Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2022-09-27

OS 2022 Class Problem Sheet #3

Problem 3.1: *unisex bathroom problem* (0 points)

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

> I wrote this problem1 when a friend of mine left her position teaching physics at Colby College and took a job at Xerox.
>
> She was working in a cubicle in the basement of a concrete monolith, and the nearest women's bathroom was two floors up. She proposed to the Uberboss that they convert the men's bathroom on her floor to a unisex bathroom, sort of like on Ally McBeal.
>
> The Uberboss agreed, provided that the following synchronization con- straints can be maintained:
>
> - There cannot be men and women in the bathroom at the same time.
> - There should never be more than three employees squandering company time in the bathroom.
>
> Of course the solution should avoid deadlock. For now, though, don't worry about starvation. You may assume that the bathroom is equipped with all the semaphores you need.

a) Develop a solution in pseudocode using semaphores.

b) 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.

```c
/*
 * 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"
```

```c
/*
 * 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.
 */

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;
}
```