Three processes $P_A$, $P_B$, and $P_C$ execute three operations $A$, $B$, and $C$ respectively. The operations are to be synchronized, which is accomplished by means of semaphores:

\[
\text{var } SA, SB, SC : \text{semaphore};\\
SA := 1; \quad SB := 0; \quad SC := 0;\\
\]

\[
\begin{align*}
\text{process } P_A: & \quad \text{repeat} \quad p(SA); \quad A; \quad v(SB); \quad \text{forever} \\
\text{process } P_B: & \quad \text{repeat} \quad p(SB); \quad B; \quad v(SC); \quad \text{forever} \\
\text{process } P_C: & \quad \text{repeat} \quad p(SC); \quad C; \quad v(SC); \quad \text{forever} \\
\end{align*}
\]

**Question 1.1:**

Draw a Petri Net in which the three operations $A$, $B$, and $C$ are synchronized in the same way as in the above program. In the net, the operations should be represented by transitions.

**Question 1.2:**

Let the number of times the operations $A$ and $C$ have been executed be denoted by $a$ and $c$ respectively. Define a predicate $I$ which characterizes the reachable combinations of $a$ and $c$ in the above program.

$I$ should thus be an invariant of the program, but you need not show this.

**Question 1.3:**

The operations are now to be executed by three sequential CSP-processes $P_1$, $P_2$, and $P_3$ respectively:

\[
\begin{align*}
\text{process } P_1: & \quad \text{repeat} \quad A \quad \text{forever} \\
\text{process } P_2: & \quad \text{repeat} \quad B \quad \text{forever} \\
\text{process } P_3: & \quad \text{repeat} \quad C \quad \text{forever} \\
\end{align*}
\]

Show how the processes may exchange void messages using CSP’s synchronous communication so that $A$, $B$, and $C$ are synchronized in the same way as in the above, semaphore-based program. [The operation to be executed by a process may occur more than once.]
PROBLEM 2  (approx. 30 %)

The questions in this problem can be solved independently of each other.

Question 2.1:

Let $x$ and $y$ be integer variables. Consider the four statements $a$, $b$, $c$, and $d$:

$$
\begin{align*}
a: & \quad \langle x := y \rangle \\
b: & \quad x := x + 1 \\
c: & \quad y := y + x + 3 \\
d: & \quad x := y - 1
\end{align*}
$$

[Above, $\langle \ldots \rangle$ indicates that the statement is executed indivisibly.]

(a) For each of the six possible selections of two different statements, determine whether the two statements are \textit{mutually atomic}.

(b) Assume that the statements $c$ and $d$ are executed concurrently. For each of the two statements, draw a transition diagram showing its atomic actions.

(c) Determine all possible final values of $(x, y)$, if the concurrent execution of $c$ and $d$ is started in the state $(0, 0)$.

Question 2.2:

Let $x$ and $y$ be integer variables. Determine for each of the predicates $P$, $Q$, and $R$ whether it is preserved by each of the actions $a$, $b$, and $c$:

$$
\begin{align*}
P & \triangleq x + y > 0 & a: & \quad \langle y := x - 1 \rangle \\
Q & \triangleq 0 \leq y < x & b: & \quad \langle x < 0 \rightarrow y := -x \rangle \\
R & \triangleq y \neq x & c: & \quad \langle x := x + 2 \rangle
\end{align*}
$$

Question 2.3:

Let $x$ and $y$ be integer variables and let the temporal logic formula $F$ be defined by:

$$
F \triangleq (x > 0 \rightarrow x = 0) \land \square(y \geq 0 \land \lozenge y = 1) \land \square x \geq y
$$

(a) Let states be given by pairs $(x, y)$. Give an example of an execution for which $F$ holds. The execution should be given as a short sequence of states which is repeated forever.

Now, consider each of the following actions within the program:

$$
\begin{align*}
a_1: & \quad \langle \text{await } x = 1 \rangle & a_3: & \quad \langle \text{await } x > 0 \lor y = 0 \rangle \\
a_2: & \quad \langle \text{await } y = 1 \rangle & a_4: & \quad \langle \text{await } x > 0 \land y = 0 \rangle
\end{align*}
$$

Assume that control has reached the particular action and that $F$ is valid for the program.

(b) Determine which of the actions will be eventually executed assuming weak fairness.

(c) Determine which of the actions will be eventually executed assuming strong fairness.
PROBLEM 3  (approx. 40 %)

The questions in this problem can be solved independently of each other.

In a system, a single sender process is to send messages of a given type Message to be received by one or more out of \( n \) \((n > 0)\) receiver processes (multicasting). Receivers are identified by an index \( id \) of the type \( \text{Range} = \{1, 2, \ldots, n\} \). The sender calls the operation \( \text{send}(msg, k) \) to send the message \( msg \) to at least \( k \) \((k \in \text{Range})\) distinct receivers. The operation does not return until \( k \) receivers have obtained the message. The receiver with index \( id \) calls the operation \( \text{get}(id) \) to receive the latest message sent. A receiver can receive a given message only once. Below, a monitor implementation of such a communication mechanism is shown.

```plaintext
monitor Multicast

var cur : Message
lim : Range
S : set of Range := Range;       // Receivers having got current message
NewMsg, Done : condition;

procedure send(msg : Message, k : Range) {
  cur := msg; lim := k;
  S := \{\};
  signal_all(NewMsg);
  wait(Done)
}

function get(id : Range) returns Message {
  if id \in S then wait(NewMsg);
  S := S \cup \{id\};
  if |S| = lim then signal(Done);
  return cur
}
end
```

\(|S|\) denotes the size of the set \( S \), i.e. the number of elements it contains.

**Question 3.1:**

(a) Explain what is achieved by initializing the set \( S \) to the full range of receivers.

(b) Define a predicate \( I \) expressing that calls of \( \text{send} \) do not wait unnecessarily and argue that \( I \) is an invariant of the monitor.

(c) Explain why the monitor would not work correctly if spurious wake-ups could occur and show how to make it robust towards these.

**Question 3.2:**

For a system with \( n = 3 \), assume that each of the three receivers repeatedly calls \( \text{get} \) with its unique \( id \) and that the sender sends the messages \( a, b, c, \) and \( d \) by the following sequence of calls:

\[
\text{send}(a, 3); \; \text{send}(b, 1); \; \text{send}(c, 2); \; \text{send}(d, 3)
\]

Enumerate all the possible message sequences the receiver with \( id = 1 \) may receive.

---

The problem is continued on the next page
Question 3.3:
Show how \( m \) processes, \( P_1, P_2, \ldots, P_m \) (\( 2 \leq m \leq n + 1 \)), may use the given monitor \( Multicast \) for barrier synchronization.

Question 3.4:
The functioning of the given monitor \( Multicast \) is now to be implemented by a module with the following specification:

\[
\begin{align*}
\text{module Multicast} \\
\quad \text{op send}(\text{msg} : \text{Message}, k : \text{Range}); \\
\quad \text{op get}(\text{id} : \text{Range}) \text{ returns Message};
\end{align*}
\]

Write a server process for the module \( Multicast \) which serves the operations by rendezvous in such a way that it functions like the given monitor \( Multicast \) as seen from the sender and the receivers.