DATA EXCHANGE

in a Multics-type environment

by

Michael J. Spier

Data Exchange in a MULTICS-type System 12/7/67

Michael J. Spier

This paper presents a study of a data-exchange facility which, I think, should be part of a multiplexed computer. In order to be able to express myself in familiar terms (and in order to be able to cite commonly known examples) I describe the data exchange facility as if it were associated with Multics.

I wish to emphasize that this paper is, in no way, neither criticism of nor a suggested design-change for Multics.

Purpose

Any superficial study of the MSPM shows that a large number of supervisor modules engage (sometimes unknowingly) in private Interprocess Communication, carried out in dedicated shared data bases according to private conventions.

Such uncontrolled communication--contrary to communication performed by a dedicated module--shows tendencies towards one or more of the following undesirable possibilities:

1. The (uncontrolled) data exchange being arbitrary in nature, it is not inconceivable for a given combination of apparently unrelated procedures to get caught in a deady "you wait for me, I wait for you" trap.

2. It has become evident that most instances of private data exchange at supervisor level have to be made "unquittable" to insure against a possible "infinite instability" of the shared data base.

3. Temporary instability of the shared data base is protected by a convention of mutual exclusion (locking) which enforces sequential processing upon every act of data exchange.

The proposed data exchange facility, as described below, claims to be immune to the above-mentioned problems.

Classes of Communication

The reader might ask, at this point, "and how do you intend to 'data-exchange' the core map, which is a systemwide shared data base"? The answer is that shared data bases seem to be divisible into 3 classes as follows:

Class 1. The shared data base is part of a layer of software in which processes are not yet recognizable as such. This class would most probably include most of ring 0, and very definitely every wired-down data base. Class 1 data bases should be regarded as system-resources which necessitate specialized conventions, access algorithms and even hard-ware considerations. However, by careful study, some of these data bases may be designed in a way so as to insure maximum parallel-processing in their manipulation. These data bases are characterized by the fact that they contain information that is of general interest but which belongs to no one in particular.

Class 2. The shared data base is part of the system supervisor. It is used by system programmers who are known to respect system conventions. Mismanagement of such a data base might cause damage to one or more processes. These data bases are used either as mail boxes in association with the interprocess communication facility (i/o buffers shared between a working process and a device manager process) or as storage areas for a process' private information which it is willing to make known to other processes.

Class 3. The shared data base is defined and manipulated by the Multics user (the guy at the console). No assumption is made as to the respect he has for system conventions, and the amount of harm that he can cause is directly related to the ring in which he executes.

The data exchange module is designed to replace (subject to certain considerations) all communications done via Class-2 data bases. Class-3 type communication may or may not be handled by the data exchange module at the user's discretion.

Implementation

The data exchange facility includes a very large (infinite) storage shared by all processes and known as the "data pool" and two handlers ("terminals") per process to handle incoming and outgoing streams of sequential data. The handlers are known as "receiver" and "transmitter" respectively. Every terminal has associated with it a single unilateral "transmission line" to the data pool. A process never "sticks its finger into the data pool"; rather, it waits for the required information to "pour out" of its receiver.

At receiver terminal level, the process is unconscious of any possible multiprocess race problem, it only knows how to recognize two states associated with the data, namely "data-received" or "data-not-received". This, evidently eliminates the above mentioned problem number 1. As far as problem number 2 is concerned, that a process engaged in interprocess data exchange should be unquittable is a fundamental concept. However, the existance of a data exchange facility localizes the point at which a process must not be quit. In this way, a process has no longer the need (and the right) to declare itself unquittable. The transmission algorithm described in appendix 1 assures maximum parallelism in the data exchange, thus applying maximum optimization to problem number 3.

Application

The data exchange facility should not, of course, be used to transmit whole segments of bulk data between processes. The data exchange handles copies of the transmitted data, and this is feasible only within reasonable limits.

However, communications which are currently handled in the following way:

a. An event signal is sent to a receiving process,

b. the receiver has to identify the associated data via some unique identifier by sequentially accessing a shared data base (possibly recursively),

c. the receiver has to lock (stabilize) the data once it has been located,

d. the receiver has to engage in some sort of housekeeping activity, where it should be remembered that every mutual exclusion (locking) invokes an enormous overhead and sets a jumbo sized event-signalling mechanism into motion (recursively). I firmly believe that such communications, as well as communications that are clearly unilateral (e.g. interprocess calls) as well as others with which I may not yet be familiar, will be speeded up by a large factor merely by having them handled by a system facility.

Appendix I

The data transmission algorithm

The data pool is of infinite size; normally all processes have the same privileges within the data pool, except for certain parts of it in which they are "overprivileged" and which are associated with a process' receiver terminal. The data pool may or may not be in one segment; it should, conceivably be a collection of segments where every process' "overprivileged" section is kept in that process' directory.

Convention 1. An interprocess transmission line connects a process' transmitter terminal to some process' receiver terminal. A transmission line may not "fork" towards more than one receiver terminal. More than one transmission line may share a single receiver terminal.

Transmission lines handle sequential flows of data. The data pool being infinite, a transmitter terminal should not have problems in allocating free storage to itself in a "wait-free" multiprocess race within the data pool.

Once that the data has been copied into the data pool, the transmitting terminal attempts to append it to the target's sequential receiving terminal line. Conflict may arise here in case of parallel processing.

Convention 2. No assumption is made as to the relative speeds of processes. Therefore, we declare the sequence of data appended to a receiver line to be established at the moment of successful linking. As there is no predetermined sequence to be maintained at append time, there is no need for appending processes to mutually exclude one another.

Another conflict may arise when a receiver terminal tries to read unstable data in its incoming line.

Convention 3. For the same reason as given in convention 2, data is considered to be "received" when it is in stable form. Data in any other state is considered as "not received".

This, I think, completes the proof that the data exchange is handled in virtual parallel processing, where sequentialism is kept down to "hardware" requirements. The reader may notice that the data transmission algorithm, unlike locking algorithms, does not include a motion of "wait" (i.e. recursion on a previously unsuccessful attempt).

Appendix 2

Interprocess communication in MULTICS

This appendix states the changes that would have to be made in the MULTICS IPC, were a data exchange module to be adopted.

The adoption of a data exchange would eliminate the need for an event queuing mechanism, which serves only for the transmission of a limited amount of information and causes a very heavy overhead. The notion of "event channel" could disappear, to be replaced by an "event counter" which could transmit only 1 bit of information (which is the process' wakeup switch). It is the process' responsibility to correctly sort out the data that "pours out" of the receiver line and to correctly associate it (through a systemwide convention) with an event signal. Yet, it must be made clear that the data exchange is in no way dependent upon the IPC facility (as it is not associated with a "wait" function) but rather ~~with~~ that the IPC facility may or may not work in conjunction with the data exchange.