Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CO20-320202
Date: 2017-09-27
Due: 2017-10-04

OS Problem Sheet #2

Problem 2.1: *pthread programming* (1+1+1+1 = 4 points)

The following pthread source code has been written by a programmer who is less experienced than you. Help him to find and fix the mistakes he has made. Note that the program is syntactically correct, it compiles and links without any errors. Error handling code has been omitted for brevity, i.e., lack of runtime error checking is not the problem this programmer is facing. For each bug you have found, explain what is wrong with the code and indicate how the problem can be fixed.

```
/*
 * p2-pthread-oops.c --
 */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

static int N = 5;

static void* run(void *arg)
{
    int *i = (int *) arg;
    char buf[123];

    snprintf(buf, sizeof(buf), "thread %d", *i);
    return buf;
}

int main(int argc, char *argv[])
{
    int i;
    pthread_t *pt = NULL;

    for (i = 0; i < N; i++) {
        pthread_create(pt, NULL, run, &i);
    }

    return EXIT_SUCCESS;
}
```

Problem 2.2: *multi-threaded coin flipping* (6 points)

There are 20 coins lying in a row on a table. P persons flip all 20 coins lying on the table N times. Write a program that emulates this by using threads, one thread emulating one person. By default, there are $P = 100$ persons and each person flips all 20 coins $N = 10000$ times. Implement command line options to control the number of persons and the number of coin flips per person.

Use a global array of characters to represent the coins, where an `X` represents one side of a coin and a `0` represents the other side of a coin. Print the coins on the standard output before the coin flipping starts and print the coins when all persons have finished flipping coins. In addition, print the time it took for all persons to flip the coins.

You must ensure that no coin is flipped by another person, while one person has his turn. Implement three strategies:

1. In the first strategy, you use a global lock to protect the coins. A person first obtains the global lock covering all coins and then the person flips N times all 20 coins and finally, when done flipping coins, the person releases the lock.
2. In the second strategy, you also use a global lock but a person obtains the lock for each iteration, i.e., a person obtains the lock, flips the 20 coins, releases the lock, and then moves to the next iteration.
3. In the third strategy, there is a lock for each coin, i.e., a person obtains a lock for a coin, flips the coin, and releases the lock immediately after each coin flip.

Let your program measure the time it took all persons to flip all coins and write the results to the standard output. An example execution might look as follows:

```
$ ./coins
coins: 0000000000XXXXXXXXXX (start - global lock)
coins: 0000000000XXXXXXXXXX (end - global lock)
100 threads x 10000 flips:    120.018 ms

coins: 0000000000XXXXXXXXXX (start - iteration lock)
coins: 0000000000XXXXXXXXXX (end - table lock)
100 threads x 10000 flips:   1654.724 ms

coins: 0000000000XXXXXXXXXX (start - coin lock)
coins: 0000000000XXXXXXXXXX (end - coin lock)
100 threads x 10000 flips:   88125.423 ms
```

Time measurement can be done in a simple (not very accurate) manner. The `timeit()` function shown below takes as arguments the number of threads and a pointer to a function implementing one of the three coin flipping strategies. The `run_threads()` function creates `n` threads (each one executing `proc`) and joins them again. The calls to `clock()` obtain the a timestamp before and a timestamp after the execution of `run_threads()`. The timestamps are then used calculate the execution time (in microseconds).

```
static double
timeit(int n, void* (*proc)(void *))
{
    clock_t t1, t2;
    t1 = clock();
    run_threads(n, proc);
    t2 = clock();
    return ((double) t2 - (double) t1) / CLOCKS_PER_SEC * 1000;
}
```

Marking:

- *1pt thread creation / joining*
- *1pt option parsing and time measurement*
- *1pt coin flipping loop global lock*
- *1pt coin flipping loop iteration lock*
- *1pt coin flipping loop coin lock*
- *1pt runtime error handling*