OPERATING SYSTEM 2

SEMAPHORE REVISION

TEST II

- A fast-food restaurant can accommodate a limited number of customers. This limit is represented by the number **T** of tables available in the restaurant.
- A customer who arrives washes their hands before sitting at a table; there are only **two (02) washrooms**, each for single use.
- Then the customer waits in a waiting room if no table is available.
- When a table becomes available, the customer sits down and eats.
- As soon as the customer leaves the restaurant, the table is freed for another customer.

SOLUTION

```
program restaurant
const NT=.....;
var
W: semaphore:=2;
T:semaphore:= NT;
```

```
process customer-i
begin
  P(W);
  washhands();
  V(W);
  P(T);
  eat();
  V(T);
end;
```

TEST II

- A store can accommodate a limited number of customers.
- This limit is represented by the number **N** of shopping carts available at the store entrance.
- A customer who arrives waits if no shopping cart is available.
- When a customer gets a shopping cart, they enter the store to do their shopping.
- Then they go to the checkout to pay for their purchases in mutual exclusion, i.e., there is only one checkout.
- As soon as they finish, they release their shopping cart when leaving the store.

SOLUTION

```
program store
const N:=....;
var
cart:semaphore:=N;
check:semaphore:=1;
```

```
process customer-i
begin
        P(cart);
        shopping();
        P(check);
        chekout();
        V(check);
        V(cart);
end;
```

TEST II

- A car service center has a **limited number of parking spots**. This limit is represented by the number **P** of parking spaces available in the center.
- **There are two types of cars: regular and large. Regular cars need one spot; however, large cars occupy two parking spots** instead of one (They must wait until **two spots** are available):
- A car that arrives waits in a **waiting lane** if no parking spot is available.
- Before parking, each car must go through a **car wash**; there are only **three (03) washing stations**, each for **single use**.
- Once the car is washed, it parks in an available spot (or two spots for large cars).
- As soon as the car leaves the parking spot, it is freed for another car.

SOLUTION

```
program parking
const spot:=....;
var
S:semaphore:=spot;
W:semaphore:=3;
```

```
process car-i (size)
begin
  if size=1 then P(S)
  else begin
            P(S); P(S);
       end;
   P(W);
  wash();
  V(W);
   if size=1 then V(S)
   else begin
             V(S);V(S);
        end;
end;
```

EXERCISE 1

- A store can accommodate a limited number of customers. This limit is represented by the number **N** of shopping carts available at the store entrance.
- A customer who arrives waits if there are no carts available.
- When a customer acquires a cart, they enter the store to do their shopping.
- As soon as they finish, they release their cart when leaving the store.
- There are two categories of customers: **subscribers** and **non-subscribers**.
  - There is **no mutual exclusion** between subscribers and non-subscribers in the store itself.
  - However, **subscribers have priority** when acquiring carts.
- Customers can be modeled as **parallel processes** and carts as **shared resources**.
  - **Task:** Write the algorithms for the "subscriber" and "non-subscriber" processes using **semaphores**.

SOLUTION

```
program store
const N = .... ;
var  mutex : semaphore :=1;
     S1, S2:semaphore:= 0 ;
     NBS, NB1, NB2 : integer :=0;
```

```
Process Subscriber_Client(i)
begin
   P(mutex);
   if NBS < N then begin
                        NBS++;
                        V(mutex);
                      end
   else begin
           NB1++;
           V(mutex);
           P(S1);
            NB1--;
        end;

   shopping ();

   P(mutex);
   if NB1 > 0 then V(S1);
   else if NB2 > 0 then V(S2);
        else NBS--;
   V(mutex);
end;
```

```
Process NonSubscriber_Client(j)
begin
   P(mutex);
   if NBS < N then begin
                        NBS++;
                        V(mutex);
                      end
   else begin
           NB2++;
           V(mutex);
           P(S2);
           NB2--;
        end;

   shopping ();

   P(mutex);
   if NB1 > 0 then  V(S1);
   else  if NB2 > 0 then V(S2);
         else NBS--;
   V(mutex);
end;
```

EXERCISE 2

- Cars coming from the north and the south must cross a bridge.
- On this bridge, only one lane of cars can pass at a time, and only in one direction.
- Write an algorithm that allows cars to pass from north to south or from south to north by synchronizing them using semaphores at the bridge.

SOLUTION with STARVATION

```
program bridge
var
   northCount : integer := 0;
   southCount : integer := 0;

   mutex1 : semaphore := 1;
   mutex2 : semaphore := 1;
   bridge : semaphore := 1;
```

```
process NorthToSouthCar-i;
begin
   P(mutex1);
   northCount := northCount + 1;
   if northCount = 1 then P(bridge);
   V(mutex1);

   CROSS ();

   P(mutex1);
   northCount := northCount - 1;
   if northCount = 0 then V(bridge);
   V(mutex1);
end;
```

```
process SouthToNorthCar-j;
begin
   P(mutex2);
   southCount := southCount + 1;
   if southCount = 1 then P(bridge);
   V(mutex2);

   CROSS ();

   P(mutex2);
   southCount := southCount - 1;
   if southCount = 0 then V(bridge);
   V(mutex2);
end;
```

FAIR SOLUTION

```
program bridge
var
   northCount : integer := 0;
   southCount : integer := 0;

   mutex1 : semaphore := 1;
   mutex2 : semaphore := 1;
   bridge : semaphore := 1;
   turn : semaphore := 1;
```

```
process NorthToSouthCar-i;
begin
   P(turn);
   P(mutex1);
   northCount := northCount + 1;
   if northCount = 1 then P(bridge);
   V(mutex1);
    V(turn);
   CROSS ();
   P(mutex1);
   northCount := northCount - 1;
   if northCount = 0 then V(bridge);
   V(mutex1);
end;
```

```
process SouthToNorthCar-j;
begin
   P(turn);
   P(mutex2);
   southCount := southCount + 1;
   if southCount = 1 then P(bridge);
   V(mutex2);
   V(turn);
   CROSS ();
   P(mutex2);
   southCount := southCount - 1;
   if southCount = 0 then V(bridge);
   V(mutex2);
end;
```

EXERCISE 3 (try to resolve it)

- Consider a system composed of three smoker processes and one agent process.
- Each smoker continuously rolls a cigarette and smokes it.
- To roll a cigarette, **three ingredients are needed**: tobacco, paper, and matches.
- One smoker process has **paper**, the second smoker process has **tobacco**, and the third smoker process has **matches**.
- The agent process has an **infinite supply of all three ingredients**.
- The agent places **two distinct ingredients on the table**, allowing the smoker who possesses the **third ingredient** to roll a cigarette, smoke it, and then signal the agent that they have finished.
- The agent then places **two other ingredients on the table**, and the cycle repeats.