Channel - a Generalized Pipe

Petr Hejda, Jiří Žára
hejda@sgi.felk.cvut.cz, zara@cslab.felk.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University, Praha, Czech Republic

Keywords
MVE, channel, pipe, filter

1. Data Flow Paradigm

Many UNIX users find the concept of pipes and filters useful and flexible. But a pipe has a limitation; there can be only one consumer at the end of a pipe. Introducing a channel (an interprocess communication means connecting one producer and one or more consumers) may bring back to life flexible filters which were displaced by huge "all singing all dancing" programs. We present an attempt at implementing a channel in an efficient way and we have drawn inspiration from the philosophy of Modular Visualization Environments (MVE).

MVEs were the first systems which allowed the building of modular and flexible applications for data visualizing [1]. These systems are mostly based on data flow paradigm [2]. The **data flow** is an abstraction of a signal. It represents continuously changing values of a data object. The changes are made by a filter.

The **filter** (or module) is a basic element of MVE systems. It receives data on its input ports (it reads them from data flow connected to the input), and after a specified operation writes results on output ports (connected output data flows). There can be an arbitrary number of input/output ports. An output port of the filter is connected to other input port(s) via a data flow. Filters without input or output ports can cover a device (e. g. sound card, network card).

Therefore a data flow provides for half duplex loss-free communication among one producer and at least one consumer.

2. Problem Definition

We believe the data flow paradigm can be useful not only for data visualization but also as a part of the operating system. But one of the drawbacks of these systems is the implementation of the data flow. Neither the socket based implementation, nor the shared memory solution is as effective as a UNIX pipe. The goal of our work was to create a channel - a mean for rapid data transfer among processes.

3. Implementation of Channel

UNIX was chosen as an implementation environment because it is the most widely-used multi-user and multitask operational system. Anyway, our solution is portable to any system possessing certain features. In general, shared memory and function calls from the X Window system are sufficient. The basic features of the implementation are:

- **Shared Memory.** There are certain basic ways how to connect two processes: message passing, signals, pipe, socket, and shared memory. The pipes and sockets can connect only one producer and one consumer. Message passing and signals do not allow transfer of large data amounts. Shared memory fulfills both these requirements: ability to transfer large data amounts among one producer and one or more consumers.
- **Synchronization.** When **reading,** consumers must not read more than is actually written. So, they must wait until the amount of data demanded is available. A similar situation occurs for **writing** because the shared memory is of a limited size. A producer must not

rewrite the part of the shared memory all consumers have not yet read. It must wait for the "slowest" consumer.

- **Multiple Buffers.** Shared memory consists of a small number of buffers arranged in a logical ring. The buffers are used exclusively; either for writing into or for reading from, i. e. more than one consumer can read the same buffer simultaneously. The multiple buffer concept speeds up the communication because processes are not synchronized before every single write/read operation.
- **Buffer Flush.** There is a special function a module programmer can use, that causes the producer to set a special flag and go over to a next buffer. So data in the buffer the producer wrote into becomes readable. By default, the implementation of the buffer flush on some of the input channels leads to the output buffers flush. This provides the filter transparency.
- **Data Flow Types.** The data flow is a continuously changing value of a data object. If a basic data object belongs to a certain type (class) of objects, the resulting data flow is of this type as well. The type of the input port can be set by application programmer or automatically after connecting to the relevant channel. This allows for the filtering of different types of data flow by one filter.
- **Graphical User Interface (GUI).** Filters and channels of our implementation build a non-linear computational graph. To easily understand and maintain this graph all filters have its GUI written under the X Windows system [3] [4].

4. Application Area

The channel can be used as an inter-process communication means not only for MVE. All programs consist of some common parts such as: input modules for reading information from a disk or a device, combination modules, database modules, and output modules. These parts running as individual processes can be connected with channels.

From all possible application areas the following two are to be mentioned. The channel was designed to be an abstraction of a signal, so its utilization for multimedia applications is straightforward.

One of the intended future research directions is the use of modular systems as an operating system for virtual reality. In such an environment neither a keyboard nor a mouse-based user interface seem to be effective, but one can easily imagine how to program by connecting flexible modules together as in MVE.

5. Conclusion

We have presented the channel as a loss-free communication means that allows for passing data from one producer to many consumers. Its features and the results of through-put measurements suggest its possible use for rapid data transfer among program modules.

The results displayed above are auspicious. But further testing in real applications is needed. Source codes and technical documentation are available at our address [5].

References

[1] Schröder, F.: "apE™ - The Original Dataflow Visualization Environment," Computer Graphics, May 1995, pp. 5-9

[2] Drakos, N.: "Constructing Object-Oriented Software in Interactive Graphical Programming Environments: An Anthology," http://cbl.leeds.ac.uk/nikos/doc/visual_objects/visual_objects.html

[3] McCormack, J., Asente, P., Swick, R. R.: "X Toolkit Intrinsics - C Language Interface. X Window System. X Version 11, Release 5. First Revision." MIT, Cambridge, MA, 1991

[4] Gettys, J., Scheifler, R. W. : "Xlib - C Language Interface," MIT X Consortium Standard. X Version 11, Release 5. First Revision. MIT, Cambridge, MA.

[5] Technical specification and source codes are available on http://sgi.felk.cvut.cz/~hejda/channel/