RTMP's copy

COMPUTER SYSTEM HAVING APPARATUS FOR ASYNCHRONOUSLY DELIVERING CONTROL ELEMENTS WITH A PIPE INTERFACE

The present invention relates to a computer system, and in particular is directed to a command delivery mechanism for a computer system. Specifically, the invention is directed to asynchronously delivering commands, replies, and status information between a personal computer class machine and a plurality of intelligent subsystems which may have attached devices.

In a data processing or computer system comprised of a host system and at least one subsystem, there is a need to communicate direct and indirect commands to an individual subsystem, usually over an Input Output (I/O) channel.

The direct Command is one which can be directly sent to a subsystem in one operation, is completely and totally self-contained, and totally specifies an operation to be executed by the subsystem. On the other hand, an indirect command is one which requires more data to specify the operation than can be provided within one channel I/O cycle. The indirect command is itself a pointer to a block of system memory which contains all the parameters necessary to totally specify an operation to be performed by the subsystem. This block of memory is often referred to as a "control block".

In a typical prior art computer system, the interface bus overhead involved with the transfer of control blocks limited the throughput capacity of subsystems and their respective I/O. The limitation is caused by the inability of the processor and the subsystem to form a synchronous connection on the bus due to other competing subsystems or processor tasks. Processor performance was impaired due to the inability of the processor to obtain service from adapters and I/O, and the performance of the adapters and I/O was impaired by the ability of the processor to send and receive

THIS SHEET HAS BEEN REPLACED

data as quickly as the adapters and I/O were capable of processing it. As a result, there was a need to eliminate the bottleneck at the interface bus which allowed neither the processor nor the adapters and their respective I/O to perform as well as their hardware would allow.

An aim of the present invention is provide a computer system in which control information and data can be selectively transferred between a processor and subsystems on an asynchronous or synchronous basis.

In accordance with the present invention, there is now provided a computer system having a bus interface comprising a distributed pipe. The pipe allows a processor to asynchronously send and receive control blocks (for the purposes of this disclosure, the control blocks will hereinafter be referred to as control elements) for direct and indirect commands without requiring intervention by an adapter or I/O device. Likewise, the adapters and I/O can asynchronously send and receive control elements to the pipe without intervention by the processor. In addition, the pipe allows subsystem to subsystem control element transfer independent of processor intervention.

Viewing the present invention from a second aspect, there is now provided a computer system comprising: a first unit type having at least one processor; a second unit type having at least one subsystem, which may have attached devices; an interface bus for interconnecting with each of the unit types; characterised in that the system further comprises: a distributed pipe comprising shared memory distributed among each of the unit types; each of the unit types further comprising: at least one entity; delivery send means for asynchronously sending control elements, addressed to another entity from the entity, to the distributed pipe without interaction with any other entity; and delivery receive means for asynchronously

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

receiving control elements, addressed to the entity, from the distributed pipe without interaction with any other entity.

Viewing the present invention from an third aspect, there is now provided, in or for a computer system, a method for transferring control elements representing requests, replies, or asynchronous notification between units in the system, comprising: interconnecting units, including at least one subsystem, with an interface bus; sharing memory physically distributed between the units as a distributed pipe; sending control elements from a sending unit, representative of requests and replies, and addressed to a receiving unit, to the distributed pipe without interaction with any other unit; and receiving control elements, addressed to a receiving unit, from the distributed pipe without interaction with any other unit; wherein the control elements are asynchronously transferred from one of the units to another of the units through the distributed pipe.
The interconnecting units preferably include at least one processor.

Preferably, the control elements can be written to and read from the distributed pipe as a circular queue. In another preferred example, the control elements are written to and read from the distributed pipe as a FIFO queue.

A preferred method of the present invention includes the step of signalling the receiving means with the sending means when a control element is placed on the distributed pipe. Preferably, the step of signalling the delivery receive means with the delivery send means to examine the state of the distributed pipe at predetermined time intervals. The step of using a management entity in each the unit to control interaction of the unit with the distributed pipe is also preferably included..

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

The control elements are preferably configured with a fixed length portion and a variable length portion, the fixed length portion further comprising a type field for describing the type of information in the control element, a length field containing the length of the control element, a source field describing the source of the control element, a destination field describing the destination of the control element, and a correlation field for correlating requests with replies. In a preferred method of the present invention, there is accordingly provided the further step of the transferring data in the variable length portion. There may also be provided the further steps of inserting a command for immediate execution by an attached device in the variable length portion; and inserting an immediate command identifier in the type field; whereby the entity sending the immediate command can encapsulate commands within a control element and pass the immediate command through the distributed pipe. There may further be provided the step of using at least one buffer storage, in the unit containing the entity sending the control element, for holding data external to the distributed pipe, and further identifying the location of data in the buffer with field in the control element.

A preferred method of the present invention includes the additional step of inserting the address of a buffer location containing a command for immediate execution by an attached device in the variable length portion; and inserting an immediate command identifier in the type field; whereby the entity sending the immediate command can encapsulate commands within a control element and pass the immediate command through the distributed pipe.

Preferably, a plurality of requests are concatenated in an execution list by; configuring the variable length field to indicate the length of the execute list, indicating the address of the execute list, and indicating the number of times the execute list is

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

to be repeated; and placing an execute list identifier in the type field of the control element.

The sending and receiving units are preferably synchronised during execution of the execute list by manipulating notify and wait bits in the type field by the sending and receiving units to indicate to each of the sending and receiving units the status of the other unit.

A preferred method of the present invention includes the step of using the delivery send means to detect when the distributed pipe is full. Another preferred method includes the step of using the delivery send means to detect when available space in the distributed pipe is insufficient to hold another control element. The latter method preferably comprises the additional step of placing a wrap element at the end of the distributed pipe space when the distributed pipe does not have available space for another control element, the wrap element indicating to the delivery receive means the end of available pipe space; whereby the delivery receive means resets to the start of the pipe space.

A preferred method of the present invention comprises the step detecting when the distributed pipe changes from the empty state to the not empty state. Preferably, the step of using the delivery receive means to detect when the distributed pipe is empty is included.

A preferred example of the present invention uses a plurality of distributed pipes. Accordingly, a preferred method of the present invention includes the additional step of using a pair of distributed pipes for each unit pair, using the first distributed pipe for control elements sent from a first unit of the unit pair to a second unit of the unit pair, and using a second distributed pipe for control elements sent from the second unit of the unit pair to

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

the first unit of the unit pair; whereby each of the unit pairs operate in full duplex mode.

In an example of the present invention, there is provided a computer system in which control information and data can be transferred between a processor and an attached subsystem independent of the availability of the attached subsystem. Similarly, the control information and data can also be transferred between the subsystem and the processor independent of the availability of the processor. The control information and data can be asynchronously transferred between the processor and attached subsystems in full duplex mode. Furthermore, the control information and data can be transferred between subsystems of a computer system without intervention of the processor. Also, the control information and data can be transferred between a processor and a multiplicity of different subsystems on a common interface. The computer system of the present invention thus enables control information and data to be transferred between a processor and attached subsystems in an improved manner.

In a preferred example of the present invention to be described later, there is provided a computer system having a distributed shared memory, which is located in the processor and the subsystems, to form a distributed pipe organised as a FIFO circular queue. Control elements are asynchronously placed on the pipe by the processor and by the subsystems. Each control element contains a fixed field carrying self describing information including the source and destination. A variable field follows the fixed field, and may contain data or a pointer to a buffer external to the pipe which can contain data or a list of commands to be executed (i.e., indirect commands). The fixed field also contains control bits which allow the subsystem and the processor to synchronously execute a repetitive list of control elements when necessary. The asynchronous sending and receiving of control elements to and from

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

the pipe allows the processor and the subsystems to be free running, thereby avoiding interrupt delays. In addition, a common interface is achieved by the control element structure which allows different subsystem types to use the same pipe.

A preferred embodiment of the present invention will now be described with reference to the accompanying drawings in which:

Figure 1 is a simplified diagram of the invention using a single sending unit and a single receiving unit.

Figure 2 is a overview diagram of the signalling mechanism used to notify units of the system.

Figure 3 is a block diagram of the state information used by the signalling mechanism in figure 2.

Figure 4 is a diagram of the interrupt logic used by the system.

Figure 5 is a diagram of the invention using multiple subsystems and entities.

Figure 6 is another diagram of the invention using multiple subsystems and entities, which also shows the delivery send mechanism and the delivery receive mechanism.

Figure 7 is a diagram of a control element.

Figure 8 is a diagram of an immediate SCB command.

Figure 9 is a diagram of a base SCB.

Figure 10 is a diagram of an execute list.

This Sheet Has Been Replaced

Figure 11 is a diagram of the control elements used for command chaining.

Figure 12 is a diagram showing the format of a read list request.

Figure 13 is a diagram showing the format of control blocks used for indirect data chaining.

Figure 14 is a diagram showing the location of the notify and wait bits.

By way of general overview, the invention allows a processor and a subsystem to asynchronously transfer data and control information and thereby avoid interrupt delays which degrade performance of the processor and the subsystem. In addition, the data and control information can also be asynchronously transferred between subsystems without processor intervention. The microprocessor system which includes a processor unit with system memory and a separate buffer memory, one or more subsystem adapter units with memory, optional I/O devices which may attach to the adapters, and a bus interface (which can interchangeably be called a micro channel). The memory in the processor and the memory in the adapters are used by the system as a shared memory which is configured as a distributed FIFO circular queue (a pipe). Unit to unit asynchronous communication is accomplished by placing control elements on the pipe which represent requests, replies, and status information. The units send and receive control elements independent of the other units which allows free flowing asynchronous delivery of control information and data between units. The shared memory can be organised as pipe pairs between each pair of units to allow full duplex operation by using one pipe for outbound control elements and the other pipe for inbound control elements. The control elements have standard fixed header fields

with variable fields following the fixed header. The fixed header allows a common interface protocol to be used by different hardware adapters. The combination of the pipe and the common interface protocol allows many different types of hardware adapters to asynchronously communicate, resulting in higher overall throughput due to lower interrupt overhead.

An important feature of the invention is the use of a shared memory to hold the control information. The shared memory used to hold the control information is physically distributed between the processor and the subsystems and will hereafter be interchangeably referred to as a distributed pipe or a pipe. A unit of the system, such as a processor or subsystem adapter, can send self-describing control elements to the distributed pipe for later retrieval by another unit. The control elements carry commands, data, or addresses of data stored in a buffer which is external to the pipe. The unit of the system which is intended to receive the control element will retrieve it from the pipe as soon as it can, but does not require the sending unit to wait for the receiving unit to act. The sending unit can, therefore, place the control element in the distributed pipe and proceed to other work rather than waiting for a response from the other unit.

Figure 1, used for illustrative purposes, is a simplified diagram of the invention which uses a single sending and receiving unit and a single pipe. In addition, the sending unit has a single client entity, which can be an application program or the like, and a single server entity, which, for example, may be an I/O device attached to a subsystem. It should be understood by those skilled in the art that, in practice, multiple sending and receiving units would be used, as well as multiple pipes, clients and servers. Figure 1 shows the general flow of data and control information in the invention. The sending and receiving units 102, 108 comprise a unit pair which may be a processor unit and a subsystem unit, or

both may be subsystem units. A sending unit 102, will request that a receiving unit 108 perform some work by placing a control element 104 into shared memory 106 (the distributed pipe). Commands and control information are contained in the control element 104. If data is to be transferred from the sending unit 102 to the receiving unit 108, it can be transferred in one of two ways. First, it can be directly transferred within a variable length field of the control element 104. The control element, including the placement of data within the control element, will be discussed in more detail below. Second, it can be placed in the buffer 110 by the sending unit 102. The sending unit 102 will then place the address and length of the data contained by the buffer 110 into the control element 104 prior to sending the control element 104 to the distributed pipe 106. It should be understood that the actual movement of data into and out of the buffer, as well as the pipe, can be accomplished by a number of conventional methods, well known in the art, such as through the use of DMA logic. As a general rule, it is preferable to use the external buffer 110 to transfer data since the distributed pipe 106 has a finite size and could be rapidly filled by the large data transfers used in many applications. At this point, the sending unit 102 does not have to wait for the receiving unit 108 to respond. The sending unit 102 can proceed with other tasks including the placement of other work requests in the distributed pipe 106 in the form of control elements 104.

The receiving unit 108 will retrieve control elements 104 from the pipe 106, execute the work request contained therein, and reply to the sending unit 102. If the control element 104 addresses data in the buffer 110, the receiving unit 108 will read or write the data from the buffer 110. When the work request is complete, the receiving unit will reply to the sending unit 102 by placing a control element 116 on the pipe 112. (Control elements 104 and 116 are functionally identical, as are pipes 106 and 112. They are

identified separately to denote that the pipe is physically located in different units of the system. In addition, the preferred embodiment uses logically separate pipes for sending and receiving control elements.) As was the case with the sending unit 102, the receiving unit 108 does not wait for a response from the sending unit 102. It can proceed immediately to other work, including the retrieval of more control elements 104 from the pipe 106.

When the sending unit retrieves the control element 116 from the pipe 112 which contains the reply to the sending unit's 102 request, it can take the appropriate action based on the reply.

The logical steps discussed above are supported by the delivery support 118 (which includes a delivery send mechanism and a delivery receive mechanism) and the bus 114. The delivery support mechanism in the preferred embodiment is state logic well known in the art. Control information and data are physically moved to and from the sending and receiving units to the buffer 110 and the pipe 106, 112 by way of the bus 114. Further, the delivery support mechanism 118 includes the enqueue/dequeue logic (not shown) to control placement of control elements on the FIFO queue in pipe 106, 112.

Figures 2, 3, and 4 show the basic mechanism used by the invention for signalling, state logic, and interrupt handling. The basic flow of control elements, shown in figure 2, operates using the send interface 202 and enqueue logic 204 to place control elements into a pipe 220. It then uses dequeue logic 206 and receive interface logic 208 to remove the control element from the pipe 220. The enqueue and dequeue logic use the state information shown in figure 3. There is also logic 210, 214 to recognise certain state changes in the pipe and to signal the other side. The signalling is done by pushing state change surrogate state information to the surrogate enqueue logic 212, 216, then pushing reason codes to the reason area (shown in figure 4) for this unit.

Once the surrogate information and reason information is pushed, a hardware level primitive causes an interrupt (shown in figure 4). The hardware primitive may be either interrupt controller hardware or may be done by writing I/O ports or mapped memory locations which cause hardware interrupts.

Figure 5 shows a structure which is more representative of the invention in actual use than Figure 1. However, as in the discussion of Figure 1, the actual number of entities is not limited to the number of entities shown, but is instead a function of the hardware capacity of the particular system components in question. Figure 5 shows a processor unit 502 which includes multiple client entities 504, 506, 508, 510. Each entity may be a program or some other system task which requires interaction with subsystem units. The subsystems are represented in the figure as they would typically appear in practice, as adapter cards 512, 514. Each adapter 512, 514 in turn has multiple server entities 516, 518, 520, 522, 524, 526 which may be I/O devices such as communication lines, printers, terminals, or the like. Figure 5 also illustrates the use of multiple distributed pipes 528, 530, 532 within a shared memory 534. As shown, pipes for unit pairs can be used by processor/subsystem unit pairs 502/512 and 502/514, or by subsystem/subsystem unit pairs 512/514. Each of the pipes would move control elements 536 on the bus 114, as discussed above in the description of Figure 1.

Figure 6 is a diagram showing the general arrangement of the invention which is designed to show the relationship between the circular pipe and the system. There are a pair of circular pipes 602, 604 which deliver control elements in opposite directions. The basic operation of each is the same. There is state information 606, 608, 610, 612, 614, 616, 618, 620 needed for the enqueueing and dequeueing of control elements (not shown). The state information is distributed between the two units 622, 624. The delivery support 626, 628, 630, 632 is shared between multiple entity to entity pairs

634, 636, 638, 640, 642, 644, 646, 648. Portions of the delivery support communicate with each other over the micro channel 650.

Figure 7 shows the arrangement of fields in a control element. A general discussion of how the invention uses the control elements follows.

Work Orders:

Requests and the replies to those requests, are exchanged between clients (entities) in a system unit or adapter, and servers (entities) in an adapter or system unit using the delivery services of the SCB architecture. These requests, as well as their corresponding replies, are referred to as work orders. Each work order consists of a control element used to describe the unit of work being requested, and one or more control elements used to indicate the completion or current state of the requested unit of work. Error conditions or changes in the processing state of a unit of work are reported in error and event control elements respectively. Using this definition, each of the following would be a valid work order:

- a single request element with no reply element;
- a single request element with a single reply element;
- a single request element with multiple reply elements;
- a single request element with an error element;
- a single request element with an event element;
- a single request element with both event and error elements;
- a single request element with event, reply, and error elements.

When request elements are chained together, each request element in the chain is treated as one part of a single work order.

Elements:

Elements, or more precisely control elements, are used to exchange control information and, in some cases, data between a client and a server. They are like an envelope with a see through window; allowing the delivery service to use information in the window to deliver the control elements without having to know or understand what is contained within the remainder of the control element. Typically, a control element contains the following:

- a 16 bit type field
- a 16 bit length field
- a 16 bit source field
- a 16 bit destination field
- a 32 bit correlation field
- a variable length value field

The first five fields are used by the delivery service as well as the client and server. It constitutes the information visible through the window in the envelope. The remaining field, the value field, contains the contents of the envelope. It contains information meaningful only to the client and the server.

The following sections describe in more detail, the format and content of each of these fields as it pertains to the delivery service.

Type Field:

The type field of an element is a 16 bit word used to identify an element and indicate how to interpret the remaining fields within the element. The type field consists of the following sub-elements:

EID element identifier - a (2) bit sub-field used to identify the element type. There are four element types:

1 - request (00) - An element sent by a client to a server asking it to perform a unit of work on its behalf. It requires the transfer of data and control information between a client and the server.

2 - reply (01) - An element sent by a server in response to certain request elements. Not all request elements are answered by replies, only those that request information.

3 - event (10) - An element containing information about the progress of a request, about the side- effects of a previous request or about the status of an entity. Events may be sent by either a server or a client.

4 - error (11) - An element that provides error information about a previous request, or about the health and welfare of a server or delivery agent.

S suppress reply flag - indicates whether or not the client wants the server to return a reply element when the processing of this request element has been completed successfully. When set to one, the reply is suppressed and no reply element will be returned. When set to zero, whether or not a reply is returned is determined by the entity to entity protocol.

C chaining flags - indicate whether or not command chaining is in effect. When chaining is in effect and all elements in the chain represent a single unit of work, to be processed in the order in which they appear. The following table indicates how the chain bits are to be interpreted:

00 - indicates no chaining (first and last)

01 - indicates the start of a command chain (first element)

11 - indicates an element within the chain (intermediate element)

10 - indicates the end of chain (last element)

I indirect flag - indicates whether the value field of the element contains the parameters for the specified function or just the length of, and a pointer to, a data area where the actual parameters are stored. When set to one, the value field contains a pointer to the parameter value.

N notification flag - indicates whether or not the client is requesting notification by the server when processing of this element begins. When set to one, a notification is to be returned to the client using a notification event element.

W wait flag - indicates whether or not the client is requesting the server to wait before processing this request element, or any other request elements from this client, until it is told to do so with a subsequent resume event element from the client. When set to one, processing is to be suspended.

E expedite flag - identifies a control element that the client wants the server to process as expeditiously as possible, i.e. before any other control elements that may be waiting to be processed. When set to one, the control element is to be expedited.

FC function code - a seven (7) bit sub-field used to identify the function to be performed and indicate how to interpret the contents of the value field.

Length Field

The length field of an element is a 16 bit word used to specify the total length of the element in bytes. This includes 2 bytes for the type field, 2 bytes for the length field, 2 bytes for the source field, 2 bytes for the destination field, 4 bytes for the correlation field, and n bytes for the variable length value field.

Source Field

The source field is a 16 bit word used to identify the originator or source of an element. It is a structured field containing a unit identifier and an entity identifier. For the Micro Channel, the unit id is the bus unit with an allowable range of 0 to 15, and the entity id is one of 256 possible sources. Entity id zero (0) is reserved for management purposes. Together, they provide sufficient information to unambiguously identify a bus unit as well as the entity within the bus unit responsible for originating the element.

Destination Field

The destination a 16 bit word used to identify the target or destination of an element. Like the source field, it to is a structured field containing a unit identifier and an entity identifier For the Micro Channel, the unit id is the bus unit with an allowable range of 0 to 15, and the entity id is one of 256 possible destinations. Entity id zero (0) is reserved for management purposes. Together, they provide sufficient information

to unambiguously identify the bus unit and the entity within the bus unit who is the recipient of the element.

Correlation Field

The correlation field is a 32 bit doubleword used to provide an identifier for associating or correlating a response with a previous request. The format and content of the correlation identifier is determined by the entity who is the originator of the request. It may be a sequence number, the address of a control block or request packet, or the address of a data buffer. The correlation identifier is returned to the originator of the request in a reply, error, or event element.

Value Field

The value field of an element is a variable length field used to hold additional arguments, parameters, or data required by the operation identified by the function code in the type field. The type, length, and format of this information will vary depending on the function specified.

Function Codes

Within an element, a function code is used to identify the operation to be performed and to indicate how the contents of the value field should be interpreted. The High order bit of the function code is used to distinguish between common usage function codes and implementation defined function codes. When bit 6 is set to one (1), it indicates that the remaining bits (0 - 5) contain a non- architected function code. When set to zero (0), bit 6 indicates that the remaining bits (0 - 5) contain one of the following common usage function codes:

- Immediate SCB Command (FC=1)
- Initialise (FC=2)
- Base SCB Command (FC=3)
- Read (FC=4)
- Read List (FC=5)
- Read Immediate (FC=6)
- Write (FC=7)
- Write List (FC=8)
- Write Immediate (FC=9)
- Execute List (FC=10)
- Mark (FC=11)
- Cancel (FC=12)
- Reset (FC=13)
- Read Configuration (FC=14)
- Diagnose (FC=15)
- Suspend (FC=16)
- Resume (FC=17)
- Notify (FC=18)
- Inform (FC=19)
- Wrap (FC=31)

Function codes 0, 20 through 30, and 32 through 63 are reserved and should be treated as an error condition if found in a control element.

The suspend, resume, notify, inform, and wrap function codes are used exclusively with the event element. All other function codes are used with request, reply, and error elements.

The following sections describe in detail the format, content and usage of the above functions in the various element types of the extended SCB architecture support.

Immediate SCB

The immediate SCB elements provide a path for certain forms of adapter migration. They allow the delivery service to be used with the existing SCB's. The format and content of this element is shown in figure 8.

Initialise

The Initialise function is found in request, reply, and error elements. In a request element, it invokes the initialisation function of the server. The source and destination, a correlation identifier, and the value field containing any additional parameters required by the server initialisation function are passed to the server in the request element. The initialisation parameters passed in the value field of the request element are server dependent and will need to be defined on a server by server basis.

If the indirect flag is set in the type field of the initialisation request element, the value field does not contain the actual initialisation parameters, but instead contains the location and the length of a data area where the actual parameters themselves are stored.

A server reports the successful completion of an initialisation request by returning a reply element to the requestor. The source and destination, the correlation identifier from the initialisation request element, and any return values are placed in the value field of the reply element.

The return parameters passed in the value field of the reply element are server dependent and will need to be defined on a server by server basis.

A client may choose to suppress the returning of a reply element by by setting the suppress flag in the Type Field of the initialisation request element.

If unsuccessful, the source and destination, the correlation identifier from the initialisation request element, and status information identifying the cause of the failure are returned to the requestor in the value field of an error element.

Base SCB

The base SCB elements provide a path for certain forms of adapter migration. They allow the delivery service to be used with the existing SCB's. The format and content of these elements is shown in figure 9.

Read

The Read function is found in request, reply, and error elements. In a request element, it is used to setup and initiate the transfer of data and control information from the server to a client. The source and destination, a correlation identifier, and the value field containing the parameters required by the server read function are passed to the server in the request element.

It should be noted that the parameters contained in the value field of a request element are not limited to those identified above. Additional parameters may be supplied to meet the requirements of each client/server relationship.

If the indirect flag is set in the type field of the read request element, the value field does not contain the byte count, data address and optional parameters, but instead contains the location and the length of a data area where they are stored. When

the server has successfully completed the transfer, it returns a reply element with the source and destination, the correlation identifier from the read request element, and the actual number of bytes transferred in the value field of the reply element.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the read request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the read request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element.

Read List

The Read List function is found in request, reply, and error elements. It is used to setup and initiate the transfer of data and control information from a server into several possibly non-contiguous areas of memory. This is often referred to as data chaining. The source and destination, a correlation identifier, and the value field containing the list of byte count/data address pairs required by the read list function are passed to the server in the request element.

If the indirect flag is set in the type field of the read list request element, the value field does not contain the list of byte counts and data address, but instead contains the location and the length of a area where the actual list is stored.

When the server has successfully completed the transfer, it returns a reply element with the source and destination, the correlation identifier from the read list request element, and

residual byte count in the value field of the reply element. The format and content of the read list reply element is the same as the read reply.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the read list request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the read list request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element. The format and content of the read list error element is the same as the read error.

Read Immediate

The Read Immediate function is found in request, reply, and error elements. It is used to request the transfer of data and/or control information from a server to a client. It differs from the read request in that the data is to be returned in the value field of the reply element where it will be immediately available to the client. The source and destination, a correlation identifier, and the value field containing the parameters required by the read immediate function, are all passed to the server in the request element.

The amount of data that can be returned using the read immediate request element is configuration dependent. That is, it is directly related to the size of the delivery queue. Therefore, care should be exercised when using this request element.

When the server has successfully completed the transfer, it returns a reply element with the source and destination, the

correlation identifier from the read immediate request element and, in the value field, the actual number of bytes transferred followed immediately by the data itself.

The read immediate function ignores the suppress flag if set in the Type Field of the read immediate request element since it always returns a reply element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the read immediate request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element.

Write

The Write function is found in request, reply, and error elements. It is used to setup and initiate the transfer of data and control information from a client to a server. The source and destination, a correlation identifier, and the value field containing the amount and location of the data to be transferred by the write function, are all passed to the server in the request element.

It should be noted that the parameters contained in the value field of a request element are not limited to those identified above. Additional parameters may be supplied to meet the requirements of each client/server relationship.

If the indirect flag is set in the type field of the write request element, the value field does not contain the byte count, data address and optional parameters, but instead contains the location and the length of a data area where they are stored.

When the server has successfully completed the transfer, it returns a reply element with the source and destination, the correlation identifier from the write request element, and the actual number of bytes transferred in the value field of the reply element.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the write request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the write request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element.

Write List

The Write List function is found in request, reply, and error elements. It is used to setup an initiate the transfer of data and control information to a server from several possibly non-contiguous areas of memory. This is often referred to as data chaining. The source and destination, a correlation identifier, and the value field containing the list of byte count/data address pairs required by the write list function, are all passed to the server in the request element.

If the indirect flag is set in the type field of the write list request element, the value field does not contain the byte counts and data address, but instead contains the location and the length of an area where the actual list is stored.

When the server has successfully completed the transfer, it returns a reply element with the source and destination, the

correlation identifier from the write list request element, and residual byte count in the value field of the reply element. The format and content of the write list reply element is the same as the write reply.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the write list request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the write list request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element. The format and content of the write list error element is the same as the write error.

Write Immediate

The Write Immediate function is found in request, reply, and error elements. It is used to carry data and/or control information from a client to a server. It differs from the write request in that the data is present in the value field of the request element and is immediately available to the server. The source, destination, and correlation identifier, as well as the data itself, are all passed to the server in the request element.

The format as well as the meaning of the data contained in the value field of the request element are client/server dependent and as such are not defined here.

The amount of data that can be transferred using the write immediate request element is configuration dependent. That is, it

is directly related to the size of the delivery queue. Therefore, care should be exercised when using this request element.

When the server has successfully received the data, it returns a reply element with the source and destination, the correlation identifier from the write immediate request element and, in the value field, a count of the actual number of bytes received.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the write immediate request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the write immediate request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element.

Execute List

Figure 10 shows an example of an execute list. The Execute List function is indicated in the type field of request and reply elements. It is used to setup a repetitive loop for executing a list of request elements pointed to by this element one or more times. The source and destination, a correlation identifier, and the value field containing the location, length, and count required by the execute list function, are all passed to the server in the request element.

A list of requests to be executed may consist of one or more request elements which must be contiguous in memory. Each element in the list contains its own length field so that the start of the next element can easily be determined. The "length of list" field determines where the list ends. On each pass through the list, the

repetition count is decremented by one, and if not zero, the list is executed again. If the initial repetition count is negative or zero the request terminates with no execution.

The request elements that make up the list can contain any of the function codes defined in this section and use any of the optional flags available to them.

When the repetition count goes to zero, the list is terminated and the server returns a reply element with the source, destination, and correlation identifier from the original execute list request element.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the execute list request element.

Should an error occur while processing any element within the list, the list will be terminated. The error element returned, however, will be that of the offending request element.

Mark

The Mark function is used only in a request element to provide a means of marking or indicating a point of possible synchronisation between a client and a server. It performs no other function other than the passing of a correlation identifier to the server.

The power of this function is in the way it can utilised the Notify and Wait flags to inform a client of the current processing state of the server.

The Mark function ignores the indirect, suppress, and expedite flags if set in a mark request element.

Cancel

The Cancel function is used in request, reply, and error elements. In a request element, it is used to request the cancellation of one or more outstanding request elements. The source and destination, a correlation identifier, and the value field containing the cancellation qualifier and an optional list of request element correlation identifiers are all passed to the server in the request element.

The cancellation qualifier bits define which request elements are to be cancelled:

- all outstanding requests - (bit 0 = 1)
- requests that match the source ID - (bit 1 = 1)
- first request matching the cancellation list value(s) - (bit 2 = 1)
- all requests matching the cancellation list values(s) - (bit 3 = 1)

A cancellation list is present whenever bits 2 or 3 of the cancellation qualifier are set to one. If bit 1 is set, then bit 2 or 3 must also be set.

If the indirect flag is set in the type field of the cancel request element, the value field does not contain the cancellation list, but instead contains the location and the length of a data area where the actual list is stored.

When the server has successfully completed the cancel function, it returns a reply element to the requestor. The source and destination, as well as the correlation identifier from the cancel request element are returned in the reply element.

The server will attempt to cancel all requests whose correlation identifiers are in the value field of the cancel request. If for any reason the server is unable to do so, it will return an error element with status and a list of correlation identifiers for those it was able to cancel. An empty list indicates that it was unable to perform the cancel.

However, a cancel request element with the all outstanding elements qualifier will not return an error element with the list of cancelled elements.

Reset

The Reset function is used in request, reply, and error elements. In a request element, it is used to request the server be placed into a known state. The source and destination, a correlation identifier, and the value field containing the desired state, are all passed to the server in the request element.

When the server has successfully completed the reset, it returns a reply element with the source and destination, the correlation identifier from the reset request element to the requestor.

The reset function ignores the suppress and indirect flags if set in a reset request element.

If unsuccessful, an error element containing the source and destination, the correlation identifier from the reset request element, and status information identifying the cause of the failure, are returned to the requestor in the value field of the error element.

Read Configuration

The Read Configuration function is used in request, reply, and error elements. When used in a request, it causes the server to return configuration information to the client. The source and destination, a correlation identifier, and the value field containing the byte count, data address, and optionally the specific configuration information to be returned, are passed to the server in the request element.

It should be noted that the parameters contained in the value field of a request element are not limited to those identified above. Additional parameters may be supplied to meet the requirements of each client/server relationship.

If the indirect flag is set in the type field of the read configuration request element, the value field does not contain the optional parameters, but instead contains the location and the length of a data area where the actual parameters themselves are stored.

A server reports the successful completion of a read configuration request by returning a reply element to the requestor. The source, destination, and correlation identifier from the read configuration request element are returned in the value field of the reply element.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the read configuration request element.

If the server is unable to report back configuration information, an error element with the source and destination, the correlation identifier from the read configuration request element, and the reason are returned to the requestor in the value field of the error element.

Diagnose

The Diagnose function is used in request, reply, and error elements. In a request, it is used to initiate the running of diagnostic routines by a server. The source and destination, a correlation identifier, and the value field containing the specific diagnostic tests to be run as well as any operational parameters, are passed to the server in the request element.

If the indirect flag is set in the type field of the diagnose request element, the value field does not contain the optional parameters, but instead contains the location and the length of a data area where the actual parameters themselves are stored.

A server reports the results of running the diagnostic tests by returning a reply element to the requestor. The source and destination, correlation identifier from the diagnose request element, and the value field containing the test results are returned in the reply element.

A client may choose to suppress the returning of a reply element by setting the suppress flag in the Type Field of the diagnose request element.

If a server is unable to run the requested diagnostic test, a error element with the source and destination, the correlation identifier from the diagnose request element, and the value field containing the reason are returned to the requestor in the error element.

Events

In addition to request, reply, and error elements, the delivery service provides an element for conveying information about the

progress of a request, or about the side-effects of a previous request, or about the state of an entity - the event element. The event element differs from the other elements in that it it reports a change in state of one of the communicating entities (client or server). Because of the impact of these state changes, the peer-entity must be informed of the change as expeditiously as possible. The following sections describe the format, content and usage of the functions codes of an event element.

Suspend

A suspended event is used to notify a client of a change in processing state at the server. A server changes state whenever it receives a request element from a client in which the wait flag of the type field is set to one. The server suspends the processing of the request and all subsequent requests from the specified source. This change of state is communicated to the the client by means of a suspend event element. The source and destination, as well as the correlation identifier of the request element causing the suspension, are returned to the client in the value field of the event element.

Resume

A resume event is used to notify a server of a change in a clients processing state. A client uses this event to notify a server that the client is now in a state where the processing of request elements can now be resumed. The source and destination, as well as the correlation identifier of the original request element causing the suspension, are returned to the server in the value field of the resume event.

A resume event received from a source other than the one responsible for the suspended state or the receipt of a resume for a destination that is not in the suspended state, will be ignored.

Notification

A notification event is used to confirm the receipt of a request element from a client in which the notification flag of the type field is set to one. The server communicates this to the client by means of a notification event element. The source and destination, as well as the correlation identifier of the request element containing the notification flag, are returned to the client in the value field of of the event element.

Inform

An inform event is used to provide a means of conveying information regarding the processing state of either a client or a server from one to other.

Wrap

A wrap event is a management and control initiated event used internally by the delivery system. It is used to synchronise the operations associated with the in-bound or out-bound delivery pipes (queues).

A wrap event element can be sent only by a delivery agent. The wrap event informs the destination delivery agent that the associated delivery queue has wrapped and that the next element should be dequeued from the top of the delivery queue.

Command Chaining

The processing of request elements by a server may take place either synchronously or asynchronously depending on whether or not command chaining is in effect. Normally a server may process requests in any order that makes sense to the server (asynchronously). However, there are also times when a client wishes to have the server process the requests in the order in which they have been submitted (synchronously). Command chaining is used for this purpose. Whenever the command chaining field is set in a request, it means that the processing of the next element in a chain cannot begin until the server and/or device has completed processing the current request element and has returned the appropriate reply to the client. Chaining is indicated by setting the chain indicators in an element to the values BC = Beginning of chain, MC = Middle of chain, and EC = End of chain. This is illustrated in figure 11. Processing of requests then resumes with the next request element in the chain. This action continues until all elements in a chain have been processed, or an error condition is encountered. When an error condition is encountered, an error element is returned and the remaining request elements within the chain are discarded.

Data Chaining

Often it is necessary to gather data from several non-contiguous areas of memory and transfer it to a single destination (a write) or, conversely, to transfer data from a single source and scatter it into several non-contiguous areas of memory (a read). This is commonly referred to as data chaining. The SCB architecture supports two methods of data chaining: direct and indirect.

Direct Data Chaining

In direct data chaining, each non-contiguous data area is described by an entry in a chaining list. The entry contains a byte

count and an address pointer to a data area. After each data area described by an entry is transferred, the next entry in the list is processed. This action continues, until all the entries in the list have been processed, or no more data exits. Direct data chaining is invoked through the use of the Read List and Write List function codes. The format of a read list element for direct data chaining is shown in figure 12.

Indirect Data Chaining

Indirect data chaining is similar to direct data chaining except for the location of the chaining list. The list is not part of the request element. Instead the list is kept separate and is referenced by means of a indirect list pointer in the value field of the request element. Indirect data chaining is invoked by setting the indirect flag in a Read or Write List request element. The format of a read list element for indirect data chaining is shown in figure 13.

Using Notification and Wait

A client can request notification when an element is about to be processed by setting the notification flag in the type field of the appropriate request element. This causes the server to return a event notification element with the source, destination, and correlation identifier of the element containing the notification flag in its value field to the client.

A client can suspend the processing of a set of request elements at a particular point in the list by setting the wait flag in the type field of the appropriate request element prior to submitting it to the Delivery Service This causes the server to suspend further processing of request elements, to return an event notification element with the correlation identifier of the element

containing the suspend flag in its value field, and to wait for a resume event from the originating client before continuing. The original request element containing the wait flag is not processed until the resume event is received from the client. Figure 14 shows the various combinations of usage for notification and wait.

While the invention has been particularly shown and described with reference to preferred embodiments, it will be understood by those skilled in the art that various changes in detail may be made therein without departing from the spirit, scope, and teaching of the invention. For example, there are many queueing techniques known in the art, at least some of which could be substituted for the FIFO circular queue used in the preferred embodiment. Likewise, the format of the control element could be varied. Accordingly, the invention herein disclosed is to be limited only as specified in the following claims.

THIS SHEET HAS BEEN REPLACED

CLAIMS

1. An apparatus within a system for transferring control elements (104,116) representing requests, replies, or asynchronous notification between entities (642,646) located in different units (622,624) of the system, the system comprising:

a first unit type (622), the first unit type (622) further comprising at least one processor;

a second unit type (624), the second unit type (624) further comprising at least one subsystem, which may have attached devices;

an interface bus (650), interconnecting with each of the unit types (622,624);

a distributed pipe (602,604), the distributed pipe (602,604) further comprising shared memory (106,112) distributed among each of the unit types (622,624);

each of the unit types (622,624) further comprising:

at least one entity (642,646);

a delivery send means (630) for sending control elements (104,116), addressed to another entity from the entity, to the distributed pipe (602,604) without interaction with any other entity; and

a delivery receive means (628) for receiving control elements (104,116), addressed to the entity, from the distributed pipe (602,604) without interaction with any other entity;

This Sheet Has Been Replaced

* C L M S *

THIS SHEET HAS BEEN REPLACED

wherein the control elements (104,116) are asynchronously transferred from one of the entities (642,646) to another of the entities (642,646) through the distributed pipe (602,604).

2. An apparatus, as claimed in claim 1, wherein the pipe is a circular queue.

3. An apparatus, as claimed in claim 1, wherein the pipe is a FIFO queue.

4. An apparatus, as in any preceding claim, wherein the entity sending the control element is from one of the unit types (622,624), and the entity receiving the control element is from the other of the unit types (622,624).

5. An apparatus, as claimed in claim 1, wherein the pipe (602,604) contains control elements (104,116) for a plurality of entity pairs.

6. An apparatus, as claimed in claim 1, wherein the delivery send means (630) further includes means to signal the delivery receive means (628) when there are state changes in the distributed pipe (602,604).

7. An apparatus, as claimed in claim 1, wherein the delivery send means (630) further includes means to signal the delivery receive means (628) to examine the state of the distributed pipe (602,604) at predetermined time intervals.

8. An apparatus, as claimed in claim 1, wherein each of the units (622,624) as an entity dedicated to managing its respective unit's interaction with the distributed pipe (602,604).

This Sheet Has Been Replaced



THIS SHEET HAS BEEN REPLACED

9. An apparatus, as claimed in claim 1, wherein each unit further comprises an interface to the distributed pipe (602,604) which uses a common set of push, pull, and interrupt primitives.

10. An apparatus, as claimed in claim 4, wherein the control elements (104,116) further comprise a fixed length portion and a variable length portion, the fixed length portion further comprising a type field for describing the type of information in the control element, a length field containing the length of the control element, a source field describing the source of the control element, a destination field describing the destination of the control element, and a correlation field for correlating requests with replies.

11. An apparatus, as claimed in claim 10, wherein the variable length portion contains data.

12. An apparatus, as claimed in claim 10, wherein:

the variable length portion further comprises a field having a command for immediate execution by an attached device; and

the type field further comprises an immediate command identifier;

whereby the entity sending the immediate command can encapsulate commands within a control element and pass the immediate command through the distributed pipe (602,604).

13. An apparatus, as claimed in claim 10, wherein the unit containing the entity sending the control element further comprises at least one buffer storage for holding data external to the distributed pipe (602,604), and the control elements (104,116)

* C L M S *

This Sheet Has Been Replaced

THIS SHEET HAS BEEN REPLACED

further having fields to identify the location of data in the buffer which is associated with the control element.

14. An apparatus, as claimed in claim 10 wherein the entity sending the control element is in a subsystem, and the entity receiving the control element is in a subsystem.

15. An apparatus, as claimed in claim 15, wherein the unit further comprises at least one buffer storage for holding data external to the distributed pipe (602,604), and the control elements (104,116) further having fields to identify the location of data in the buffer which is associated with the control element.

16. An apparatus, as claimed in claim 13 or claim 15 wherein:

the variable length portion further comprises a field having an address of a buffer location containing a command for immediate execution by an attached device; and

the type field further comprises an immediate command identifier;

whereby the entity sending the immediate command can encapsulate commands within a control element and pass the immediate command through the distributed pipe (602,604).

17. An apparatus, as claimed in claim 10, wherein: the variable length portion further comprises a field indicating the length of an execute list containing a series of requests to an entity, a field indicating the address of the execute list, and a repetition count indicating the number of times the execute list is to be repeated; and

the type field further comprises an execute list identifier.

* C L M S *

This Sheet Has Been Replaced

THIS SHEET HAS BEEN REPLACED

18. An apparatus, as claimed in claim 17, wherein the type field further comprises notify and wait bits for indicating to the receiving entity synchronisation status for the sending entity;

whereby the sending entity is synchronised with the receiving entity.

19. An apparatus, as claimed in claim 1, wherein the delivery send means (630) further comprises means to detect when the distributed pipe (602,604) is full.

20. An apparatus, as claimed in claim 1, wherein the delivery send means (630) further comprises means to detect when available space in the distributed pipe (602,604) is insufficient to hold another control element.

21. An apparatus, as claimed in claim 20 wherein the delivery send means (630) further comprises means to place a wrap element at the end of the distributed pipe space when the distributed pipe (602,604) does not have available space for another control element, the wrap element indicating to the delivery receive means (628) the end of available pipe space;

whereby the delivery receive means (628) resets to the start of the pipe space.

22. An apparatus, as claimed in claim 1, wherein the delivery receive means (628) further comprises means to detect when the distributed pipe (602,604) is empty.

23. An apparatus, as claimed in claim 1, wherein the delivery receive means (628) further comprises means to detect when the distributed pipe (602,604) changes from the empty state to the not empty state.

This Sheet Has Been Replaced

* C L M S *

THIS SHEET HAS BEEN REPLACED

24. An apparatus, as claimed in claim 1, wherein there are a plurality of distributed pipes.

25. An apparatus, as claimed in claim 24, wherein each unit pair has a pair of distributed pipes (602,604) associated with the unit pair, the pair of distributed pipes further comprising a first distributed pipe (602) for control elements sent from a first unit of the unit pair to a second unit of the unit pair, and a second distributed pipe (604) for control elements sent from the second unit of the unit pair to the first unit of the unit pair;

whereby each of the unit pairs operate in full duplex mode.

26. A computer system comprising apparatus as claimed in any preceding claim.

27. In or for a computer system, a method for transferring control elements (104,116) representing requests, replies, or asynchronous notification between units (622,624), comprising:

interconnecting units (622,624) with an interface bus (650);

sharing memory physically distributed between the units (622,624) as a distributed pipe (602,604);

sending control elements from a sending unit, representative of requests and replies, and addressed to a receiving unit, to the distributed pipe (602,604) without interaction with any other unit; and

receiving control elements, addressed to a receiving unit, from the distributed pipe (602,604) without interaction with any other unit;

This Sheet Has Been Replaced

* C L M S *

THIS SHEET HAS BEEN REPLACED

wherein the control elements (104,116) are asynchronously transferred from one of the units to another of the units through the distributed pipe (602,604).

28. A method as claimed in claim 26 wherein the units (622,624) include at least one processor unit.

29. A computer system comprising:

a first unit type having at least one processor;
a second unit type having at least one subsystem, which may have attached devices;

an interface bus (650) for interconnecting with each of the unit types (622,624);

characterised in that the system further comprises:

a distributed pipe (602,604) comprising shared memory (106,112) distributed among each of the unit types (622,624);
each of the unit types (622,624) further comprising:

at least one entity (642,646);

delivery send means (630) for asynchronously sending control elements (104,116), addressed to another entity from the entity, to the distributed pipe (602,604) without interaction with any other entity; and

delivery receive means (628) for asynchronously receiving control elements (104,116), addressed to the entity, from the distributed pipe (602,604) without interaction with any other entity.

This Sheet Has Been Replaced

* C L N S *