Answer 1.

Program: BlockingK
Initialize: Buffer[1...N]
    I = 0
{Start}
For I = 0 to N
    Send (P, Buffer [I])
Next
{End}

Program: NonBlockingP
Initialize: Buffer[1...N]
    I = 0
{Start}
    do true
        If Message available on channel
            Receive(Buffer[I++])
        fi
        Work on G.
    od
{End}

Program: BlockingP
Initialize: Buffer [1...N]
    I = 0
{Start}
    do true
        Receive (Buffer[I++])
        Work on G for Timeslot t.
    od
{End}

As process K is blocking and this program is sending only one key-stroke at a time, channel capacity is one. In first program where process P is non-blocking; process P checks if any message is available on channel. If it is then it receives it in Buffer otherwise it continues with G. As receive function is non-blocking, P will return out of that function in definite time. So every task scheduled with P gets fair amount of time.

In second program where P uses blocking receive it gets stuck on receive function until some message arrives from process K. Once it arrives it gets stored in Buffer and then from some predefined timeslot G is executed. In this mode, there is fairness involved.
Answer 2.

Program: ProcessP
Initialize:
{Start}
  Move! N
  For I=0 to N
    Move! Buffer[I]
  Next
{End}

Program: ProcessMove
Initialize:
Start
  P? Length
  Q! Length
  For I=0 to Length
    P? Buffer[I]
    Q! Buffer[I]
  Next
End

Program: ProcessQ
Initialize:
{Start}
  Move? Length
  For I = 0 to Length
    Move? Buffer[I]
  Next
{End}

This program initially transfers length of the buffer and then transfers the complete buffer.

Answer 3. A. Message passing model

Channel is assumed to be reliable with no message loss.

Program: Client
Initialize:
{Start}
  do true []
    if resource required
      Send Res_Req message to server.
      Wait for Res_Ack.
      Utilize resource.
      Send Res_Rel message to server.
    fi
  Run the usual background computation.
In this program, first come first served technique is used. If the resource is available it is immediately assigned to requesting process. If the resource is currently being used, then the request is stored in the queue and is processed later when resource becomes available. Finite size of Q puts limit on number of request those could be buffered.

B. **Shared Memory model**

Assume that array Requests is shared across server and clients.
If Request[I] is true
    Request[I] = 2
    Wait while Request[I] != 0
fi
{End}

This program gives the priority to process according to their pids. Request[ pid] = 0 means process does not need the resource. Request[ pid] = 1 indicates a request and 2 indicates request grant.

**Answer 4.**
Following is the program that demonstrates how a distributed program can be used to evaluate maximum temperature.

**Program: MoteEvaluate**  
Initialize: Temp = Current Mote Temperature.  
{Start}  
    Broadcast My_Temp message containing Temp.  
    do My_Temp is received  
        if My_Temp.Temp > Temp  
            Temp = My_Temp.Temp  
            Broadcast My_Temp message containing Temp  
        fi  
    od  
{End}

Above program assumes that network of motes is a connected network. This program guarantees correctness but does not guarantees termination.

**Answer 5.**  
No. The program does not guarantee to terminate and bring system in stable correct form. Consider following table.

<table>
<thead>
<tr>
<th>Parent</th>
<th>Child 0</th>
<th>Child 1</th>
<th>Child 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Steps can repeat themselves for infinite times and not come to a stable correct state.