The project aims to simulate the communication between ‘N’ worker processes and a detector.
Each of the workers has 2 possible states: *Active and Idle.*
An active process can do one of the following *three* actions:

1) send a message to another process
2) receive a message from another process
3) go idle and notify the detector

The detector process captures the number of messages sent and received by a process when it goes idle.
The aim of this project is to detect a termination state in the distributed system where every process is idle and all channels are empty.

PART - 1:

Terms and variables involved in the program:

    chan = [n \in 0..N |-> <<>>];
        A global variable which serves as the channel/ inbox for each process.

Process Variables:

    outc=0;
        A variable which stores the number of messages sent out from a process.
    inc=0;
        A variable which stores the number of messages received by a process.

Detector Variables:

    outS=0;
        A variable which stores total sent messages by all processes.
    inS=0;
        A variable which stores total received messages by all processes.

In this part-1 of the project, the detector captures the total number of messages sent and received by all processes as a whole in the system.

Safety Property:
Safety == (done[0] = TRUE) => (\A x \in Procs: active[x] = FALSE) /\ (\A y \in Procs: Len(chan[y]) = 0)

The safety property above makes sure that, If done is true in the detector, then All processes are idle AND all channels are empty.
But this has been violated by the model checker in the part-1 implementation and the following counter example is shown:

The series of events occurred are:
1 -> 2
1 -> IDLE
2 -> IDLE (The message sent by 1 is still in the channel)

2 -> active (1’s message has been parsed now)
DETECTOR ADDED 1 INTO NOTIFIED => {1} and outS = 1, inS = 0
2 -> 3
3 parsed the message sent by 2
 3 -> idle
DETECTOR ADDED 2 INTO NOTIFIED => {1, 2} and outS = 1, inS = 0
DETECTOR ADDED 3 INTO NOTIFIED => {1, 2, 3} and outS = 1, inS = 1, done became TRUE

Here all 3 processes are included in notified and outS = inS. So the detector reported this as terminated.

But 2 is still active.
Thus, this implementation doesn’t capture the correct termination state and Safety is violated.

PART - 2:

The problem with the first implementation is that we are not able to capture the messages in transit i.e., messages present in the channels which cause an idle process to become active again.
So, we need to consider the state of channels along with the state of processes for determining termination.

Terms and variables involved in the program:
    chan = [n \in 0..N |-> <<>>];
        A global variable which serves as the channel/ inbox for each process.
Process Variables:
    outc=[n \in 1..N |-> 0];
        An array which stores the number of messages sent to every other process.
    inc=0;
        A variable which stores the number of messages received by a process.
Detector Variables:
    outS, inS=[n \in 1..N |-> 0];
        Arrays to store number of messages sent and received by individual processes.

Program Explanation:

Worker Processes:

The statement **while(T<MaxT)** restricts each process to do at most MaxT actions.

Action Send:

The statement **if(active=TRUE)** is used because, if this send action is chosen when the process is idle, then it should not send any message to any process.

Randomly choose a process ‘k’ and increment the outc[k]
Send a message into the channel of k

Action receive:
Set active to TRUE, because the process might have been idle before this message is received. Increment ‘inc’ of this own process.

Action go to idle:
Set active to FALSE
Send the current values of outc and inc to detector.
Reset the outc and inc values so that we don’t add the duplicate values to outS and inS in detector .

Detector process:

```
If an idle message is received,
    Include the corresponding process in the ‘notified’ set.
    Add each value of outc[i] to outS[i]. So, messages sent from this process are captured.
    Add inc to inS[i]. So, messages received by this process are captured.
Now, we need to check if the termination point has reached or not.
If all processes are included in notified set,
    If outS and inS values are equal for all channels
        Then terminate
    Else
        Continue receiving the messages worker processes
```

Satisfying safety property:

This works now because, in the Detector process, we have already checked whether all processes notified or not. Now, since all channels are empty, there is no chance for any idle process to become active again.

Progress Property:

Progress == (\A x \in Procs: active[x] = FALSE) /\ (\A y \in Procs: Len(chan[y]) = 0) => done[0]

If all processes are idle and all channels are empty it means that the system has terminated. So, done should become true when this happens.