Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2022-10-04

OS 2022 Class Problem Sheet #4

Problem 4.1: *h2o problem* (= 0 points)

Allen B. Downey describes the "Building H_2O" problem in his book "The Little Book of Semaphores". He states the synchronization problem as follows:

> There are two kinds of threads, oxygen and hydrogen. In order to assemble these threads into water molecules, we have to create a barrier that makes each thread wait until a complete molecule is ready to proceed. As each thread passes the barrier, it should invoke bond. You must guarantee that all the threads from one molecule invoke bond before any of the threads from the next molecule do.
>
> In other words:
>
> - If an oxygen thread arrives at the barrier when no hydrogen threads are present, it has to wait for two hydrogen threads.
> - If a hydrogen thread arrives at the barrier when no other threads are present, it has to wait for an oxygen thread and another hydrogen thread.
>
> We don't have to worry about matching the threads up explicitly; that is, the threads do not necessarily know which other threads they are paired up with. The key is just that threads pass the barrier in complete sets; thus, if we examine the sequence of threads that invoke bond and divide them into groups of three, each group should contain one oxygen and two hydrogen threads.
>
> Puzzle: Write synchronization code for oxygen and hydrogen molecules that enforces these constraints.

Implement a solution for this problem using POSIX threads. Make sure your program runs quietly under the Helgrind race condition checker:

```
$ valgrind --tool=helgrind ./h2o
```

Here is a possible starting point.

```
/*
 * h2o/h2o.c --
 *
 *      A simple program to create water molecules by bonding two
 *      hydrogen and one oxygen atom.
 */

#define _POSIX_C_SOURCE 200809L

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

typedef struct h2o {
    unsigned int hydrogen;
    unsigned int oxygen;
    unsigned int bonds;
```

```
    /* add whatever you may need here */
} h2o_t;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-function"

/*
 * Wrapper functions that catch and report errors.
 */

static void mutex_lock(pthread_mutex_t *mutex)
{
    int rc = pthread_mutex_lock(mutex);
    if (rc) {
        fprintf(stderr, "pthread_mutex_lock(): %s\n", strerror(rc));
        exit(EXIT_FAILURE);
    }
}

static void mutex_unlock(pthread_mutex_t *mutex)
{
    int rc = pthread_mutex_unlock(mutex);
    if (rc) {
        fprintf(stderr, "pthread_mutex_unlock(): %s\n", strerror(rc));
        exit(EXIT_FAILURE);
    }
}

static void cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)
{
    int rc = pthread_cond_wait(cond, mutex);
    if (rc) {
        fprintf(stderr, "pthread_cond_wait(): %s\n", strerror(rc));
        exit(EXIT_FAILURE);
    }
}

static void cond_broadcast(pthread_cond_t *cond)
{
    int rc = pthread_cond_broadcast(cond);
    if (rc) {
        fprintf(stderr, "pthread_cond_broadcast(): %s\n", strerror(rc));
        exit(EXIT_FAILURE);
    }
}
#pragma GCC diagnostic pop

/*
 * The oxygen thread function.
 */

static void* oxygen(void *data)
{
    h2o_t *h2o = (h2o_t *) data;

    (void) h2o;         /* do something more useful here */

    return NULL;
}

/*
 * The hydrogen threat function.
 */
```

```c
static void* hydrogen(void *data)
{
    h2o_t *h2o = (h2o_t *) data;

    (void) h2o;         /* do something more useful here */

    return NULL;
}

int main(int argc, char *argv[])
{
    int rc;
    unsigned int n = argc - 1;
    unsigned int m = 3 * n;
    pthread_t tids[m];
    h2o_t h2o = {
        .hydrogen = 0,
        .oxygen = 0,
        .bonds = 0,
    };

    (void) argv;
    for (unsigned int i = 0; i < m; i++) {
        rc = pthread_create(&tids[i], NULL,
                            (i % 3 == 0) ? oxygen : hydrogen, &h2o);
        if (rc) {
            fprintf(stderr, "pthread_create(): %s\n", strerror(rc));
        }
    }

    for (unsigned int i = 0; i < m; i++) {
        if (tids[i]) {
            rc = pthread_join(tids[i], NULL);
            if (rc) {
                fprintf(stderr, "pthread_join(): %s\n", strerror(rc));
            }
        }
    }

    if (h2o.bonds != n) {
        fprintf(stderr, "got %d bonds, expected %d bonds\n", h2o.bonds, n);
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```