The problem of the sleepy hairdresser

(Originally *Sleeping barber problem*)

Task

A hairdresser works in a hair salon. The salon has a separate waiting room with *N* places where clients are waiting. The hairdresser has only one chair in which she can receive a client and work on his hair.

The hairdresser opens the door of the salon in the morning and performs the following cyclical activity until the end of her working hours: she sits in a chair and sleeps until she is woken up by the arrival of a client or her clock rings when her working hours are over. If a client has come, he takes him and works on his hair. Then, if there are more clients, he takes them one by one. If the waiting room is empty, he sits down in the chair again and sleeps. When there are clients in the waiting room, the hairdresser must not sleep.

The new client first enters the waiting room. If there are no empty seats, i.e. he is already waiting in the waiting room *N* clients, the client leaves without the work done. If there are empty chairs, she sits down and signals her arrival to the hairdressers. When the hairdresser invites him, he comes to her for a haircut. Then he leaves the salon.

Simulate the displayed system with a device/process *hairdresser* and actions/processes *client* which creates an auxiliary (maybe initial or main) process/process, which also signals the end of working hours, after which new clients are no longer accepted.

Proposal of a possible structure of the solution (the solution does not have to look like this)

```
dretva hairdresser ()
    open the salon and place the "OPEN" sign,
    repeat
        if it is the end of working hours then
            put a "CLOSED" sign

        if there are clients in the waiting room then
            take the first client from the waiting room //semaphore is
            working on a haircut //sleep
        otherwise if it is not the end of working hours
            sleep //semaphore
        otherwise
            // end of working hours and the salon is empty
            close salon
            finish with work

client company (x)
    write that you appeared (x)
    if the "OPEN" sign is posted and there is room in the waiting room then {
        update the number of seats in the waiting
        room signal the hairdresser //semaphore
```

```
        wait for your turn //semaphore
        // the hairdresser called you, go get a haircut, but first update the number
places  in waiting rooms
        // - it is not necessary to additionally synchronize waiting for the hairstyle to be
done
        // - it is enough to print the state and finish with the client tree
    }
    otherwise {
        today nothing from hair //print + finish work
    }
```

Notes

Using shared variables (eg *Open*, *br_place*) protect with a binary semaphore!

Let the initial entity/process create all other entities/processes first *hairdresser* and then actions/processes *clients* and marks the end of working hours. You determine the duration of work on the hairstyle and the schedule of client arrivals.

An example of a printout (five clients were created immediately, and a few more later):

```
Hairdresser: I'm opening the salon Hairdresser: I'm
putting up an OPEN sign Hairdresser: I'm sleeping
until the clients come
        Client(1): I want a haircut Client(1): I'm
        entering the waiting room (1) Hairdresser: I'm
going to work on client 1
        Client(1): the hairdresser is doing my hair
        Client(4): I want a haircut
        Client(4): Entering the waiting room (1)
        Client(3): I want a haircut Client(3): Entering
        the waiting room (2) Client(2): I want a haircut
        Client(2): Entering the waiting room (3) Client
        (5): I want a haircut

        Client(5): There's no room in the waiting room, I'll be back tomorrow
Hairdresser: Client 1 is done
Hairdresser: I'm going to work on client 4
        Client(4): the hairdresser is doing my hair.
Hairdresser: Client 4 is done
Hairdresser: I'm going to work on client 3
        Client(3): the hairdresser is doing my hair.
Hairdresser: Client 3 is done
Hairdresser: I'm going to work on client 2
        Client(2): hairdresser is doing my hair
Hairdresser: Client 2 is done
Hairdresser: I sleep until the clients come
        Client(6): I want a haircut Client(6): I'm
        entering the waiting room (1) Hairdresser: I'm
going to work on client 6
        Client(6): the hairdresser is doing my hair.
Hairdresser: Client 6 is done
Hairdresser: I sleep until the clients come
```

Client(7): I want a haircut Client(7): I'm
entering the waiting room (1) Hairdresser: I'm
going to work on client 7
Client(7): the hairdresser is doing my hair.
Hairdresser: Client 7 is done
Hairdresser: I'm sleeping until clients come
Hairdresser: I'm putting up a CLOSED sign
Hairdresser: I'm closing the salon
- - - - - -

Cannibals and missionaries

Task

Solve the problem of transporting cannibals and missionaries. On the bank of a wide river there is a boat that transports cannibals and missionaries to the other side of the bank. The capacity of the boat is 7 passengers. There must be at least 3 passengers in the boat for it to start. There may not be more cannibals than missionaries in the boat, while all other combinations of passengers are allowed (eg only cannibals can be in the boat). Cannibals and missionaries come from both sides of the river. One missionary comes every two seconds, and a cannibal every second (coast selection is random). After crossing the river, the passengers go further (they are no longer in the system). Let there be only one boat in the system, and each missionary and cannibal represent one thread/ process. The boat is also a dretva/process that writes out who it is transporting at each transition (eg "Transported: missionary, cannibal, missionary, missionary"). Assume that the boat is initially on the right bank. After three (or more) places in the boat are filled, the boat waits for another second, during which someone else can get into the boat, according to the stated rules, and then moves across the river - which takes two seconds. Missionary and cannibal trees/processes are created by auxiliary tree/process. Correctly synchronize cannibal, missionary and boat trees/processes.

Sample printout

Legend: M-missionary, K-cannibal, C-boat,
LO-left bank, DO-right bank L-left, D-right

C: empty on the right bank
C[D]={} LO={} DO={}

M1: came to the left bank C[D]={}
LO={M1} DO={}

K1: came to the right bank

C[D]={} LO={M1} DO={K1}

K1: entered the boat C[D]={K1}
LO={M1} DO={}

K2: came to the left bank C[D]={K1}
LO={M1 K2} DO={}

M2: came to the right bank C[D]={K1}
LO={M1 K2} DO={M2}

M2: entered the boat
C[D]={K1 M2} LO={M1 K2} DO={}

K3: came to the right bank C[D]={K1 M2}
LO={M1 K2} DO={K3}

M3: came to the right bank C[D]={K1 M2}
LO={M1 K2} DO={K3 M3}

M3: entered the boat
C[D]={K1 M2 M3} LO={M1 K2} DO={K3}

K3: entered the boat
C[D]={K1 M2 M3 K3} LO={M1 K2} DO={}

C: three passengers on board, leaving in one second C[D]={K1
M2 M3 K3} LO={M1 K2} DO={}

M4: came to the right bank
C[D]={K1 M2 M3 K3} LO={M1 K2} DO={M4}

M4: entered the boat
C[D]={K1 M2 M3 K3 M4} LO={M1 K2} DO={}

K4: came to the right bank
C[D]={K1 M2 M3 K3 M4} LO={M1 K2 K4} DO={}

C: I transport from the right to the left bank: K1 M2 M3 K3 M4

K5: came to the right bank
C[D]={K1 M2 M3 K3 M4} LO={M1 K2 K4} DO={K5}

M5: came to the left bank
C[D]={K1 M2 M3 K3 M4} LO={M1 K2 K4 M5} DO={K5}

C: translated from right to left bank: K1 M2 M3 K3 M4 C: empty on
left bank
C[L]={} LO={M1 K2 K4 M5} DO={K5}

M1: entered the boat
C[L]={M1} LO={K2 K4 M5} DO={K5}

K2: entered the boat
C[L]={M1 K2} LO={K4 M5} DO={K5}

M5: entered the boat

C[L]={M1 K2 M5} LO={K4} DO={K5}

C: three passengers on board, leaving in one second

K5: entered the boat
C[L]={M1 K2 M5 K4} LO={} DO={K5}

C: I transport from the left to the right bank: M1 K2 M5 K4 . . .