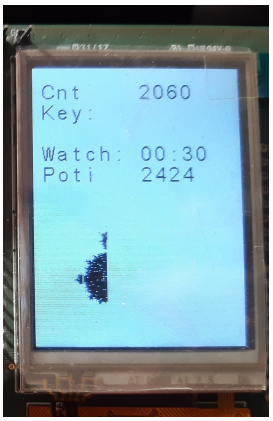

Abbildung 2: Prio 100

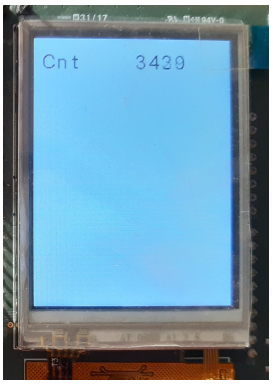

Abbildung 3: Prio 200

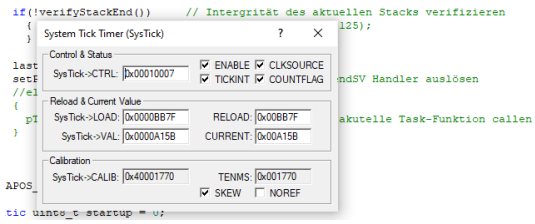

Abbildung 4: Systick vor Stack-Checking

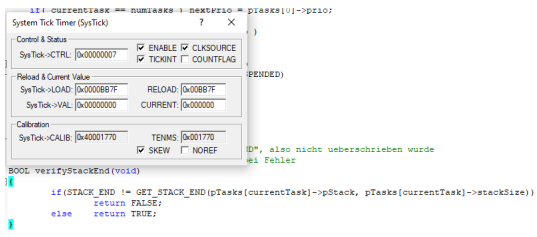

Abbildung 5: Systick nach Stack-Checking