laSalle

UNIVERSITAT RAMON LLULL

OPERATING SYSTEMS

Academic year 2022-2023

Problems and exercises from chapter 4
Exam problems from previous years

November 1, 2022

CONTENTS

Chapter 4. Mechanisms of communication, synchronization and mutual exclusion.

Exercise 1: Precedence graph

Given the following sentences:

```
a := x + y (S1)
b := z + 1 (S2)
c := a + b (S3)
w := c + 1 (S4)
```

a) Create a precedence graph and indicate which ones could be executed in parallel.
b) Demonstrate according to Bernstein conditions which sentences can be executed at the same time.

Exercise 2: Simple synchronization

To calculate the combinatorial number (n k)=n·(n-1)·(n-2)...(n-k+1)/k! we develop a program with two processes. Process P1 calculates the numerator and puts the result in a shared variable x, while P2 calculates the factorial and puts the result to the variable y. Synchronize processes P1 and P2, using binary semaphores, so that process P2 performs the x/y division.

```
var
      n, k, x, i : integer
endvar

x:=1 i:=1

process P1()
    for i:=n-k+1 until n
        x:=x*i
    endfor
endprocess
```

```
process P2()
    y:=1
    for j:=2 until k
        y:=y*j
    endfor
    write(x/y)
endprocess
```

Exercise 3: The simultaneous ones

There are the processes P1, P2 and P3, with the corresponding instruction sequences shown. Synchronize the processes so the sequences b in P1, f in P2 and h in P3 are executed by a maximum of two processes at a time.

```
process P1()
    while TRUE do
        a
        b
        c
    endwhile
endprocess
```

```
process P2()
    while TRUE do
        d
        e
        f
    endwhile
endprocess
```

```
process P3()
    while TRUE do
        g
        h
        i
    endwhile
endprocess
```

Exercise 4: Synchronization I

We have two processes that share the same variable, x (the other variables are local to the processes). Modify the processes so that P1 uses all x values provided by P2, using binary semaphores. The values C and N are numerical constants.

```
process P1()
    while TRUE do
        m := 2*x - N
        write(m)
    endwhile
endprocess
```

```
process P2()
    while TRUE do
        read_keyboard(d)
        x := d - C*5
    endwhile
endprocess
```

Exercise 5: Synchronization II

Two processes P1 and P2 pass information through an ED data structure. The integer n, which indicates the current number of useful elements in ED, has an initial value of 0. The process P2 removes from ED in each iteration the last element added by P1, and waits if there are no elements until P1 adds one again. Suppose ED has unlimited size (large enough to never be filled).

```
var
    n: integer
endvar

n:=0
```

```
process P1()
  while TRUE do
      calculated_data:=calculate_data()
      n:=n + 1
      ED[n]:=calculated_data
  endwhile
endprocess
```

```
process P2()
  while TRUE do
      data:=ED[n]
      n:=n-1
      operate_data(data)
  endwhile
endprocess
```

Exercise 6: Assembling bikes

In a bicycle factory, we have three operators called OP1, OP2 and OP3. OP1 mounts wheels, OP2 mounts the bike frames and OP3 the handlebars. A fourth operator, the assembler, is in charge of taking two wheels, a frame and a handlebar and assemble the bicycle. Synchronize the actions of the three operators and the assembler if the operator OP1 has space to store 20 pieces and the operators OP2 and OP3 have space to store 10 pieces of those they produce. The assembler cannot take any part if it has not been previously manufactured by the corresponding operator.

```
process OP1()
    while TRUE do
        wheel()
    endwhile
endprocess
```

```
process OP2()
    while TRUE do
        frame()
    endwhile
endprocess
```

```
process OP3()
    while TRUE do
        handlebar()
    endwhile
endprocess
```

```
process Assembler
    while TRUE do
       take_wheel()
       take_wheel()
       take_frame()
       take_handlebar()
       assemble_bike()
    endwhile
endprocess
```

Exercise 7: The Restaurant

Two customers are seated at a table in the DareToEat restaurant. Each one of them is having a bowl of soup and they share a salad in the middle of the table. These two people are not very confident and cannot eat both from the salad at the same time. For each spoonful of soup, they eat a little bit of salad. There is a bartender in charge of removing the soup plates when they are empty, that is, when each of the customers has taken 10 sips of soup. Solve the problem taking into account that the bartender must remove the plates from both customers at the same time.

Implement the customer and bartender processes in pseudocode using semaphores.

You can use the following actions in the code: EatSoup(), EatSalad(), CleanUpPlates().

Exercise 8: The barbershop

Suppose a barbershop with a barber, a haircut chair, and 4 chairs for customers to sit and wait for their turn. If there are no customers, the barber, who is quite lazy, sleeps in his chair. When a customer arrives he must wake up the barber. If more clients arrive while the barber is busy and there are free chairs, they sit down and wait their turn. If there are no free chairs they leave. Program a solution for the Barber and Customers using semaphores.

Exercise 9: The hamsters

We have several hamsters in the same cage where there is a plate of food and a wheel. All hamsters want to eat and then play the wheel. But they have a couple of problems or limitations: only 3 hamsters can fit in the wheel at a time and only one can eat from the plate at a time. Define a system using semaphores that implement the Hamster process.

Exercise 10: Oktoberfest

The Beer Festival in Munich is very famous. Participants spend all day and all night drinking beer (with liter jars), then singing a song and then going to the services to make more space... like that to infinity. The place where the party takes place has only MAXSERVICES to use. You are asked to implement the Participant() process using semaphores.

Chapter 4. Exam exercises

Exam Christmas 2004: The zoo - Estimated time: 45 minutes

The Barcelona Zoo intends to have total control of the various installations with the goal of not degrading the environment that is necessary for the animals that live there. So he asked us to implement a small system that serves to control the number of visitors to the Park.

Therefore, the following specifications have been defined. First, access to the Park must be controlled. A maximum of 1500 daily visitors is allowed. So, if a visitor wants to access the Park (for any of their entrances) the system will deny them the access if this limit has been exceeded.

Once inside, the person can freely visit the whole Park with the exception of the Terrarium, an installation that needs special conditions and that for reasons of space and temperature stability we will only allow a maximum of 125 visitors at a time. If a visitor wants to enter the Terrarium and it is full, he will have to wait until someone else leaves before he can enter.

Once inside the Terrarium, there is a specially protected gallery: the African Arachnids Room. This area, to avoid noise, can only be accessed individually. So, you will have to queue to be able to visit it.

A) Briefly explain what concurrency problems you detect and what they consist of. Present it in the form of a table with the following format:

Concept	Resource	Description

B) Design and implement in pseudocode the system that allows the simulation of the requirements described above. Specifically, it is necessary to declare the shared variables that you consider necessary and initialize it if necessary. Finally, implement the Visitor() process, which is a process that simulates a person's visit to the zoo.

A person visiting the zoo is considered to perform the following actions sequentially: first try access if he can; otherwise he leaves. Secondly make a free tour of the entire zoo. He then goes to the Terrarium and, inside, visits the Hall of Arachnids. Finally, leaves the zoo. To facilitate your implementation, we give you the following already created procedures that you can call when necessary:

- FreeVisit(): simulates a free visit to the Zoo.
- VisitTerrarium(): simulates a visit to all the rooms of the Terrarium except the Arachnids.
- VisitArachnidsRoom(): simulates the visit to the Arachnids Room.

NOTE: All appropriate mechanisms must be created to resolve concurrency issues. You can use semaphores to implement some of these mechanisms, if you need to.

Exam Christmas 2004: Print at full speed - Estimated time: 45 minutes

The computers of a large Local Area Network of a company are connected to a print server that handles all job requests to print documents. This server is represented by an array of integers, called *printers*, where these represent the PID of the processes assigned to a specific printer at a given time. We can consider that we have N printers available (you can set N to any value, 10 for example). A process that wants to print must run the RequestPrinter() procedure, then it can print to the assigned printer, and finally it must invoke the ReleasePrinter() procedure that will release the resource (see below for an example of a process that wants to print).

You are asked to write the code for these two procedures RequestPrinter() and ReleasePrinter() and define the necessary semaphores so that:

A) The use of printers is regulated (a process will wait until a printer is available).
B) Assign to the requesting process the available printer from the array. To do this, the PID will be stored at the corresponding position of the array. For example, if a PID process equal to 12540 wants to print, it will look for one that is free. If there were no printers available, the process must wait; otherwise it must take the first one available (let's suppose number 7) and run `printer[7]:=12540`.

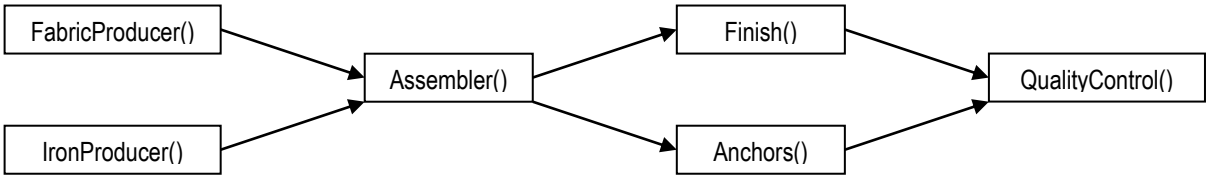
You are asked to write the pseudocode of the InitPrinters() procedure in which you must initialize the printer array to -1 to indicate that all printers are initially available. You must also initialize all necessary semaphores for managing the system.

Annex
Example of a process that requests to print a document:

```
process P1()
    var
        pid: integer
    endvar

    pid:=GetPid()
    RequestPrinter(pid)
    PrintDocument()
    ReleasePrinter(pid)

endprocess
```

Exam June 2004: The factory - Estimated time: 30 minutes

Lately everybody talks about the production capacity of the Chinese industry. Regardless of the discussions about the working environment and the conditions, it is true that they production lines are exemplary. Specifically, here we have a production line model of an industry that manufactures tents:



- FabricProducer(): is a process that is responsible for generating the necessary fabric for the tents. It uses the ProduceTentFabric() procedure to produce a unit of fabric used in one tent.
- IronProducer(): Is a process that keeps producing the metallic pieces used in the tent. The ProduceMetal() procedure generates two different metal kits that would allow to produce two tents.
- Assembler(): this process takes the manufactured fabric and 1 metal kit and assembles the tent. This operation is performed by the BuildTent() procedure.
- Finish(): this process adds the finishes to the store such as zippers, sewing pockets, etc. It uses two procedures over the tent: MountZippers() and OtherDetails().
- Anchors(): in parallel after the assembler, this other process is responsible for preparing in a separate case all the anchors needed to fix the tent in the ground. The procedure FillAnchors() prepares those necessary packs for one tent.
- QualityControl(): finally, when the tent is completely produced, the quality control process checks it using the QC() function, that returns an integer that will indicate if the tent can be safely distributed.

Implement in pseudocode all processes of the production line described above adding all synchronization mechanisms needed.

Exam June 2004: Montserrat Library - Estimated time: 45 minutes

The Montserrat Library is a cultural institution highly visited and attended by many types of public and people of science and culture. We are asked by this institution to create a system that regulates physical and virtual access to this institution.

Specifically, it can be accessed in three different ways: as a general public, as a person of culture, or virtually from the Internet. We will create 3 processes for these cases: Public(), Erudite() and Browser().

Access to the Library is limited to a maximum monthly capacity. If we want to physically access it, we will need to verify that there were no more MAXVISITORS this month. In this case, our access will be denied and we must leave. In addition, the library has 20 areas or sections that must be protected from physical access also because the volumes contained in it are very delicate. That is why we have information on the maximum number of visitors at a time in each area. If access to an area is requested and it is full, we will have to wait a while to access it.

So we have:

```
const
        MAXVISITORS = 2000
endconst

var
        MaxCapacity = array[1..20] of integer
endvar
```

In addition to these restrictions, consider the following constraints regarding the peculiarities of the 3 types of visitors:

• Erudite(): This type of visitor must meet all the restrictions explained above. He usually enters the library and makes inquiries in one or more sections.
• Public(): This visitor has the restrictions on entry to the library explained above but does NOT have access to ANY of the special sections of the library. Therefore, it is limited to a free visit to the site or public part.
• Browser(): This visitor does not have to meet any of the above physical restrictions as their query is made via the Internet. We just want the number of monthly visits to the library to be recorded through this consultation service.

You can also consider the following functions implemented: VirtualView(), PublicPartView(), and SectionCheck(n).

A) Declare all shared variables, constants, and semaphores (in addition to those already described by the problem) that you need to implement your system. Also perform its initialization.
B) Implement in pseudocode the Public(), Erudite() and Browser() processes.

If you need a tool to perform mutual exclusions or process synchronization, you can ONLY use general or binary semaphores properly initialized.

Exam September 2005: Automatized digital publications
Estimated time: 60 minutes

A document printing center wants to create an automated printing request system where each user who decides to print a document will only need to run a process on the system to make the request and print it. For any order a user, running this process, will need to configure the print format ('L' for black and white laser printing; 'C' for digital printing in PDF format that will be recorded on a CD; or ' P 'for plotter printing). Also, once the service has been granted, it will be necessary to configure the path where the document is located and its name (for example Z:\DATA\EXAM.DOC).

This system has the following resources and printing options:

- Black and white print on paper: This can be done by making a print request for the black and white laser printers available on the system. If we request a print in this mode and the 15 laser printers are busy, the customer will have to wait for one of these resources to be released before making the request and provide the name and path of the file. Once access is granted, the system will store this information in a structure that it has defined (and initialized to blank spaces) like this:

  ```
  Jobs: array[1..15] of string[1..512] of char
  ```

  Then, the request process will execute LaserBWPrint() procedure, which is responsible for printing this job. Once the printing is finished, the slot of the array used must be freed by setting it to blank spaces again.

- CD print in PDF format: If we have requested the 'C' option, what we want is a CD with a PDF copy of our document in it. In this case, we only have one CD recorder, as the demand for this option is not very common. So, as in the previous case, you will have to wait if the recorder is busy and then ask for the path and file name, and call PdfGeneration() procedure which will create the PDF and burn it into the CD.

- Plotter print: Finally, if we chose the 'P' option, it is because we want to print something in color and in large format (blueprint, panoramic photographs, etc.). Obviously, this company has only one plotters due to their cost. However, it is used a lot and it is often collapsed, so the process will not wait if the plotter is busy and it will finish. The user will need to call the process again if they want to try printing again using this method. If you have access to the plotter, you will need to request the path and file name and call PrintPlotter() to do the job.

You are asked to implement the Request() process, which performs all of the features described above. It is also necessary to declare all the variables shared between processes and their corresponding initialization. If you need to use mechanisms to resolve possible concurrency issues (mutual exclusion and/or synchronization) you can use semaphores.

Exam Christmas 2005
EXERCISE 4: The Intraweb
Estimated time: 40 minutes

A financial portal has decided to put an access control on its intraweb in order to guarantee its performance and functionality. Specifically, to access it you must enter the correct password and ensure that there are no more than 2500 customers within the intraweb at once. If any of the above two restrictions are not met, the customer won't have access at that time and is informed that they may try again later, as the service is now collapsed.

Once inside, the customer has various functionalities that also have usage restrictions depending on the number of customers who are using them at the time. This information is stored in an array of integers, which we will assume is correctly initialized with the values that the entity considers suitable.

```
var
    functionalities: array[1..MAXFUNC] of integer
endvar
```

Therefore, if a customer requests access to a functionality and there is the maximum number of customers using it, the intraweb will refuse the access. The customer doesn't need to wait for this feature and should be able to select another one.

You are asked to implement in pseudocode the IntrawebCustomer() process which must work as follows:

- The client is first asked for the password and compared with a correct one (to simplify the problem you can assume this correct password as a constant).
- Then, the Intraweb access restrictions described above are applied.
- If you have access, the customer can then request functionalities (from 1 to MAXFUNC) on the intraweb (which may or may not be granted according to the restrictions specified above) until he decides to leave the system (which will be done entering a '0' as the functionality).
- To access the functionalities, we will assume that we can call the ExecuteFunctionality(n) procedure, where n is the number of functionality to be executed, and it already performs the appropriate actions.

EXAM JUNE 2006 - EXERCISE 1: The parking lot
Estimated time: 40 minutes

A parking lot from a company in Sant Cugat has an access control system in order to guarantee its correct operation and avoid possible misunderstandings. The parking lot has several doors through which vehicles can access it. Specifically, only employees of that company can access the parking lot. To access the car parking, you must insert the card into a reader on each of these doors and the reader will give the employee a code that corresponds to the index of the array that stores the information of each employee.

```
type
    place = struct
             BarCode: integer
             Parked: bool
            endstruct
endtype

var
    parking: array[1..MAX] of place
endvar

for i:=1 until MAX
    read(parking[i].BarCode)
    parking[i].Parked:=FALSE
endfor
```

As you can see, to ensure proper access, the parking barrier will only open if the program verifies that the car is not parked inside (hence the boolean who reports it). In addition, we must ensure that there are free spaces out of the 120 available in the parking lot. Otherwise, the barrier will not go up, access will not be allowed and they will have to look for parking outside on the street.

But ... ATTENTION!! There are duplicate cards!! There are employees who as they do different time shifts (in principle) have the same card replicated. So, since they have the same Bar Code, if one of the two employees has already parked, the access system won't let him enter!! It could happen that two or more employees with the same replicated card try to enter the parking lot at the same time through different doors!! The system has to control this fact and just make sure one of the two can enter but not both!!

Finally, to control the exit access (we also have N exits) we will have another process in the exit readers that verifies that the car trying to exit is parked inside the parking lot (checking the corresponding field) before opening the barrier.

The system has two functionalities that you can consider already implemented:

- ReadCard(): function that allows the reader to read the card passed to it and return the place code of the card (which corresponds to the index of the array).
- OpenBarrier(): procedure that raises the barrier to access the car park.

It is requested to implement in pseudocode the processes EntranceParking() and ExitParking() to control the facility. If you need synchronization or mutual exclusion mechanisms, you can only use semaphores.

The efficiency and use of optimal resources in the solution will be valued.

FEBRUARY 2010 Problem 3 (30 minutes)

Bob is a sponge who lives with his pet, Gary the snail, in a two-bedroom apartment inside a pineapple in Bikini Bottom. Bob's dream is to become the most famous chef of the sea. He is an expert in making burgers (the famous *Krabby Patty*) in the restaurant where he works, the *Krusty Krab*, run by *Mr. Eugene Krabs*.

Precisely, and due to the great increase in popularity that the *Krusty Krab* has received, it has been requested to simulate the operation of the place to see if everything works correctly and quickly with the new rules. Specifically, it is necessary to simulate the Customer() process that fits the following characteristics and constraints:

- Any customer wishing to access the *Krusty Krab* knows that this restaurant has a capacity for 65 guests. According to the new regulations, a customer who wants to access the restaurant must do so whenever there is room. Otherwise it will queue, but this queue will have a maximum of 15 customers. That is, at most we will have 80 customers, 65 inside the building and 15 queuing. So, if a client arrived and the queue was 15 individuals, it would leave.
- Once he customer has accessed the place, he will have to wait for one of the waiters to come and take the order. There are 3 waiters in the restaurant.
- Then the customer usually watches TV to entertain himself while they bring him the order and then starts eating. Finally, he pays, but to collect we only have *Mr. Krabs*, so to perform this action will probably have to queue in case there were more people waiting to pay.

In order to be able to simulate this in pseudocode, you have a list of already implemented procedures that can be invoked by the Customer() process, whose actions do not need to be defined because they are evident by their own name: `Enter_Restaurant()`, `Sit()`, `Exit_Restaurant()`, `Place_Order()`, `Eat()`, `Pay()` and `Watch_TV()`.

It is requested the Customer() process in pseudocode that strictly follows the operation and restrictions specified above. This is why it is allowed to define any type of shared, global or local variables. If you need synchronization or mutual exclusion tools, only semaphores can be used.

February 2012 (45 minutes)
"Ca la Sandra" Restaurant

In a town near Barcelona there is the small restaurant called "Ca la Sandra". It is a somewhat special restaurant for the system of serving the food they have and the logistics used in the dining room. Specifically, the place has only a capacity for 15 people and they only serve one dish: the famous Ca la Sandra stew, stewed by his father who is the chef. But the most unique is the system of serving food.

In the dining room there is a single table that seats 15 people and a marmite (or huge marmite) in the middle with the famous stew. Towards noon the customers start to arrive. Each customer who gets a seat at the table fills his plate with a good serving of stew and sits down to eat. Guests arriving and not taking place are waiting their turn. The marmite is initially filled with 25 servings of stew. Every time the marmite is emptied, the first diner who has no food to serve calls the chef to refill it.

Program in pseudocode the behavior of the **Chef()** and each **Customer()** in this scenario assuming that they are all independent and concurrent processes. Use semaphores so that there are no problems with concurrence, no one sneaks in, no active waiting, and so on. Keep in mind that it takes the chef some time to refill the marmite and he doesn't like it at all to get swamped with repeated requests. Avoid them! The procedures FillPlate(), Eat() and FillMarmite() are already implemented.

FEBRUARY 2013
PROBLEM 2 (30 minutes, 3 points): The guided tour

The Bunkers Park in Martinet de Cerdanya is a museum where you must always visit with a guide. That's why a simulation of the environment has been requested to see if the planned logistics will work well. Specifically, it will be necessary to program a `Guide()` process and a `Visitor()` process (the latter of which can have N executions in a concurrent or parallel environment) following the specifications described below.

The customer wants to take a guided tour of the Bunkers Park. The visit is relatively short but as he knows there is only one guide if he sees that there are more than 25 people in the Museum he will leave and will try it another day. Once the visitor enters the Park, he will have to notify the guide and wait for the guide to call him to form a group. The group consists of the number of people that indicates the constant PEOPLE_GROUP. It is currently 5 people but this constant can change its value depending on the day and the operation must remain the same. Once the visitor is in a group, the visit starts, simulated by the Visit() procedure. Once finished, he leaves the Park.

```
const
      PEOPLE_GROUP = 5
endconst
```

Regarding the Guide process, its goal is to make as many group visits as needed. It must wait to receive requests from visitors and, when it has a group, make the visit, an action that we will simulate with the GuidedVisit() procedure.

It is asked the pseudocode the Visitor and Guide processes. If you need a tool for synchronization or mutual exclusion issues, you can only use semaphores. The optimization and resource efficiency will be valued, as well as all other requirements described in the specifications.

FEBRUARY 2013
PROBLEM 3 (20 minutes): Readers-Writers++

The following solution it proposed to solve the well-known problem of readers and writers:

```
var
  mutex_r, mutex_w: semaphore
  readers, writers: semaphore
  nr, nw: integer
endvar
```

```
nw:=0
nr:=0

Init(mutex_r, 1)
Init(mutex_w, 1)
Init(readers, 1)
Init(writers, 1)
```

```
process Reader()

  wait(readers)
  wait(mutex_r)
  nr := nr + 1
  if nr = 1 then
    wait(writers)
  endif
  signal(mutex_r)
  signal(readers)

  ReadDB()

  wait(mutex_r)
  nr := nr - 1
  if nr = 0 then
    signal(writers)
  endif
  signal(mutex_r);

endprocess
```

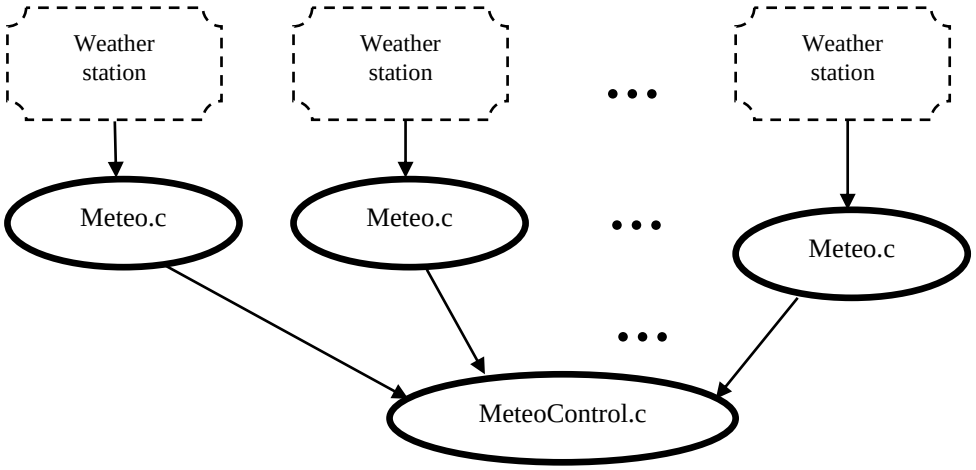
```
process Writer()

  wait(mutex_w)
  nw := nw + 1
  if nw = 1 then
    wait(readers)
  endif
  signal(mutex_w)

  wait(writers)
  UpdateDB()
  signal(writers)

  wait(mutex_w)
  nw := nw - 1
  if nw = 0 then
    signal(readers)
  endif
  signal(mutex_w)

endprocess
```

A) Does this solution solve the basic specifications of the problem? That is, allowing the access of multiple readers at the same time; and prevents that both readers and writers (or multiple writers) access the DB at the same time? Explain your response indicating clearly why, and the behavior of the system in each case.

B) Which process is prioritized in this solution? The readers, the writers, or maybe there is no priority? Is it starvation possible? Explain your response.

FEBRUARY 2013
PROBLEM 4 (40 minutes): MeteoControl

We have the requirements to be able to design and implement a system that manages and controls several weather stations distributed throughout the territory. Every geographic location where there is a weather station has a PC with a process called Meteo.c. It is responsible for collecting data from the sensors every certain time (collecting temperature, humidity and atmospheric pressure data). This program is always running and the data it collects is sent to a central computer in Barcelona which has another process called MeteoControl.c which stores and manages this information. A schematic approach would be the following:



MeteoControl.c is also a program that performs several functions. On the one hand, as mentioned, it is responsible for collecting the data sent by each station within a local variable that is defined as follows:

```
typedef struct{
            long id_station;
            char data[11];          //format "yyyy-mm-dd"
            char hour[9];           //format "hh:mm:ss"
            int humidity;           //percentage
            int pressure;           //in hPa
            float temperature;      //in °C
} Data;

void main(){
    Data register[10000];
    ...
}
```

We can assume that the array is correctly sized so it will not to have problems with the capacity. As we have mentioned, both programs are always running and if they need to be stopped, it can only be done using CTRL+C, which should allow them to be stopped, freeing system resources if there are any.

You are asked to design this entire system that has been described with special emphasis on all system tools that have been studied in Operating Systems. Optimization and resource efficiency will be valued, as well as all other requirements described in the specifications.

February 2017: Exercise 3 (60 minutes): Smarthome.

A family that likes the latest in technology has been gradually implementing a whole set of home devices that can be controlled. So they already have several appliances at home (heating, air conditioning, temperature sensor, fire alarm, refrigerator, answering machine, lights, etc.) fully connected and controlled internally in their home.

The son of this family, a student who studied Operating Systems at La Salle campus Barcelona this year, has grown up and, with everything he has learned during the course, he has told to his family that he is able to redesign the whole system they have installed and, in addition, have home automation control of the devices from outside the home, specifically via mobile. So he has told his parents not to worry that he will redesign the whole system and implement a new one that achieves this purpose. The parents put their hands on their heads, started asking for medication but, with an act of confidence towards their son, they told him to go ahead!!

The environment to be designed corresponds to the scheme (not design) in Figure 1. This environment consists of several elements. First of all, we have the different appliances distributed around the house that are controlled by an IP address. Second, we have a program on a home computer (`SmartHome.c`) that is responsible for controlling all devices. The third element is the Database, located on the same machine, which stores all the configuration parameters and values of home automation devices. Finally, we have the `Mobile.c` application which would be the one installed on the mobile device and which allows us to remotely control the whole house by connecting to `SmartHome.c`.

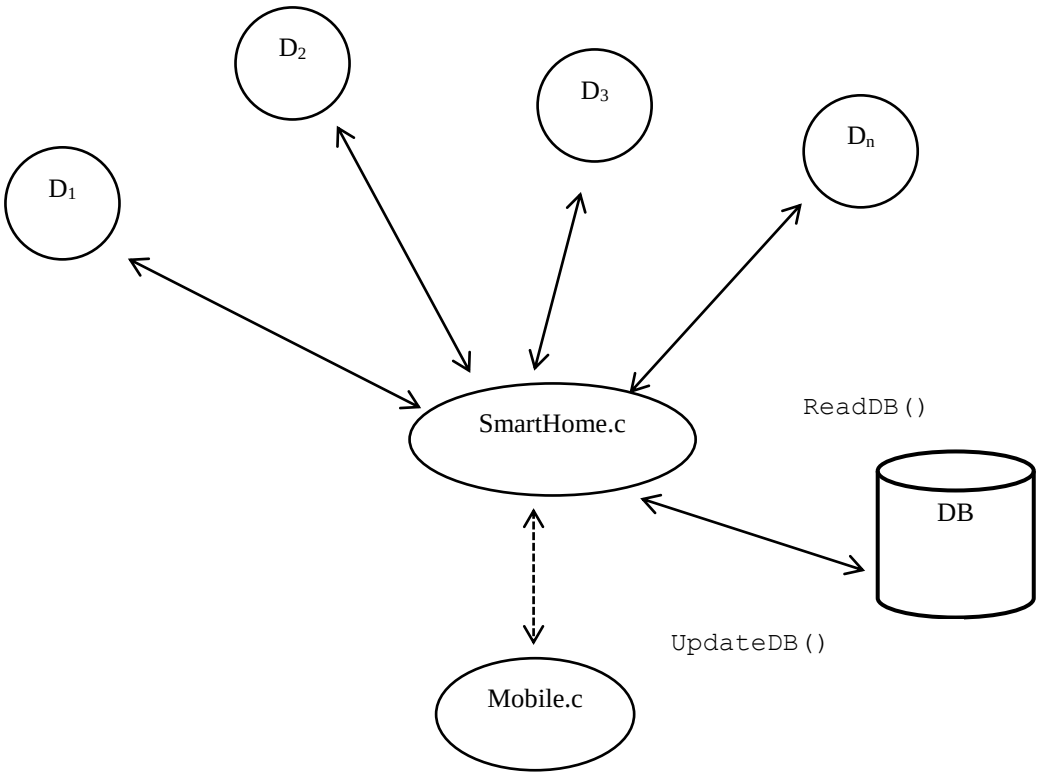


Figure 1. Descriptive diagram of the environment to be designed

From the existing environment, the student will try to reuse part of the infrastructure and some functionalities already created. For example, the module access to the Database that stores information about the devices will be the same and at this level of design we only need to know that from `SmartHome.c` you can operate on the DB by using:

- `UpdateDB()`: is a function that allows to add any information to the database, either configuration information, such as updating values and parameters of home automation devices (temperature at which the heating is on, whether it is on or off, lights on or off, sensor value temperature and humidity, refrigerator temperature, how cold the refrigerator is set, etc.).
- `ReadDB()`: is a function that allows to perform any data query of the contents of the Database.

`SmartHome.c` is the process that is responsible for controlling and operating on the various devices, either by putting them into operation or turning them off or collecting information that they have (turn on lights, look at the temperature, turn on the heating, etc.). Also from the PC has a console that allows you to configure new devices and operate on existing ones. For example, with the command `CONFIG_ADD <device_name> <ip>` it allows you to add a new device connected to the system. And with the command `CONFIG_EXIT <device_name>` allows you to disconnect an appliance that you want to remove from the home system. The user can also make inquiries from this home PC to find out information about the system. For example, `GET <device_name>` allows you to get data from the device (for example whether the light is on or not, or whether the heating is on or not and at what temperature). There are also other commands such as `COMAND <device_name> <parameters>` that allow us to operate on the device (such as turning it on, stopping it, or changing configuration settings).

While it is clear that all information is stored in the Database, for efficiency reasons, the most important one is kept in RAM. Thus the `SmartHome.c` process will keep in memory the necessary important structures, such as the list of connected devices and the data or parameters that are considered relevant.

As for the new functionality that will implement the `Mobile.c` process, it should allow us to operate the home automation system as if we were at home. That is, all the functionalities from `SmartHome.c` can now be performed from the mobile. For example, you can turn on the heating, you can turn the lights on and off, you can check the temperature at home, you can see what temperature the refrigerator is at, you can see a photo of the surveillance camera connected to the system, etc. The only two things that will not be allowed to do would be to be able to add or disable devices as it is considered that these operations can only be done from home (for obvious and security reasons).

In a first analysis, for optimization and efficiency, it has also been decided that when a mobile phone asks for information about a home automation device, this consultation will not be done at that time in the device as this would delay the response, but from the database or memory data to return this information. Therefore, it will be necessary to have the information of the various devices updated from time to time in the `SmartHome.c` program.

The parents, excited by the specifications they have been able to see (and with the pills taking effect) have obviously claimed to be able to have each one on their mobile a copy of the program.

You are asked to design all the infrastructure that has been presented according to the requirements described, justifying the design decisions and with special emphasis on the tools seen in the subject. The optimization and efficiency of the design carried out, the correctness and clarity of the proposed design schemes as well as the good explanations and reasoning written in the proposed solution will be valued.

Exercise 1 January 2019 (30 minutes): Critical operations!

We have a system that we have been asked to design a method for calculating mathematical formulas to optimize computational cost. The method is:

- We will decompose any mathematical operation into different processes that make different arithmetic calculations separately. So we will have different processes to add, subtract, multiply or divide.
- Processes that perform multiplications or divisions can ONLY do it with two operands (A*B or A/B).
- Processes that execute addition or subtraction may have more than two operands (A+B+C+... or A-B-C-...).

If we have the following formula:

$$R = \frac{x^2 + y^2 + 5}{x - 1} + t$$

It is requested:

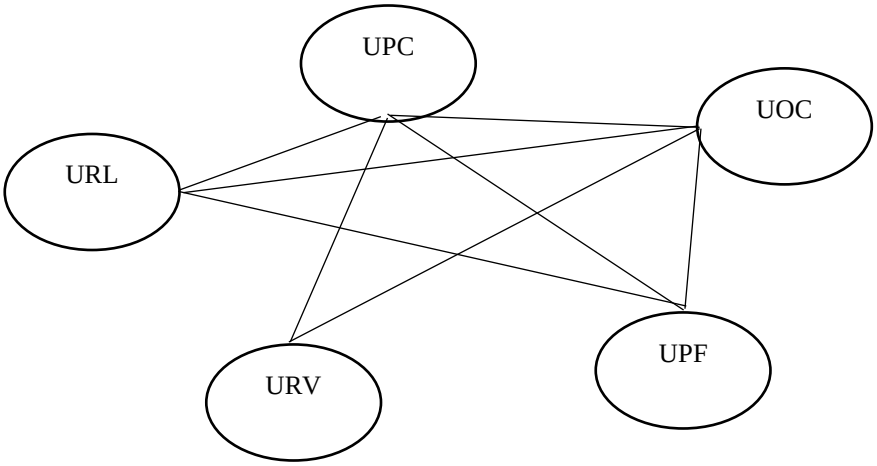
1. Make the precedence diagram showing how the formula calculation will be performed.
2. Perform the pseudocode programming that is used to calculate the formula. You can use local, global, and shared variables if you need to. All execution must work by process synchronization. The only synchronization mechanism that can be used is semaphores. Efficiency in the use of shared resources will be valued.

NOTE: you can consider that the variables x, y and t are already declared, are of real type, are shared and are also correctly initialized. It is also ensured that only one calculation of the formula will be performed.

Exercise 3 January 2020 (45 minutes): SHIT!

More and more cases of plagiarism are being detected and not just about projects, but TFM, TFG and research articles. The 12 Catalan Universities have agreed to have their own internal system where you can cross-check information on their repositories of TFM, TFG and research articles. The SHIT system (Secured History Index for Teachers) will be the program that each university will have and will manage this type of exchange.

First, SHIT has a shell with a set of commands that allows it to execute the available requests: a CONNECT to connect to another university; a DISCONNECT to cut the connection; a REQUEST to send a query to get information of a specific document; a SEND to send information that has been claimed; etc. In the figure below you can see an example of connectivity at a given time.



When a university wants to verify if another university has a copy or plagiarism of a certain document that it has, it will perform the following steps:

1. Connect-to the other university
2. Transmit a REQUEST sending the PDF file with the document to check.
3. Wait the response from that university which will send back the % of copy rate of the document detected among its original sources.

Regarding the university receiving the request:

1. It must always be prepared to attend incoming connections from other universities.
2. It has to check the received PDF file. That's why it has a feature already implemented with a separate program called Plagiarism() that, when executed with the PDF file name, returns the % of copy.
3. It must send this answer to the other university.
4. It will keep the connection in case they want to make more requests from the other university.

The typology of PDF sent is quite diverse. It can range from a simple research paper with few pages, to a very large document that includes a degree, master or doctoral thesis.

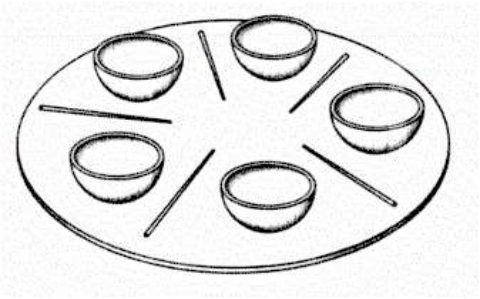
You are asked to design the entire system, with special emphasis on the system tools used. It is necessary to justify its need and specify its use. The scalability of the system as well as its efficiency will be valued.

Exercise 1 January 2020 (20 minutes): Philosophers are back!!

"What the superior man seeks is in himself; what the small man seeks is in others." (Confucius)

A class of concurrency problems is the so-called philosophers' problem. But their solutions have rarely been analyzed. Let's remember the problem first:

- 5 philosophers are engaged in thinking and eating at a circular table.
- The table is set with 5 five plates and 5 chopsticks, as shown in the figure.
- When a philosopher is hungry, he sits at the table to eat.
- The philosopher can only take one chopstick at a time, and cannot take it from the hand of another philosopher.
- When a philosopher has the two chopsticks, he starts eating without letting them go until he finishes.
- When the philosopher finishes, he goes to thinking again.



You are asked to comment on the 3 solutions indicating for each if they work or not and why. Any unreasoned answer will be discarded.

Finally, regardless of whether or not the 3 previous solutions work, you are asked to propose a solution strategy, a fourth solution, that does work. You do not need to implement it, but you do need to clearly explain the resolution strategy and resources used.

SOLUTION 1

```
process Philosopher (i: integer)

      while TRUE do
            Think()
            TakeChopstick(i)
            TakeChopstick((i+1) mod 5)
            Eat()
            ReleaseChopstick(i)
            ReleaseChopstick((i+1) mod 5)
      endwhile

endprocess
```

SOLUTION 2

```
var
   s: semaphore
endvar
Init(s, 1)

process Philosopher (i: integer)

      while TRUE do
            Think()
            wait(s)
            TakeChopstick(i)
            TakeChopstick((i+1) mod 5)
            Eat()
            ReleaseChopstick(i)
            ReleaseChopstick((i+1) mod 5)
            signal(s)
      endwhile

endprocess
```

SOLUTION 3

```
var
      s: array[0..4] of semaphore
endvar

for i:=0 until 4 do
      Init(s[i], 1)
endfor

process Philosopher (i: integer)

      while TRUE do
            Think()
            wait(s[i])
            TakeChopstick(i)
            wait(s[(i+1) mod 5]
            TakeChopstick((i+1) mod 5)
            Eat()
            ReleaseChopstick(i)
            signal(s[i]
            ReleaseChopstick((i+1) mod 5)
            signal(s[(i+1) mod 5])
      endwhile

endprocess
```

Exercise 2 (3 points, 40 minutes): Safe campus during exams!

As you know, our campus has established a protocol for conducting exams that must be respected scrupulously. But since the management team doesn't trust its operation, they want to simulate an exam situation to see whether everything goes smooth. Of course, they had chosen the OSFG group (Operating Systems Fucking Geniuses) to implement the simulation.

Specifically, they asked them to simulate the two most important processes: Student() and Teacher(). They should adjust those processes to implement the behavior that the protocol establishes for both actors. For the simulation, we will consider that there is only one Teacher() process, but there can be as many Student() processes as needed.

To simplify the protocol (which surely everyone remembers) we give you the following indications from two points of view: teacher and student.

- The teacher will have put all the exams on the table and it will not be necessary to hand them out, just to collect them.
- Initially, a teacher should be responsible for controlling students' access to the classroom. The classroom will have a maximum capacity indicated by the constant MAXCLASSROOM.
- To facilitate the logistics, the teacher will have to put a sign on the front door (single) that says EMPTY or FULL to make it easier for the student to know if it is worth entering or running to find another classroom.
- The teacher will initially be at the door giving access to the students, one by one, until the class is full (which is always guaranteed in the simulation). At this moment, the teacher will change the sign to FULL.
- Once this is done, the teacher will be extremely bored while the students take the exam (in the simulation there are NO questions) and will only have to wait for the students to warn him (again, one by one) to pick up the exams and let them out.
- The teacher, when he is sure that all the students have finished the exam, will go home and finish the process.

From the student's point of view:

- The student arrives in the classroom with enough time and sees if it is full or not.
- If it is full, the student runs to find another classroom where he or she can take the exam (and the simulation finishes).
- If there is space, he lets the teacher know that he has arrived, so that the teacher allows him to enter. He will not be able to enter without the teacher's permission.
- Once inside and seated, the exam will be on the table and the student will be able to solve it (if he knows how).

- Once finished, the student will need to notify the teacher (he can't get up!!) and wait until the teacher has picked up the exam and authorized him or her to leave and go home.

It is requested:

- That you program the Student() and Teacher() processes in pseudocode or similar, following the instructions given above.
- You can use shared variables as well as semaphores if you need to solve the problem.
- The efficiency of the solution will be valued, paying special attention not to make active waits or to have errors with the integrity of the data.
- For Student() process, you can use the following procedures already implemented: DoExam(), GoHome() and LeaveRunning().
- For Teacher() process, you can use if you need the following procedures already implemented: IndicateLocation(), CollectExam() and LeaveClassroom().

Exercise 3 (5 points, 45 minutes): Let's save the bars!

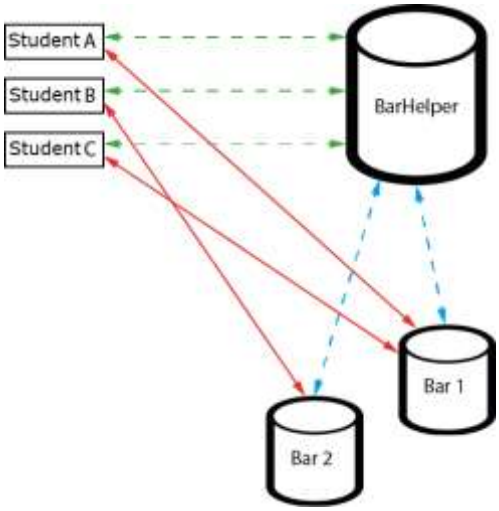
Bars and restaurants have not opened normally for many months. The limitation of hours and capacity ended in a large accumulation of people at certain times of the day, resulting in long queues for customers of the different bars in Catalonia. The students of Operating Systems have agreed to try to help as many businesses as possible, taking advantage of the knowledge acquired in the subject. It has been decided to design a set of programs to help college students go to the less crowded bars when they decide to go for a drink, before or after class.

The BarHelper system will be composed of two well-differentiated entities: Students and Bars. These two will be different processes that can be executed as many times as desired. Also, BarHelper will be a third program that will manage the interaction between students and bars, as explained below.

The process that will simulate the bars (called Bar.c) is a food server. Each restaurant will have a list of drinks and tapas. Its function will be to serve virtual customers. It will accept their orders and issue the invoice to be paid when a customer completes their order.

The customer process (which we will call Student.c) is a process that will connect to the BarHelper program. This system, and transparently to the customer, will assign one of the open bars. Then the customer will see the menu, can place orders, and can finish the orders.

As mentioned, one thing is the operation of the BarHelper system and the other is its internal architecture to ensure the scalability of the distributed system and also its efficiency. The following figure shows the system architecture to be designed:



When the Student program is running, the first thing it will do is to connect to the BarHelper program (this will always be running). Once connected, BarHelper will send the Student a Bar to connect to (IP and port). This Bar will be the one with the fewest customers at that moment (a simple load balancer). After BarHelper sends this information to Student, they will end their connection. Finally, Student will establish a new connection with Bar. This whole process should happen automatically when Student runs and should be transparent to the user.

When the Bar program runs, it will start a connection with BarHelper to tell its name and location (IP and port). Also, Bar will periodically connect to BarHelper to send the number of connected clients (Student processes) it has. These connections will not be stable over time, that is, the Bar and BarHelper processes will establish and close connections continuously. Therefore, BarHelper might receive Bar or Student connection requests at any time. Also, when Bar is running, it will start listening for connection requests from the Student processes that want to connect.

Once Student has established the connection with Bar, it will have a shell with a set of commands that will allow the customer to interact with the functionalities of the system:

- A MENU command to send a request to receive the list of dishes and drinks from the Bar.
- A PHOTO <ELEMENT_NAME> command to send a request to receive a photograph of a dish or drink from the Bar menu.
- A command ADD <ELEMENT_NAME> <NUM_UNITS> that will add certain units of a dish or drink to the customer's order.
- A DELETE <ELEMENT_NAME> <NUM_UNITS> command that will remove certain units from a dish or beverage from the customer's order.
- A PAY command that will process the customer's order.
- An EXIT command to disconnect from the Bar.

You are requested to design the entire system, with special emphasis on the system tools used (necessary data structures, data integrity systems, communication, etc). It is mandatory to justify its need and specify its use. The scalability of the system as well as its efficiency will be valued.

Exercise 1 (3 points, 40 minutes): Urban Mobility

Building a distributed application is the main goal of a microservice-based architecture. Each component of the application must be independent of the others, allowing them to be deployed on their own.

In this way, we are asked to simulate the communication between a customer and a driver within the Urban Mobility system. Microservices are not synchronized, so we will assume that there is a shared space between them that they can use. These are:

- Customer(), microservice that simulates a customer. The functionalities it has are:
  - Customers will try to connect to the system application. This only allows a total of MAX_CUSTOMERS of connected customers. If it is unable to login, then it will need to close the application.
  - If it has been able to login, it sends a message to the driver through the application and waits for a response from the microservice that handles the messages.
  - Once the confirmation message is received, it will wait for the driver to pick it up.
  - When the driver is at the meeting point, it will notify the customer who will relax and listen to the radio while the driver takes it to its destination.
  - Once they arrive at their destination, the customer only has to pay the corresponding amount.
- Driver(), microservice that simulates the driver. It has the following functionalities:
  - Initially, it will be waiting for a message from the message microservice that will notify it that a customer is requesting its services.
  - Once the message is received, it will be answered by confirming receipt to the message microservice and it will go where the customer is located.
  - Once it has reached the meeting point with the customer, it will notify the customer so that it can get in the car, and they will be able to start the journey.
  - Finally, at the end of the trip, it will have to place the amount at the POS terminal (TPV) for the customer to pay.
- Messages(), microservice responsible for managing the communication. It has the following features:
  - This microservice will be waiting to receive a message from the customer. Once received, this will be forwarded to the driver and it will wait for its response.
  - When the driver responds, it will then forward the response to the customer.
- Payment ():
  - It is only responsible for making the transaction of the corresponding amount when the Customer and Driver are ready.

Representative diagram:

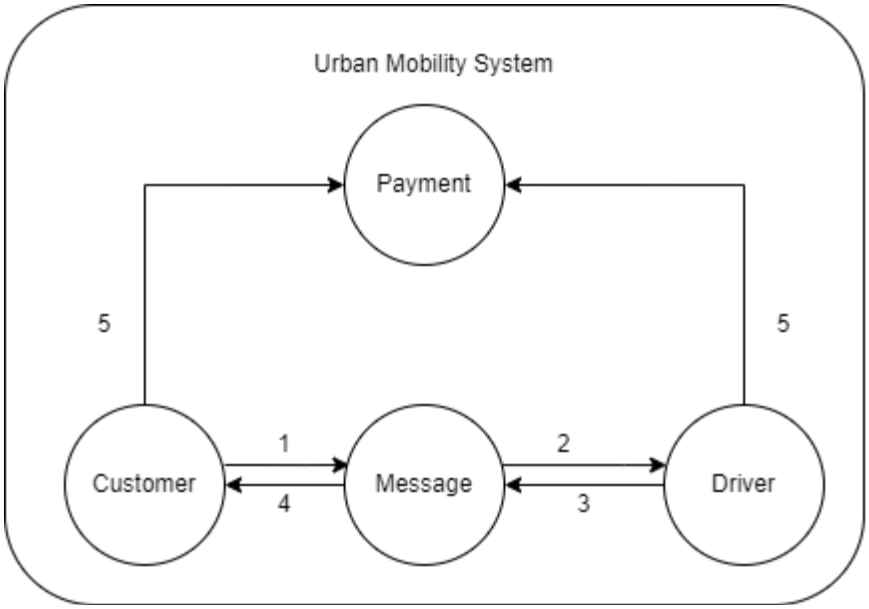


Figure 1 Precedence diagram

Note that each number corresponds to the order in which communication between processes is performed.

You are asked to rewrite the following code so that it meets all the requirements of the statement, adding whatever is necessary. Variables must be declared where appropriate (shared, global, or local) and, if synchronization or mutual exclusion mechanisms are needed, semaphores can be used by declaring and initializing them appropriately.

NOTE: Actions performed in processes can never be considered to have any implicit synchronization between the 4 processes in the statement.

```
void Customer() {
       if(num_customers < MAX_CUSTOMERS) {
              num_customers++;
              SendMessage()
              ReceiveReply()
              GetInCarAndListenToRadio()
              Pay()
              num_customers--;
       }
}

void Driver() {
       ReceiveMessage()
       ReplyMessage()
       DriveToCustomer()
       DriveAndTurnOnRadio()
       GetPayment()
}
```

```
void Message() {
       ReceiveCustomerMessage()
       ForwardCustomerMessage()
       ReceiveDriverMessage()
       ForwardDriverMessage()
}

void Payment() {
       PerformTransaction()
}
```

Exercise 2 (3 points, 40 minutes): Kitchen Nightmares!

After a tough pandemic, restaurants have been able to reopen for almost a year. Right now, most have their reservations full and many have to wait for weeks, even months, to be able to book a table.

The "Ou-perating" restaurant (specializing in omelets, of course) wants to change its operation. He wants to eliminate the archaic reservation system they had and start working without being able to book in advance. They are not sure if the new mode will work for them, and they are asking us to simulate the whole restaurant with different processes so that they can study if the new system without reservations achieves the expected results.

We will simulate the restaurant with 4 different processes, referring to the 4 people we can find in the restaurant. The processes will be as follows: Client(), Maitre(), Waiter(), and Chef(). Let's see what each of these processes does:

- Client():
  - Process that will simulate a customer who wants to go to dinner at the restaurant. This guest will be able to access the restaurant if it does not have full capacity (30 people). If it is full, the customer will queue, but will wait if there are less than 5 people in the queue. That is, if a customer arrives at the restaurant and the queue is 5 people, the customer leaves.
  - Once the customer arrives at the restaurant, it will notify Maitre to assign him a table. Once the Maitre assigns him a table, the customer will read the menu (simulated with the ReadMenu() procedure) and choose the dinner (with the ChooseMenu() procedure). It then warns one of the waiters in the room, waits until it arrives, and tells the waiter what it has chosen (Order() procedure).
  - Finally, he will wait for them to bring him the dish, Eat(), and leave the restaurant.
- Maitre():
  - Process that simulates the head waiter of the restaurant. It will receive the customers, wait for the different customers to notify him, and will guide them to the table in order of arrival.
- Waiter():
  - The waiters in the restaurant will have two functions: first of all, they will check if the Chef() has prepared a dish that needs to be distributed. If there are any left, they will distribute it (DistributeOrder()).
  - In the event that there is no dish ready to be delivered or after having delivered a dish to a customer, the waiters will wait for a customer to notify them so that they can take note (TakeOrder() procedure, which must be executed at the same time as the Order() procedure of the Client process). Once they have the order, they will notify the cook to prepare it.
- Chef():
  - Process that simulates the chef of the restaurant, who will receive orders and prepare food (with the procedure CookOrder() will prepare an order).

You are asked to implement the Client(), Maitre(), Waiter(), and Chef() processes in pseudocode. These processes must strictly follow the operation and restrictions specified above. You are allowed to define any type of shared, global, or local variable. If you need synchronization or mutual exclusion tools, you should use semaphores.

Exercise 3 (4 points, 60 minutes): CatSalut asks us for help!

We all know everything is crazy: Omicron, Deltacron, Flurona, etc. The systems are saturated and, with no intention of making fun of the problem (just the opposite) in the face of saturation of systems, CatSalut has asked us to help them design a new system, very basic but agile and efficient, which will improve the communication of positive cases within Catalan territory. Of course, the SOS Group (Special Operating Systems group) is always ready to provide a solution!

The scenario is familiar to everyone, but we'll give you a quick overview of the basic elements and requirements of the system that you have to design.

We must first think of 3 major players in the system: people, pharmacies, and CatSalut.

- For the general public, it is important to think that access to the system will be done through a program for the mobile device (AppCovid.c)
- For pharmacies, there will be a program on a server for each of the pharmacies (FarmaCovid.c)
- For CatSalut there will be a central server with large disk capacity and main memory to be able to provide adequate service to the application to be designed (CatCovid.c)

The most important feature to consider, obviously, is to create a communication of positive cases system as agile and fast as possible. To do this, the sequence must be followed:

- A positive person may have detected it by a self-test. In this case, it will use the mobile application to send a set of basic data to a qualified pharmacy. It will be necessary to send the basic data (DNI, Date, vaccination, etc.), as well as a document in PDF or JPG with the test performed. When a person is positive they will immediately have to send this information from AppCovid.c to FarmaCovid.c.
- A person can also detect their positive by taking a test at a pharmacy. In this case, it will be the pharmacy directly through its FarmaCovid.c program that will be responsible for sending the positive information to CatSalut. Remember that, as in the first case, only the data part of the person will be sent to CatSalut.

As you can see, FarmaCovid.c is the responsible application (and the only channel) to be able to communicate with CatCovid.c. In order not to saturate the system, each pharmacy is expected to send all the data collected so far (only data, never PDF or images that are stored in pharmacies) only twice a day.

To end this first part of the description of the functionality, it must be said that CatCovid.c will have a data structure that allows it to save all the files of ALL the positives sent with their data (DNI, date of notification, vaccination, etc.).

Once all this protocol is designed, we will add a set of additional basic functionalities to the system. We describe them briefly for the design implications they can have:

- A feature for FarmaCovid.c so that a pharmacy can connect to the system and ask CatCovid.c to send you a list of all positive cases that that pharmacy has reported in a given period (start date - end date).
- A feature for AppCovid.c so that a certain person can confirm that their positive case has been successfully registered in FarmaCovid.c.
- A feature for FarmaCovid.c to display of all documents names received in a given period.

You are required to design the entire system, with special emphasis on the system tools used (data structures, data integrity systems, communication, protocols, etc.). Justify in a reasoned way its need and specify its use. The scalability, efficiency, and the clarity of the proposal will be valued more than ever (it would help schemes that facilitate the understanding of the proposed design).