Fault Tolerant of Communication Link on Distributed System

Hany Ferdinando

Department of Electrical Engineering
Petra Christian University, Siwalankerto 121-131 Surabaya, Indonesia – 60236

Abstract: In distributed system, every node is connected to each other so that they can do some communication. They communicate not only data but also command control from one node to other nodes. The communication is essential, therefore, it should be available all the time. If the communication link fails, then the distributed system can be halted. To this case, a fault tolerant on communication link can be proposed. It means the system will use one of the alternative links when the primary link fails. The implementation uses link driver concept of the CT Library. For demonstration purpose, there are two nodes of ADSP-21992. One node sends data to the other node. When the communication link fails, the system switches to the alternative link. The primary link is the CAN (Control Area Network) bus and the alternative one is SPORT (Serial PORT). System always uses the highest link which is available. The experiments show that it is easy to build fault tolerant of communication link on the distributed system. Response of the system is limited by the method in which the link detection is implemented. Another constraint is the speed of data transfer rate used on the fieldbus.

1. INTRODUCTION

In distributed system, there are several nodes connected to each other to make a network. One can use fieldbus to connect those nodes. Nodes send and receive data from other nodes. Therefore the availability of the communication link is essential or the communication link should be reliable all the time. For this purpose, the distributed system must have the fault tolerance feature using the existing fieldbus. Fault tolerant is an approach by which the reliability of a computer system can be increased [1].

The fault tolerance is made on top of the fieldbus protocol using the CT Library, developed by Control Engineering group, University of Twente, the Netherlands. Special remark is on the link driver concept proposed by Hilderink [2], because all implementations are based on this concept.

The system uses two ADSP-21992 development boards as the node. Each board has its own task, but they communicate to each other. One board sends data to the other board. They synchronize this communication; therefore the system will not loose the data.

2. DESIGN AND IMPLEMENTATION

The whole implementation is based on figure 1. Producer, on one node, sends data to consumer on the other node through channel. Producer and consumer as processes should communicate to each other via channel. Processes communicate via channel do not care whether the real communication is via wireless, fieldbus, etc.

This project chose the CAN (Control Area Network) bus and SPORT (Serial PORT) as the communication link. The CAN bus serves as the primary link with the highest priority, while SPORT is the alternative link. The system always uses the available link with the highest priority.

From figure 1, the implementation is expanded as seen in figure 2. First, link driver only consists of one user’s link driver. The extended version as shown in figure 2 uses the general link driver for communication.

![Figure 1. Producer and consumer communicate to each other and synchronize via channel](image1)

![Figure 2. Extended concept of figure 1](image2)
Therefore, this general link driver should have certain method to plug the user’s link driver for the system.

For demonstration purpose, the system uses two ADSP-21992 boards and the CTC (part of the CT Library for C). Structure of the class used in software design is available in [3]. Link driver for the CAN bus and the SPORT are made according to this class structure.

The CANLinkDriver and the SPORTLinkDriver are derived from abstract class, called LinkDriver. This class is abstract class provided by the CTC. Write and read method should be made under the restriction of this class. To write those methods, one has to understand the operation of the CAN bus and the SPORT both for sending (write method) and receiving (read method) because these operations need to access specific register in the controller, in this case ADSP-21992.

This project uses two active processes called producer and consumer. Producer produces value and sends to consumer. Both processes are hardware independent part of the system. It means, the node can use any processor or controller without producer and consumer modification.

For synchronization, the semaphore concept proposed by Dijkstra [4][5] is used. With semaphore, producer and consumer can do certain synchronization for data communication. Figure 3 [3] shows the detailed operation for producer and consumer.

Now the implementation of fault tolerant is discussed. The main idea is to make the connection reliable. It means the system must know whether the communication link is good or not. Figure 4 [3] shows this idea in timing diagram, while figure 5 [3] is in sequence diagram.

There is one master and one slave for this method. Master sends regular message to slave and waits for an echo from the slave. When there time out for the waiting time, the master know if the connection is not available anymore. The slave waits for the message from the master and send the echo back. If within predefined time there is no message from the master, then the slave know that the connection is unavailable.

To implement this idea, the master needs to timers. One for timing the regular message transfer and the other for the time out. For the slave, it needs only one timer.

This idea can be expanded for the multi slave. The idea is using time frame for each slave. There is some kind of token for the slave to send echo back to the master. With this idea, the master not only know whether the connection to that slave is available or not but also know that the slave is in trouble and cannot do certain communication.

The system is designed to use the available communication link with the highest priority. When the primary link is broken, the system uses the alternative one. However, the system will not use the alternative link all the time. When the system detect that the primary link is available again, this is used again.

3. EXPERIMENTAL RESULTS

The first experiment is to connect the producer and the consumer with either the CAN bus or the SPORT. The goal of this experiment is to show that both processes can use any communication link without modification. The result is interesting, since both process can communicate to each other, either with the CAN bus or the SPORT.

Table 1 shows the result of that experiment with the CAN bus, while table 2 shows for the SPORT.

<table>
<thead>
<tr>
<th>CAN speed (kbps)</th>
<th>Time consumed (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.00</td>
<td>20.4413</td>
</tr>
<tr>
<td>12.50</td>
<td>13.0220</td>
</tr>
<tr>
<td>25.00</td>
<td>6.5775</td>
</tr>
<tr>
<td>50.00</td>
<td>3.3906</td>
</tr>
<tr>
<td>80.00</td>
<td>2.1974</td>
</tr>
<tr>
<td>100.00</td>
<td>1.7986</td>
</tr>
</tbody>
</table>

The time consumed is not constant actually; therefore, the measurements are taken for several experiments and both tables show the average values. The CAN bus and the SPORT looks like to have the same value.
Table 2. Time consumed using the SPORT with various data transfer rate

<table>
<thead>
<tr>
<th>SPORT speed (kbps)</th>
<th>Time consumed (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.00</td>
<td>18.7963</td>
</tr>
<tr>
<td>80.00</td>
<td>2.0880</td>
</tr>
<tr>
<td>800.00</td>
<td>0.4728</td>
</tr>
</tbody>
</table>

The next experiment is to test the method to detect the broken connection. One board serves as the master sends regular message to the slave and waits for an echo as explained before. One needs to calculate how often the master can send its message to the master. If the master sends message too often, it will overload the network because this message has the highest priority. It means, other message (data) cannot be sent via the network. Sending message too seldom makes the response of the system when there is a problem with the connection late. Therefore, this should be decided carefully. For the CAN has more variation for data transfer rate, all calculation is based on the table 1.

The time out value of the system is chosen three times the value of table 1, while the cycle time is four times. With this value, the network will not be overloaded by this mechanism. Actually, to decide this value, one should do some sophisticated calculation but for this simple application, that calculation is too expensive. Beside, this calculation is also problem dependent.

Table 3 shows the value of those calculations. From the experiment with those values, the system can work properly. It means, the system can switch from primary link to alternative one and vice versa and the system does not lose the data.

Table 3. The values for cycle time and time out for link checking

<table>
<thead>
<tr>
<th>CAN speed (kbps)</th>
<th>Time out (ms)</th>
<th>Cycle time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.00</td>
<td>61.3239</td>
<td>81.7652</td>
</tr>
<tr>
<td>12.50</td>
<td>39.0660</td>
<td>52.0880</td>
</tr>
<tr>
<td>25.00</td>
<td>19.7325</td>
<td>26.3100</td>
</tr>
<tr>
<td>50.00</td>
<td>10.1718</td>
<td>13.5624</td>
</tr>
<tr>
<td>80.00</td>
<td>6.5922</td>
<td>8.7896</td>
</tr>
<tr>
<td>100.00</td>
<td>5.3958</td>
<td>7.1944</td>
</tr>
</tbody>
</table>

4. DISCUSSION

A fault tolerant for communication link on the distributed system is proposed and tested. The idea of detecting link by sending regular message can be used to check the availability of the network connection. This method is problem specific and needs more sophisticated calculation for more complex problem.

The producer and the consumer are built based on the separation of hardware dependent and hardware independent part as proposed by Hilderink [2]. This concept is helpful for embedded system application where the platform can be anything but the processes are the same. Experiment with the producer and the consumer using different communication link shows this result.

Although one interested in seeing how fast the system will respond if the communication link is broken, this cannot be fulfilled. The explanation is simple. The system switch from one link to another link is based on the time out value (see table 3). For example, the time out is 5 ms. After sending message to the slave, the master has to wait for an echo in 5 ms. If the communication link is broken 1 ms after the master sent its message, the system will not response it immediately. The master waits until the timer is time out. The situation on the slave is the same as on the master. With different link checking method, this point can be measured.

Reference:
Master Slave

Wait for an echo
Send data
Time out
Get echo and set the flag
Wait for an echo
Send data
Time out
Get echo and set the flag

Read data
Set the flag
Wait for regular data
Set the flag
Wait for regular data

Figure 4. Timing diagram of link checking method

Receive ISR
set the flag
Timer's ISR
clear the flag

Figure 5. Sequence diagram of link checking method