

3. Verarbeitungszeit

a) *waste_msecs*

In dieser Funktion verwenden wir eine for-Schleife, die immer wieder die aktuelle Laufvariable durch dreizehn teilt und damit Zeit verbraucht. Um herauszufinden, wie viele Iterationen es benötigt, um eine Millisekunde abzuwarten, verwenden wir eine *waste_msectest*-Funktion, die eine fixe Anzahl an Iterationen derselben Schleife durchläuft. Mit einem entsprechenden Main-Programm messen wir vor und nach dem Schleifendurchlauf die Zeit und können so die genaue Zeit ermitteln, die *waste_msectest* verbraucht. Durch Ausprobieren konnten wir 37750 Iterationen als den Wert ermitteln, bei dem stabil eine Millisekunde während des Methodenaufrufs verging.

Im folgenden konnten wir also 37750 als Iterationswert für *waste_msecs* verwenden und um diese Methode zu parametrisieren und nicht nur den Durchlauf von einer, sondern auch von mehreren Millisekunden zuzulassen, werden die Iterationen durch Multiplikation der 17750 mit einem Übergabeparameter berechnet. Dieser übergibt die Anzahl der Millisekunden, die gewartet werden sollen.

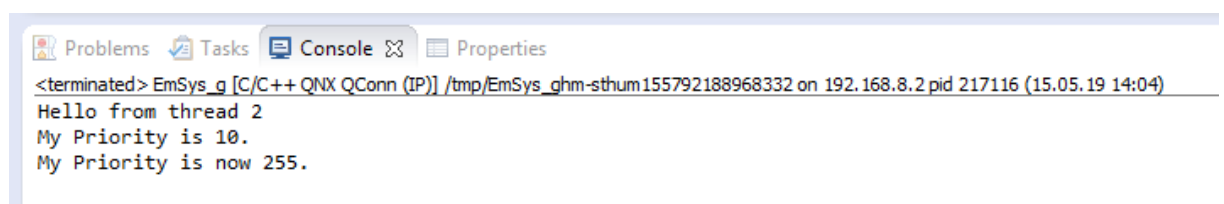
b) *telnet*

Prioritäten von Prozessen in einer der Target-Shell *telnet* lassen sich mit dem Befehl *pidin* ermitteln. *Höhere Zahlen entsprechen höheren Prioritäten. Die höchste beobachtete Priorität betrug 255.*

c) *Prioritäten-Manipulation*

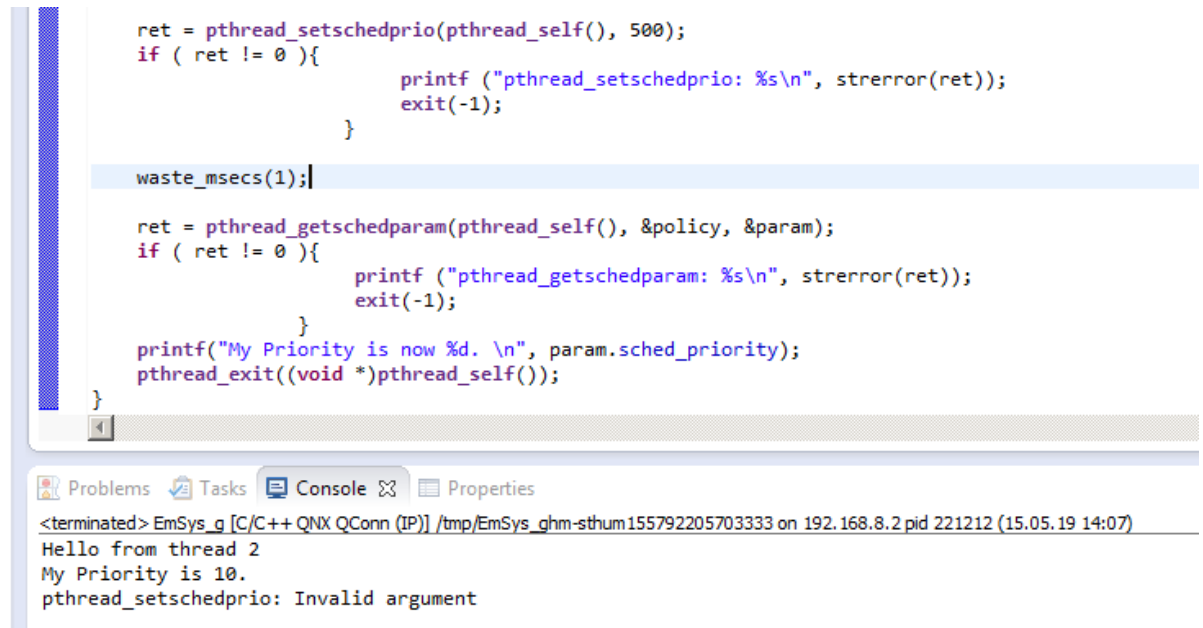
Um die Priorität einer Routine zu messen, benötigen wir einen Thread, in unserem Fall *threadScheduling*. Dieser Thread benötigt ein Struct *param* vom Typ *sched_param* sowie eine *policy*, die das Scheduling-Verfahren angibt. Wir wählen in unserem Fall Round Robin. Anschließend *haben* wir mit Hilfe von *pthread_getschedparam* die Priorität unseres eigenen Threads mit der angegebenen Policy und dem *param*-Struct aufgerufen und in der Variable *sched_priority* gespeichert, die ein Teil des *param*-Structs ist. Anschließend geben wir die aktuelle Priorität auf der Konsole aus, unser Thread hat die Priorität 10. Die höchste Priorität ist 255, also hat der Thread eine recht niedrige Priorität.

Als nächsten Schritt wollen wir die Priorität auf den höchsten Wert, also 255 setzen. Dies erreichen wir mit *pthread_setschedprio* mit der PID unseres laufenden Threads und dem Wert, auf den die Priorität gesetzt werden soll. Anschließend wird die oben konfigurierte *waste_msecs* mit Parameter eins aufgerufen, der Thread arbeitet also eine Millisekunde, anschließend wird erneut mit *pthread_getschedparam* die Priorität abgefragt und ausgegeben. Nun sehen wir, dass die Priorität erfolgreich auf den Wert 255 gesetzt wurde.



```
<terminated> EmSys_g [C/C++ QNX QConn (IP)] /tmp/EmSys_ghm-sthum155792188968332 on 192.168.8.2 pid 217116 (15.05.19 14:04)
Hello from thread 2
My Priority is 10.
My Priority is now 255.
```

255 ist der maximale Wert, der für eine Priorität gesetzt werden kann. Versucht man die Priorität des Threads auf einen höheren Wert zu setzen, zum Beispiel 500, wird ein Fehler ausgegeben, denn der Wert ist keine mögliche Eingabe:



```
ret = pthread_setschedprio(pthread_self(), 500);
if ( ret != 0 ){
    printf ("pthread_setschedprio: %s\n", strerror(ret));
    exit(-1);
}

waste_msecs(1);

ret = pthread_getschedparam(pthread_self(), &policy, &param);
if ( ret != 0 ){
    printf ("pthread_getschedparam: %s\n", strerror(ret));
    exit(-1);
}

printf("My Priority is now %d. \n", param.sched_priority);
pthread_exit((void *)pthread_self());
}
```

The screenshot shows a C++ IDE with a source editor and a console window. The source code attempts to set the thread priority to 500 using `pthread_setschedprio`. The console output shows the thread successfully prints "My Priority is now 10." before the program terminates with the error "pthread_setschedprio: Invalid argument".

Problems Tasks Console Properties

<terminated> EmSys_g [C/C++ QNX QConn (IP)] /tmp/EmSys_ghm-sthum155792205703333 on 192.168.8.2 pid 221212 (15.05.19 14:07)

Hello from thread 2

My Priority is 10.

pthread_setschedprio: Invalid argument