PROGRAMOWANIE SYSTEMÓW CZASU RZECZYWISTEGO

LABORATORIUM

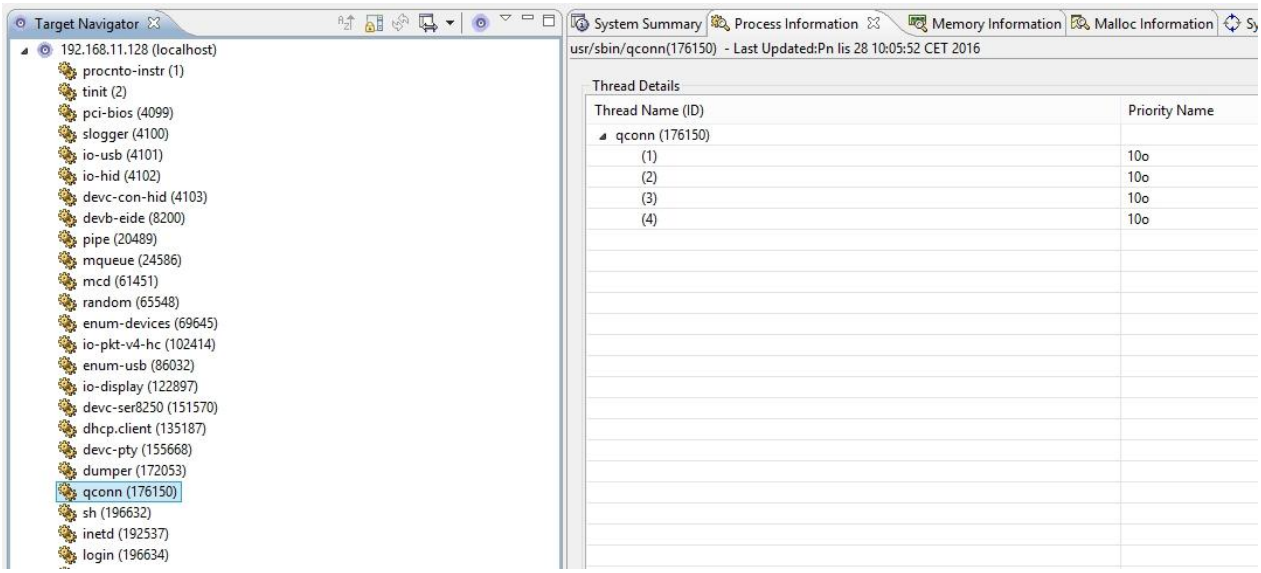
Temat: THREADS

Mariusz Rudnicki

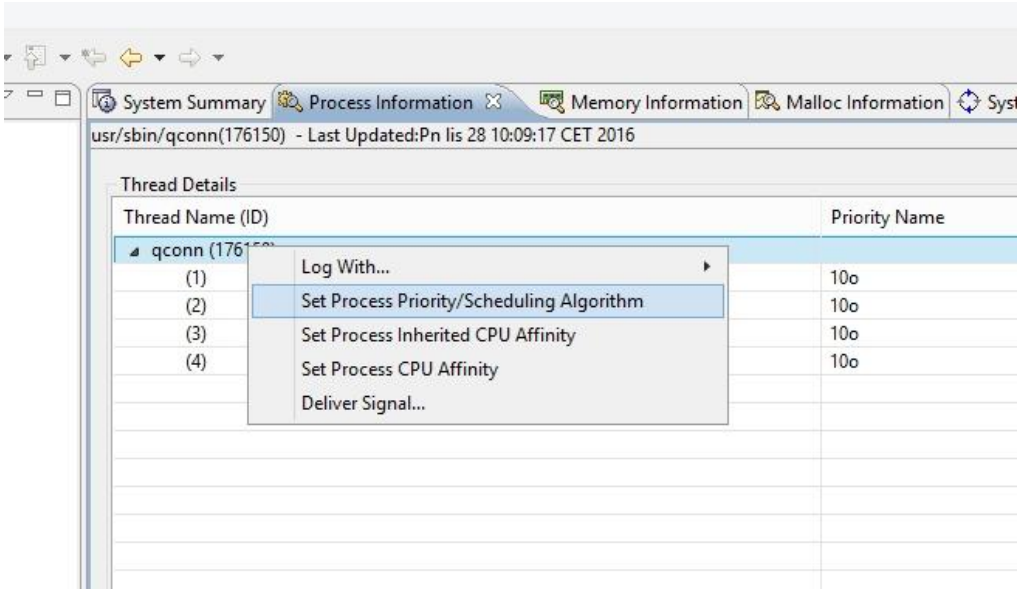
2016

1. Przygotowanie platformy i środowiska IDE.

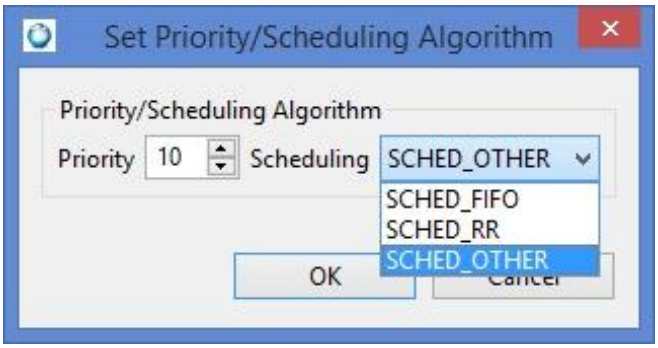
Przed uruchomieniem własnego kodu zwiększ priorytet procesu ***qconn*** z poziomu środowiska IDE. W tym celu wybierz perspektywę ***QNX System Information***. Wybierz zakładkę ***Process Information***. Następnie z listy procesów uruchomionych na platformie docelowej wybierz proces ***qconn***.



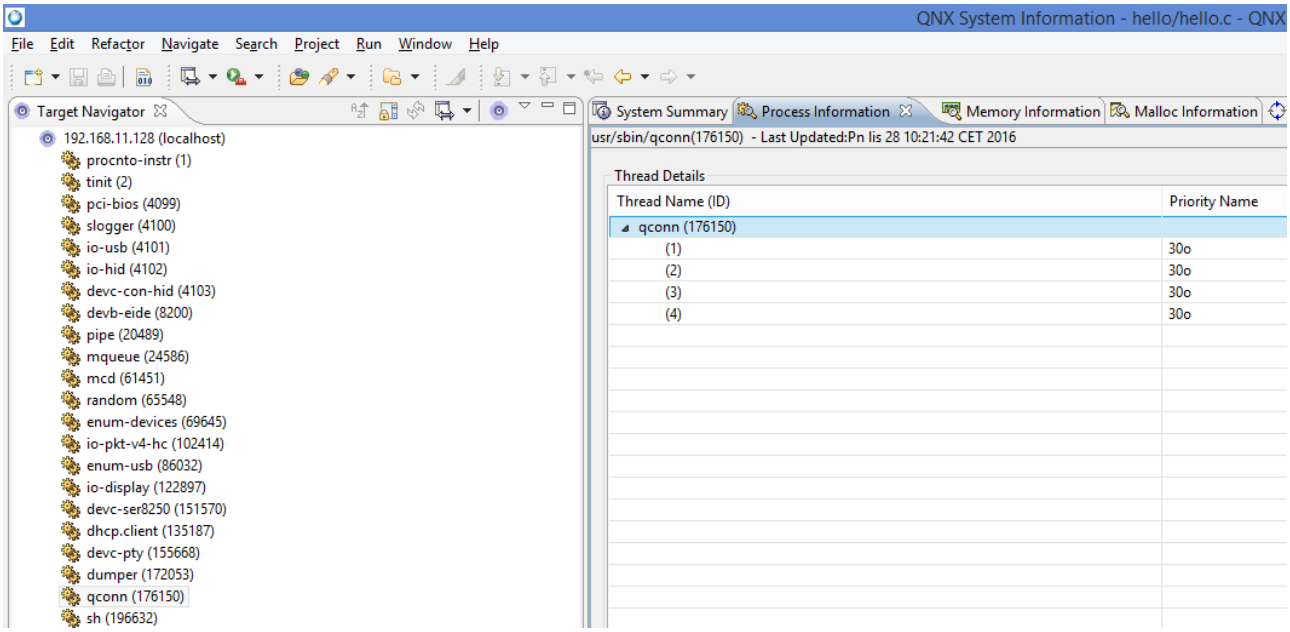
W zakładce ***Process Information*** wciśnij prawy klawisz myszy na nazwie procesu głównego ***qconn***. Pojawi się menu kontekstowe – rysunek poniżej.



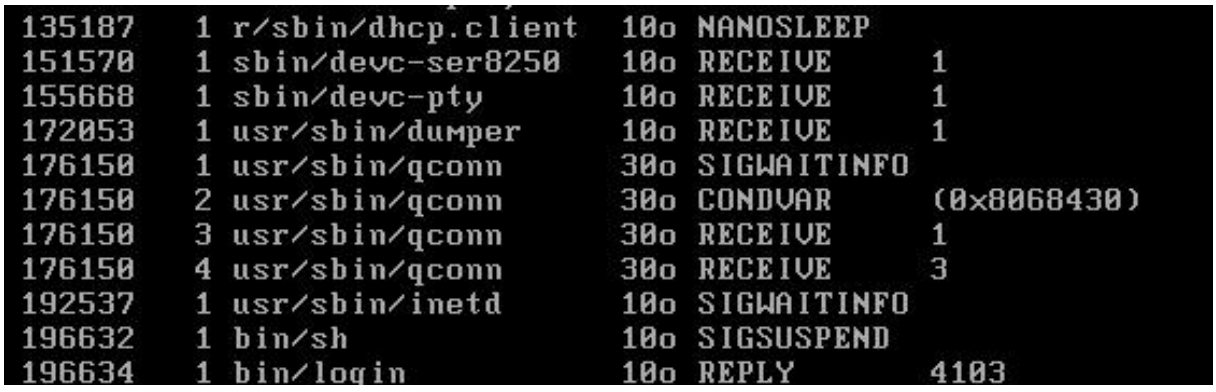
Wybierz opcję ***Set Process Priority/Scheduling Algorithm*** na ekranie pojawi okno.



W polu ***Priority*** wpisz wartość 30. Algorytm szeregowania pozostaw bez zmian. Wciśnij klawisz OK zatwierdzając zmiany. Wynik zmian powinien być widoczny w zakładce ***Process Information***, jak na poniższym rysunku.



Zmianę ustawień możesz sprawdzić również w wirtualnej maszynie używając polecenia ***pidin***.



2. Przeanalizuj poniższy kod. Zwróć uwagę na wynik działania funkcji update_thread().

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <sched.h>

/*
The number of threads that we want to have running simultaneously.
*/

#define NumThreads      1

volatile int     var1;
volatile int     var2;
/*
The header of the main function of thread. Thread starts its job in this function.
*/
void    *update_thread (void *);

char    *progname = "threads";

int main(int argc, char *argv[]) {
    pthread_t           threadID [NumThreads];  // a place to
                                                // hold the thread
                                                 // ID's
    pthread_attr_t      attrib;                  // scheduling
                                                 // attributes
    struct sched_param  param;                   // for setting
                                                 // priority
    int                 i, policy;
    int                 oldcancel;

    setvbuf (stdout, NULL, _IOLBF, 0);
    var1 = var2 = 0;        /* initialize to known values */
    printf ("%s:  starting; creating threads\n", progname);
/*
We want to create the new threads using Round Robin scheduling, and a lowered priority, so set up a thread
```

```
attributes structure. We use a lower priority since
these threads will be hogging the CPU
*/
    pthread_getschedparam( pthread_self(), &policy, &param );
    pthread_attr_init(&attrib);
    pthread_attr_setinheritsched(&attrib, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy (&attrib, SCHED_RR );
    param.sched_priority -= 5;      // change priority a couple
                                    // levels
    pthread_attr_setschedparam (&attrib, &param);
    pthread_setcanceltype( PTHREAD_CANCEL_ASYNCHRONOUS, &oldcancel);
/*
Create the threads. As soon as each pthread_create call
is done, the thread has been started.
*/

    for (i = 0; i < NumThreads; i++) {
        pthread_create( &threadID [i], &attrib, &update_thread,
                        (void *) i);
    }
/* Let the other threads run for a while */
    sleep (120);
/* and then kill them all*/

    printf ("%s:  stopping; cancelling threads\n", progname);

    for (i = 0; i < NumThreads; i++) {
        pthread_cancel (threadID [i]);
    }

    printf ("%s:  all done, var1 is %d, var2 is %d\n", progname,
            var1, var2);
    fflush (stdout);
    sleep (1);
    return EXIT_SUCCESS;
}

void *update_thread (void *i)
{
    while (1) {
      if(var1 != var2) {
        printf ("thread %d, var1 (%d) != var2 (%d)!\n",(int) i,
                var1, var2);
        var1 = var2;
      }
```

```
        var1++;
        //sched_yield(); /* for faster processors, to cause problem
          to happen */
        var2++;
    }
    return (NULL);
}
```

3. Jaki jest wynik działania powyższego programu, czy wartości zmiennych var1 i var2 będą sobie równe, czy też różne?
4. Uzupełnij kod projektu **Threads**, tak aby uruchomiony został jeden wątek o priorytecie o pięć niższym niż priorytet procesu głównego, algorytm szeregowania Round Robin. W helpie sprawdź funkcje, których wywołania należy uzupełnić.
5. Sprawdź wynik działania uruchomionego kodu. Czy wynik jest zgodny z przewidywaniami z pkt 3. Do sprawdzeń użyj **perspektywy QNX System Information.** Wynik działania programu przedstaw prowadzącemu.
6. Zmodyfikuj kod programu tak, aby proces główny tworzył 16 wątków, które będą rozpoczynały pracę w funkcji **update_thread().** Jaki będzie wynik działania programu, czy wartości zmiennych var1 i var2 będą sobie równe, czy też różne?
7. Sprawdź wynik działania uruchomionego kodu. Do sprawdzeń użyj **perspektywy QNX System Information.** Wynik działania programu przedstaw prowadzącemu.
8. Jakie może być rozwiązanie ewentualnych problemów?

9. Zmodyfikuj kod w taki sposób, aby proces główny tworzył trzy wątki, które będą rozpoczynały pracę w różnych funkcjach, będą miały różny priorytet oraz różny sposób szeregowania.
10. Sprawdź wynik działania uruchomionego kodu. Do sprawdzeń użyj **perspektywy QNX System Information.** Wynik działania programu przedstaw prowadzącemu.

11. Sprawozdanie.

11.1. Przewidywany wynik działania programu z pkt 2.

Zależność pomiędzy zmiennymi:

a) var1 == var2;
b) var1 <> var2;
c) var1 > var2;
d) var1 < var2.

11.2. Sprawdzenie działania kodu pkt 5.

Zależność pomiędzy zmiennymi:

a) var1 == var2;
b) var1 <> var2;
c) var1 > var2;
d) var1 < var2.

11.3. Przewidywany wynik działania kodu pkt 6.

Zależność pomiędzy zmiennymi:

a) var1 == var2;
b) var1 <> var2;
c) var1 > var2;
d) var1 < var2.

11.4. Sprawdzenie działania kodu pkt 7.

Zależność pomiędzy zmiennymi:

a) var1 == var2;
e) var1 <> var2;
f) var1 > var2;
g) var1 < var2.

11.5. Rozwiązanie ewentualnych problemów:

… … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … …

Grupa: