Jurnal Teknik Elektro Vol. 4, No. 2, September 2004: 126 - 132

Pustaka *Communication Thread for Java* (CTJ)

Hany Ferdinando[1], Zulkifli Hidayat[2]
[1] Fakultas Teknologi Industri, Jurusan Teknik Elektro Universitas Kristen Petra
email: hanyf@petra.ac.id
[2] Control Engineering Group University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands

Abstrak

Pustaka *Communication Thread for Java* (CTJ) merupakan bagian dari pustaka (the CT Library) yang dikembangkan oleh Control Engineering group di University of Twente. Dengan menggunakan pustaka ini, seseorang dapat melakukan pemrograman berbasis *Communication Sequential Process* (CSP) pada PC. Sebelumnya, pemrograman CSP hanya dapat diimplementasikan lewat Occam dan diaplikasikan pada transputer. Pada pustaka ini, dikembangkan konsep *hardware independent* dan *hardware dependent*. Bagian yang *hardware independent* adalah program-program seperti perhitungan aritmatika, operasi logika, dll. Control bagian yang *hardware dependent* adalah pembacaan data dari ADC, proses pengiriman data lewat fieldbus, dll. Dengan menggunakan konsep ini, seorang dapat dengan mudah melakukan migrasi program ke platform yang lain dengan hanya mengubah bagian program yang *hardware dependent*. Paper ini bertujuan untuk memperkenal pustaka CT yang dapat diaplikasikan pada system embedded dan system kendali terdistribusi. Secara khusus, percobaan akan dilakukan pada bagian program yang *hardware dependent*, karena bagian ini yang sangat terkait dengan aplikasi pada system embedded dan system kendali terdistribusi.

Kata kunci: pustaka CT, CSP.

Abstract

The Communicating Thread for Java (CTJ) is part of the Communicating Thread (CT) Library developed by Control Engineering group, University of Twente, the Netherlands. One can write software based on Communicating Sequential Process (CSP) concept on PC with this library. Before, CSP programming can only implemented through Occam programming with transputer as its target. Inside the CT Library, the hardware independent and hardware dependent concept are proposed. As examples for the hardware independent part of the program are arithmetic operation, logic operation, etc. Examples from the hardware dependent are to read data from ADC, to send data via a fieldbus, etc. This concept enables one to move one system to another by changing the hardware dependent part of the program. The goal of this paper is to introduce the CT Library for embedded system and distributed control system application. The experiments done in this paper is emphasized on the hardware dependent part, since it relates to embedded system and distributed control system application.

Keywords: *the CT Library, CSP.*

1. Pendahuluan

Konsep *Communication Sequential Process* (CSP) pertama kali diperkenalkan oleh Hoare pada tahun 1980an [1]. Saat itu, Hoare memikirkan diperlukannya semacam tools untuk menganalisa suatu program. Dalam CSP, semua yang bagian yang dapat dijalankan disebut sebagai *proses*. Proses-proses ini saling berhubungan lewat suatu *channel*. Proses-proses ini dapat dikomposisi secara *parallel, sequential*, atau bahkan membuat pilihan alternatif. Namun, saat itu CSP masih berupa konsep. Akhirnya konsep ini diterapkan dalam sebuah bahasa pemrograman yang diberi nama Occam dan diterapkan dalam suatu jenis *microprocessor* yang disebut Transputer. Tetapi setelah transputer hilang dari pasar karena tidak diproduksi lagi [2], pemrograman CSP tidak dapat lagi dilakukan.

Berdasarkan keinginan ini berbagai macam pustaka sudah dibuat untuk pemgrograman CSP ini dalam bahasa pemrograman populer seperti Java, C dan C++. Beberapa pustaka itu adalah JCSP [6] dan CCSP [7] dari University of Kent, UK; CTJ, CTC dan CTCPP dari Control Engineering group University of Twente, Belanda.

Catatan: Diskusi untuk makalah ini diterima sebelum tanggal 1 Desember 2004. Diskusi yang layak muat akan diterbitkan pada Jurnal Teknik Elektro volume 5, nomor 1, Maret 2005.

2. *Communicating Sequential Process* (CSP)

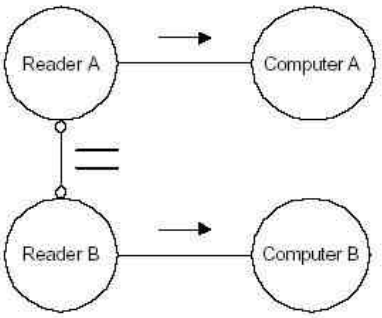
CSP adalah sebuah notasi matematika yang dapat dipergunakan untuk menganalisa sebuah program pada *concurrent system* [1]. CSP pertama kali dicetuskan oleh C. A. R. Hoare pada tahun 1985 yang saat itu mengajar di Oxford University.

CSP menggunakan notasi matematika untuk menggambarkan konsep komunikasi menggunakan pernyataan aljabar yang dapat dipergunakan untuk menganalisa, membuktikan dan menghilangkan keadaan yang tidak diinginkan. Hal-hal itu misalnya: *deadlock*, *livelock* dan *starvation*.

Dalam CSP, concurrent *software* dapat dinyatakan dalam proses (*process*), even (*event*) dan komposisi (*composition*). Kanal (*channel*) dipergunakan untuk komunikasi antar proses. Ketika proses berkomunikasi, itu terjadi dalam suatu even. Komposisi di sini mengacu pada bagaimana proses-proses yang ada berinteraksi satu sama lain. Operator komposisi ini adalah PAR (*parallel*), ALT (*alternative*) dan SEQ (*sequential*). *Programmer* dapat mengkomposisikan proses-proses ini dan melakukan analisa terhadap komposisi tersebut. SEQ merupakan operator yang selama ini dipergunakan dalam pemrograman. ALT memberikan suatu *alternative* untuk memilih satu proses dari daftar proses yang ada untuk dijalankan. Sedang PAR memperbolehkan proses berjalan secara bersamaan.

CSP menggunakan dua macam diagram, yaitu diagram komposisi dan diagram komunikasi. Diagram komposisi menunjukkan bagaimana proses-proses yang ada dikomposisikan menggunakan operator PAR, ALT dan SEQ. Gambar 1 menunjukkan diagram komposisi dari CSP secara sederhana. Diagram komunikasi menunjukkan bagaimana proses-proses tersebut berkomunikasi; kanal mana yang dipergunakan dan jenis *link driver* yang dipakai. Gambar 2 dan 3 merupakan diagram komunikasi dua proses.

Dalam menggunakan CSP, biasanya dinyatakan dalam bentuk gambar atau notasi matematika. Dalam gambar, misalnya seperti terlihat pada gambar 1. Pada gambar ini, proses *Reader* A dan *Reader* B berjalan secara *parallel* (ditandai dengan dua garis sejajar di antara kedua proses).



Gambar 1. Diagram Komposisi pada CSP [5]

CSP saat ini terus dikembangkan, misalnya dengan adanya konsep *timed*-CSP. Selama ini konsep CSP biasa tidak memasukkan waktu dalam analisanya. Padahal dalam pemrograman masalah waktu ini terkadang menjadi sangat penting, terutama dalam bidang system kendali. Versi elektronik dari buku *Communicating Sequential Process* yang ditulis oleh Hoare dapat diambil secara gratis di http://www.usingcsp.com.

3. Pustaka *Communication Thread (CT)*

Jika sebuah program dapat dianalisa dengan menggunakan CSP, maka akan lebih mudah pula jika program tersebut dapat ditulis dalam format CSP. Oleh karena itu dibuatlah sebuah bahasa pemrograman yang diberi nama Occam. Hasil kompilasinya dapat dimasukkan ke dalam sebuah prosesor yang diberi nama Transputer. Berbagai macam aplikasi telah dibuat dengan menggunakan Occam dan Transputer.

Namun, Occam tidak selamanya ada. Setelah Transputer tidak diproduksi lagi [2], tidak ada bahasa pemrograman lain yang memiliki *hardware* pendukung. Oleh karena itu, timbul keinginan untuk mengimplementasikan gaya pemrogramam CSP ini dalam bahasa pemrograman popular seperti Java, C, C++, dll [3,4].

Pustaka CT yang dikembangkan oleh Hilderink dari University of Twente, Belanda mengimplementasikan semua operator komposisi yang terdapat dalam CSP. Bagian-bagian program dibuat sebagai proses dan kanal dipergunakan untuk komunikasi antar proses. Hilderink mengembangkan pustaka ini untuk Java (CTJ), C (CTC) dan C++ (CTCPP).

Dengan menggunakan pustaka ini, occam programming dapat lagi dilakukan dengan mudah dalam Java. Selain operator baku yang

terdapat dalam CSP, dalam pustaka ini juga diberikan tambahan operator seperti PRIALT dan PRIPAR. Operator PRIALT akan memilih satu proses dari proses-proses yang siap dijalankan berdasarkan prioritas tertinggi. Pada PRIPAR, prinsipnya sama dengan PAR, hanya jika terdapat kekurangan resources, proses akan dijalankan berdasarkan prioritasnya.

Dalam pustaka yang dikembangkan oleh Hilderink, dibuat suatu pemisahan antara bagian yang *hardware independent* dan *hardware dependent.* Hal ini membuat perpindahan suatu program ke *platform* yang lain menjadi lebih mudah. [3] menjelaskan dengan lebih detil tentang hal ini. CTJ dapat diambil secara gratis di http://www.ce.utwente.nl/javapp.

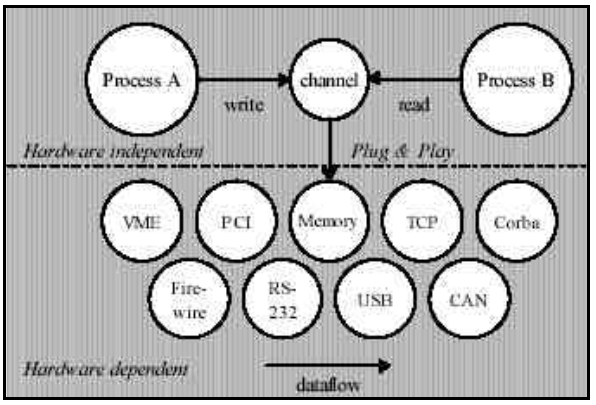
4. *Plug and Play*

Dalam konsep *hardware independent* dan *hardware dependent,* proses-proses yang berkomunikasi melalui channel tidak pernah peduli dengan media yang akan dipakai oleh channel untuk mengirim maupun menerima informasi. Media ini bisa berupa memory biasa maupun suatu communication *link* antar node seperti RS232, RS485, TCP/IP, fieldbus (CAN, profibus, modbus, dll), dll. Gambar 2 meng-ilustrasikan hal ini. Proses A dan B yang saling berkomunikasi hanya tahu kanal tempat mereka berkomunikasi, tetapi bagaimana kanal ini menyampaikan informasi tidak dipedulikan. Di dalam satu kanal hanya terdapat satu *link driver.* Oleh karena itu, kanal hanya dapat meng-hubungkan dua proses untuk berkomunikasi.

Hardware dependent yang pertama disebut context switch. Pembuatan context switch dilakukan berdasarkan arsitektur prosesor yang dipilih. Sehingga untuk satu jenis prosesor hanya memiliki satu context switch. Context switch bertugas untuk melakukan manajemen thread pada prosesor.

Bagian yang *hardware dependent* yang kedua disebut sebagai *link driver* yang akan mengenkapsulasi semua proses yang terkait dengan operasi yang bersifat khusus (*hardware* related operation). Jika *context switch* dibuat berdasarkan arsitektur prosesor, *link driver* dibuat berdasarkan *peripheral* yang ada pada prosesor tersebut. *Peripheral* tersebut bisa berupa fieldbus seperti Control Area Network (CAN), profibus, TCP/IP, dll. atau *hardware* pendukung seperti ADC, DAC, *timer,* dll. *Link driver* dapat juga berupa lokasi memori yang dipergunakan bersama-sama. Ini disebut sebagai *memory link driver*.

Jadi dalam penggunaan *link driver* dalam kanal, *programmer* tinggal melakukan proses *plug and play*. Sebab proses yang menggunakan kanal tidak peduli dengan jenis *link driver* yang dipergunakan.



Gambar 2. *Link Driver*, Kanal dan Proses yang Berkomunikasi [3]

Jika pada sebuah system menggunakan RS232 untuk komunikasi antar proses dan ingin menggantinya dengan RS485, *programmer* cukup melakukan *plug and play* seperti yang digambarkan pada gambar 2. Proses tidak akan mempedulikan hal ini karena mereka berfokus pada kanal tempat mereka berkomunikasi.

5. Aplikasi

Pustaka CT ini dapat diaplikasikan pada embedded system yang biasanya menggunakan berbagai macam prosesor untuk proses yang sama. Oleh karena itu, *programmer* tinggal membuat satu proses yang *hardware independent* dan menggunakannya pada setiap prosesor dengan mengganti *hardware dependent* sesuai dengan prosesor yang dipergunakan.

Sistem terdistribusi juga dapat memanfaatkan pustaka ini. Biasanya terdapat semacam jaringan yang dipergunakan untuk proses pertukaran data. Jika sebuah system berjalan dengan baik akan dipergunakan lagi dengan fieldbus yang berbeda, maka tidak diperlukan pemrograman ulang secara keseluruhan. *Programmer* cukup meng-ganti bagian yang *hardware dependent.*

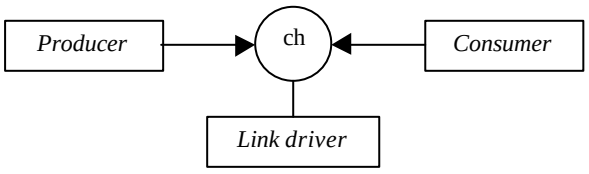
Dalam dunia system kendali dan instrumentasi, peran pustaka ini sangat besar karena banyaknya proses instrumentasi yang melibatkan *peripheral* dari sebuah prosesor. Jika *link driver* untuk *peripheral* sudah dibuat semuanya, maka proses pembuatan program tidak akan terlalu lama. *Programmer* tidak perlu membuat program lagi untuk menggunakan *peripheral* yang ada.

6. Percobaan

Percobaan yang dilakukan pada makalah ini menggunakan CTJ dan dua jenis *link driver*, yaitu memori dan TCP/IP. Untuk melakukan hal ini, diperlukan dua proses yang berkomunikasi melalui sebuah kanal. Proses tersebut adalah *Producer* dan *Consumer*. *Producer* menghasilkan nilai tertentu yang akan dikirimkan ke *Consumer* melalui kanal. Keduanya melakukan proses sinkronisasi, artinya *Producer* tidak akan mengirimkan data berikutnya jika data sebelumnya belum diterima oleh *Consumer*.

Producer dapat mewakili suatu sensor yang menghasilkan besara listrik yang akan dibaca oleh pengendali. Dalam sudut pandang yang lain, *producer* bisa merupakan input dari luar seperti penekan *keypad*, saklar, *limit switch*, dll. Sedangkan *Consumer* dapat mewakili sebuah actuator yang terdapat dalam dunia sistem kendali atau sebuah *server* tempat penyimpanan data dilakukan.

Producer dan *Consumer* dikomposisikan secara *parallel* dan akan dijalankan pada satu PC untuk percobaan dengan menggunakan *memory link driver*. Untuk percobaan dengan TCP/IP, akan dipergunakan dua PC yang terhubung dengan jaringan dengan satu proses pada tiap PC.



Gambar 3. Diagram Komunikasi *Producer* dan *Consumer* yang Berkomunikasi Melalui Kanal

Listing 1 dan 2 menunjukkan program untuk proses *Producer* dan *Consumer* yang ditulis dalam Java.

Listing 1. Proses *Producer*

```
import csp.lang.*;
import csp.lang.Process;
import csp.lang.Integer;

class Producer implements Process{
   ChannelOutput_of_Integer channel;
   Integer object;
   int i;

   public
Producer(ChannelOutput_of_Integer out){
      channel = out;
      object  = new Integer();
   }

   public void run(){

      for (i=0;i<10;i++){
         object.value = 100+i;
         System.out.println("Prod. write
"+object+" to channel");
         channel.write(object);
      }
   }
}
```

Listing 2. Proses *Consumer*

```
import csp.lang.*;
import csp.lang.Process;
import csp.lang.Integer;

class Consumer implements Process{
   ChannelInput_of_Integer channel;
   Integer object;
   int i;

   public
Consumer(ChannelInput_of_Integer in){
      channel = in;
      object  = new Integer();
   }

   public void run(){
      for(i=0;i<10;i++){
         channel.read(object);
         System.out.println("Cons. read
"+object+" from channel\n");
      }
   }
}
```

Memory *link driver* merupakan *link driver* yang sudah terdapat secara default pada setiap pustaka CT. Oleh karena itu, *programmer* tidak perlu membuatnya terlebih dahulu.

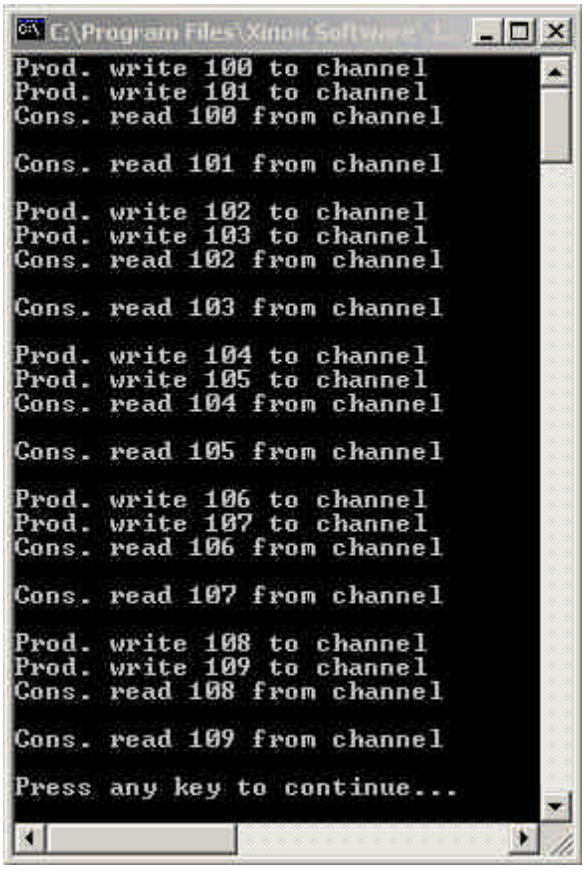
Listing 3 menunjukkan main program yang mengkomposisikan *Producer* dan *Consumer* secara parallel yang menggunakan memory sebagai media dalam channel.

Listing 3. Main program

```
import java.io.*;
import csp.lang.*;
import csp.lang.Process;
import csp.lang.Integer;

public class Main{
   public static void main(String[]
args){
      new Main();
```

```
   }
      public Main()  {
      Channel_of_Integer channel = new
Channel_of_Integer();

      Process par = new Parallel(new
Process[]{
         new Producer(channel),
         new Consumer(channel)
      });

      par.run();
   }
}
```

Hasil yang didapat dapat dilihat pada gambar 4 yang merupakan cuplikan dari layar monitor saat program dijalankan. Jika urutan *Producer* dan *Consumer* pada *listing* 3 dibalik, hasil yang didapat tetap sama. Terlihat bahwa *Producer* sepertinya menulis data tanpa menunggu acknowledge dari *Consumer*. Hal ini karena dalam channel terdapat buffer yang memungkinkan terjadinya penulisan data ke channel sebelum data sebelumnya dibaca terlebih dahulu.



Gambar 4. Layar Monitor Saat Channel dengan Memory Dijalankan.

Berjalannya proses *Producer* dan *Consumer* pada satu PC merupakan bagian dari manajemen *thread* yang dilakukan oleh kernel dalam pustaka CT. Proses yang terhenti karena sesuatu hal, misalnya menunggu data untuk dibaca, akan dikeluarkan dari running thread dan statusnya dimasukkan ke dalam *stack*. Proses yang siap segera dimasukkan ke dalam *dispatcher* untuk dijalankan.

Hal inilah yang terjadi pada *Producer* dan *Consumer*. Secara default, *Consumer* tidak akan bisa berjalan karena harus menunggu data siap terlebih dahulu. Jadi walaupun *Consumer* dijalankan terlebih dahulu, proses ini akan terhenti dan dikeluarkan dari dispatcher. Karena ada proses lain yang sudah siap, maka proses ini dimasukkan ke dalam dispatcher untuk dijalankan. *Producer* tidak dapat selamanya berdiam dalam dispatcher karena proses ini juga akan terhenti saat menunggu acknowledge dari *Consumer*. Demikianlah *Producer* dan *Consumer* secara bergantian dijalankan oleh PC.

Percobaan kedua adalah menjalankan *Producer* dan *Consumer* di dua komputer yang terpisah yang dihubungkan dengan jaringan TCP/IP. Sehingga terdapat satu main program untuk masing-masing komputer. *Listing* 4 dan 5 menunjukkan main program untuk *Producer* dan *Consumer*.

Listing 4. Main program untuk *Producer*

```
import csp.lang.*;
import csp.lang.Process;
import csp.lang.Integer;
import csp.io.linkdrivers.*;

public class MainProd{
   public static void main(String[]
args){
      new MainProd();
   }

   public MainProd(){
      // create channel object

   java.lang.System.out.println("Produce
r makes a channel");
      final Channel_of_Integer channel =
new Channel_of_Integer(new

   TCPIP("130.89.20.205",1701,TCPIP.REAL
TIME));

   java.lang.System.out.println("Channel
made");
      // create parallel construct with a
list of processes
      Process par = new Parallel(new
Process[] {new Producer(channel)});
      // run parallel composition
      par.run();
```

```
   }
}
```

Listing 5. Main program untuk *Consumer*

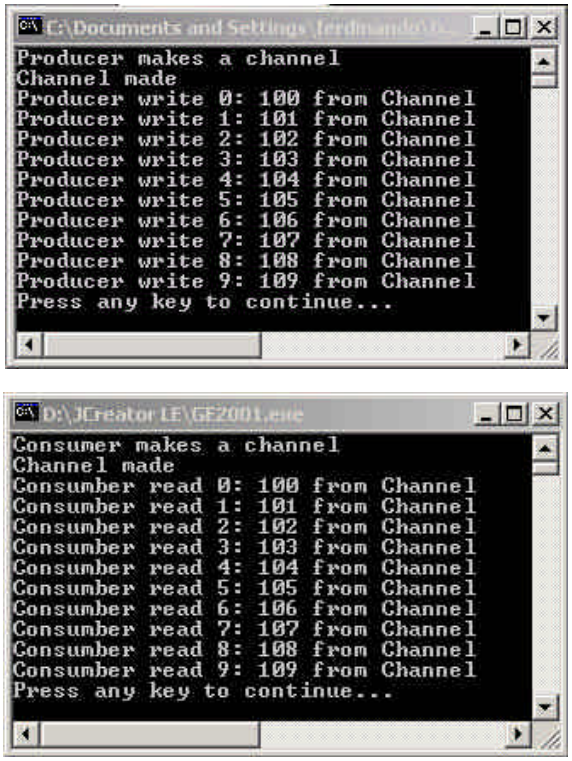
```
import csp.lang.*;
import csp.lang.Process;
import csp.lang.Integer;
import csp.io.linkdrivers.*;

public class MainCons{
   public static void main(String[]
args) {
      new MainCons();
   }

   public MainCons(){
      // create channel object

   java.lang.System.out.println("Consume
r makes a channel");
      final Channel_of_Integer channel =
new Channel_of_Integer(new
         TCPIP(1701,TCPIP.REALTIME));

   java.lang.System.out.println("Channel
made");
      // create parallel construct with a
list of processes
      Process par = new Parallel(new
Process[] {new Consumer(channel)});
      // run parallel composition
      par.run();
   }
}
```

Dalam percobaan ini, pada tiap PC hanya terdapat satu proses, tetapi selalu ada proses *idle* yang secara otomatis ditambahkan oleh pustaka ini. Sehingga jika sebuah proses tidak dapat berjalan lagi, maka proses *idle* ini akan dijalankan terus.

Dalam bagian ini, *Producer* di komputer pertama menulis data di channel yang menggunakan TCP/IP dengan IP *address* komputer tempat dijalankannya *Consumer*. Pada percobaan ini, saat *Producer* harus menunggu proses *idle* akan dimasukkan ke dalam dispatcher menggantikan proses *Producer*. Hal yang sama juga terjadi pada PC tempat proses *Consumer* dijalankan.

Dalam percobaan ini, dipergunakan *Producer* dan *Consumer* yang sama seperti tertulis pada *listing* 1 dan 2. Gambar 5 menunjukkan tampilan layar pada komputer tempat *Producer* dan *Consumer* dijalankan.

Percobaan ini dapat dilakukan dengan menggunakan media lain seperti RS232 maupun RS485. Tetapi *link driver* untuk kedua media komunikasi tersebut harus dibuat terlebih dahulu. Kemudian *programmer* cukup membuat main program yang baru seperti pada *listing* 4 dan 5.



Gambar 5. Tampilan Dilayar Monitor Saat *Producer* dan *Consumer* Berkomunikasi Lewat TCP/IP

7. Diskusi

Pada dua percobaan di atas dipergunakan proses *Producer* dan *Consumer* yang sama. Hal ini dimaksud untuk mendemonstrasikan pemisahan antara *hardware independent* dan *hardware dependent* di dalam pustaka CT ini. *Programmer* tinggal memasukkan *link driver* yang akan dipergunakan di dalam kanal dan proses tidak akan pernah peduli dengan pergantian ini.

Dalam operator PAR yang dijalankan pada satu PC, proses-proses sebenarnya berjalan secara bergantian, bukan bersamaan. Tetapi hal ini tidak menjadi masalah, karena dari sudut pandang aplikasi, proses tetap berjalan secara paralel.

Makalah ini tidak membahas percobaan pada operator ALT dan SEQ. Karena perilaku proses dalam operator ini sudah umum terjadi dalam bahasa pemrograman biasa.

8. Kesimpulan

Pemisahan *hardware independent* dan *hardware dependent* dalam suatu program benar-benar sangat bermanfaat, karena banyak program yang pada prinsipnya sama, tetapi menggunakan perangkat keras yang berbeda. Dengan menggunakan pemisahan ini, migrasi program dari satu platform ke *platform* yang lain dapat dilakukan dengan mudah. Seperti halnya *Producer* dan *Consumer* di atas, kedua percobaan menggunakan *Producer* dan *Consumer* yang sama.

Dalam penggunaannya, *link driver* ini dapat dikumpulkan menjadi semacam dokumentasi yang mengumpulkan semua operasi yang berhubungan dengan perangkat keras (*hardware related operation*), sehingga pengguna dapat langsung mengambil *link driver* yang diperlukan dari dokumentasi tersebut.

Penggunaan pustaka ini untuk aplikasi system kendali dan instrumentasi dapat memperpendek waktu pembuatan. Sebab *link driver* yang sudah dikumpulkan dapat langsung dipergunakan.

Daftar Pustaka

[1] Hoare, C. A. R., *Communication Sequential Process*, electronics edition, 2003

[2] Welch, P.H., M.D. May and P.W. Thompson, Networks, Routers and Transputers: Function, Performance and Application, 1993, pp. ISSN: 90-5199-129-0 [http://www.cs.ukc.ac.uk/pubs/1993/271]

[3] Hilderink, G.H., A.W. Bakker and J. Broenink, A Distributed Real-time Java Based on CSP, *Proc. The Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2000*, March 15-17, 2000, Newport Beach, California, (Ed.), 2000.

[4] Welch, P. H., "Java Threads in the Light of occam/CSP," *presented at Architecture, Language and Patterns for Parallel and Distributed Applications*, WoTUG-21, Amsterdam, 1998.

[5] Visser, P. M., *Control Software Design and Safeguarding with the Support of UML and CT, M.Sc Thesis*, University of Twente, ref: 19CE2002, 2002.

[6] Welch, P. H., Communicating Sequential Process for Java (JCSP), [http://www.cs.kent.ac.uk/projects/ofa/jcsp/]

[7] Moores, J., *Architectures, Languages and Techniques for Concurrent Systems*, volume 57 of *Concurrent Systems Engineering series*, pages 147-168, Amsterdam, the Netherlands, WoTUG, IOS Press. April 1999.