Operating Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2023-10-06
Due: 2023-10-13

Problem Sheet #5

Problem 5.1: *taxi stand offering shared rides* (2+8 = 10 points)

Write a program to simulate a taxi stand offering shared rides. Your program models persons that are either taxi drivers or travelers. Both kinds of persons regulary show up at a taxi stand to provide transport service or to use transport service. A taxi stand operates according to the following rules:

1. If a taxi driver arrives at the taxi stand and there are no waiting travelers, then the taxi driver sits down and waits.
2. If a taxi driver arrives and there are waiting travelers, then the taxi driver takes all waiting travelers into his car and drives them to their destinations (we assume that the car has sufficient capacity).
3. If a traveler arrives at the taxi stand and there are no waiting taxi drivers, then the traveler sits down and waits.
4. If a traveler arrives and there are waiting taxi drivers, then the travel picks the taxi driver waiting for the longest time and they leave.

a) Solve this synchronization problem using (counting) semaphors. Both, travelers and drivers, are implemented as separate threads that execute a mainloop that looks like this (pseudocode):

```
person_life(person)
{
    while (1) {
        switch (person->type) {
          case traveler:
            stand_visit_traveler(stand, person);
            break;
          case driver:
            stand_visit_driver(stand, person);
            break;
        }
        sleep(random());
    }
}
```

Write a solution of the synchronization problem in pseudocode.

b) Implement the solution in C using POSIX mutexes and condition variables and no other synchronization primitives (no semaphores, barriers, message queues, ...).

Your program must support the command line option `-t` to define the number of travelers (with a default value of 1) and the command like option `-d` to define the number of drivers (with a default value of 1).

Your program should produce a trace of what is happening at the taxi stand. The trace consists of trace lines where each trace line descibes a change at the taxi stand. The first column shows the name of the program, the second column shows the list of persons waiting at the taxi stand, and a third column describes the event, which caused the trace line to be generated.

The trace below shows how traveler t0 enters the empty stand and decides to wait. Subsequently, driver d1 arrives at the taxi stand, picks up traveler t0, and leaves the taxi stand. The traveler t0 then wakes up and leaves the taxi stand as well.

```
$ ./taxi -c 1 -d 1
taxi:   []                              t0 entering
```

```
taxi:   [t0]                              t0 waiting...
taxi:   [t0]                              d1 entering
taxi:   [d1,t0]                           d1 picking traveler t0
taxi:   [d1,t0]                           d1 leaving
taxi:   [t0]                              ...t0 waking up
taxi:   [t0]                              t0 leaving
```