REAL TIME SYSTEMS

Exercise 5: POSIX THREADS

TASKS

M.Sc. Michael Rethfeldt

1 Thread creation

a. Write a program that creates one child thread which prints "Hello World!". To create a POSIX thread the function *pthread_create()* is used. Get to know how POSIX libraries (here: the *pthread* library) can be linked to the program!

Hint: The function *pthread_create()* expects as parameter a function pointer of type *void* (*start_routine)(void *)*. If the function passed does not comply with the expected type, the compiler fails with an error.

Expected thread function format:

```
int main(int argc, char *argv[])
{
    …
    pthread_create(&thread, &attr,
                   thread_start, 0);
    …
}

void* thread_start(void *ptr)
{
    return 0;
}
```

2 Creating multiple threads

a. Write a *"Hello World!"* program that creates two threads. The first thread shall print *"Hello"* and the second thread shall print *"World!"*. Run the program multiple times! What do you observe? Explain the term 'zombie thread'!

   Experiment with different output variants, e.g.
   - Calling Thread: "I have just created two threads!"
   - Thread 1: "I am thread 1 and have to print 'Hello'"
   - Thread 2: "I am thread 2 and have to print 'World!'"
   - Repeat the printing within the son threads by using loops!

b. Try to eliminate the race conditions that you experienced. What could be suitable ways to achieve an ordered output? Can you think of easy solutions without changing scheduling parameters?

c. As a next step, think of an adequate scheduling policy and choice of thread priorities! How can these parameters be configured in Linux? What scheduling policies and priorities are available and how can they be applied to processes and threads? Does this require certain user privileges? Use the online documentation for help (see PDF file "*Linux Scheduling Priorities and Policies*").

3 Recursion

a. Explain the program in Fig. 1. Check its behavior and, if necessary, perform modifications and extensions.

b. Test your solution on a Linux machine. What do you observe?

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

////////////////////////////////////////////////////////////////////////////////
void*   threadcode(void *ptr);
int     rec(int number);
#define COUNT   300000

////////////////////////////////////////////////////////////////////////////////
int main(void)
{   int                number;
    pthread_attr_t  attr;
    pthread_t       thread1;

    pthread_attr_init(&attr);
    pthread_create(&thread1, &attr, threadcode, 0);
    pthread_join(thread1, 0);
    return 0;
}

////////////////////////////////////////////////////////////////////////////////
void* threadcode(void *ptr)
{   printf("\nResult is: %d.\n", rec(0));
    return 0;
}

////////////////////////////////////////////////////////////////////////////////
int rec(int number)
{   if (!(number%200))
        printf("\n%6d:", number);
    printf("%d", number%10);
    if (number<COUNT)
        return number+rec(++number);
    else
        return 0;
}
```

Fig. 1: aufgabe3.c