Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: 320202
Date: 2012-02-29
Time: 10 min.

Quiz Sheet #2

Problem 2.1: *posix threads and semaphores* (2+2+2 = 6 points)

Consider the following C program. Assume all system calls and library function calls succeed (error handling code has been omitted for brevity).

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

pthread_t t1, t2;

sem_t sem1, sem2;

void* thread1(void* arg) {
    sem_wait(&sem1);
    printf("In thread 1\n");
    sem_wait(&sem2);
    return NULL;
}

void* thread2(void* arg) {
    sem_post(&sem2);
    printf("In thread 2\n");
    sem_post(&sem1);
    printf("Bonus\n");
    return NULL;
}

int main() {
    sem_init(&sem1, 0, 0);
    sem_init(&sem2, 0, 0);

    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Done\n");
    return 0;
}
```

a) Will the program finish execution? If yes, explain why it finishes. If not, provide an execution path under which it may deadlock.

b) Is the program deterministic? Explain!

c) What does the program print to the standard output (if it finishes execution)?

Solution:

a) The program always will finish execution. The semaphores are initialized to 0 and the second thread is incrementing the semaphores and never blocking. The first thread potentially blocks on the semaphores but since the second thread posts the semaphores, the first thread will eventually acquire the semaphores and finish.

b) The program is not deterministic. One possible execution is that the second thread completes before the first thread even starts running. Another possible execution is that the second thread runs until the second post and then the first thread runs to completion before the second thread continues.

c) The output produced by the program is one of the following:

```
In thread 2         In thread 2
Bonus               In thread 1
In thread 1         Bonus
Done                Done
```

Problem 2.2: *posix threads and mutex locks* (2+2 = 4 points)

Consider the following C program. Assume all system calls and library function calls succeed (error handling code has been omitted for brevity).

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

pthread_t t1, t2;

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;

void* thread1(void* arg) {
    pthread_mutex_lock(&mutex1);
    printf("T1 <- M1\n");
    pthread_mutex_lock(&mutex2);
    printf("T1 <- M2\n");
    pthread_mutex_unlock(&mutex2);
    pthread_mutex_unlock(&mutex1);
    return NULL;
}

void* thread2(void* arg) {
    pthread_mutex_lock(&mutex2);
    printf("T2 <- M2\n");
    pthread_mutex_lock(&mutex1);
    printf("T2 <- M1\n");
    pthread_mutex_unlock(&mutex1);
    pthread_mutex_unlock(&mutex2);
    return NULL;
}

int main() {
    pthread_create(&t1, NULL, thread1, NULL);
    pthread_create(&t2, NULL, thread2, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Done\n");
    return 0;
}
```

a) Will the respective program finish execution ? If not, provide an execution path under which it may deadlock.

b) If the program terminates, what is the output produced? Explain!

Solution:

a) The program deadlocks if the first thread acquires `mutex1` and the second thread acquires `mutex2` and both threads then start blocking on the second call of `pthread_mutex_lock`.

b) If the program terminates, the output is one of the following:

```
T2 <- M2            T2 <- M2
T2 <- M1            T2 <- M1
T1 <- M1            T1 <- M1
T1 <- M2            T1 <- M2
Done                Done
```