OPERATING SYSTEMS FOR RECONFIGURABLE NETWORK ARCHITECTURED SYSTEMS: THE NODE KERNEL

by

DANIEL ALBERTO CANAS, ING., M.S.

MAY 1983 TR-228

OPERATING SYSTEMS FOR RECONFIGURABLE NETWORK ARCHITECTURED SYSTEMS: THE NODE KERNEL

by

DANIEL ALBERTO CANAS, ING., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY
THE UNIVERSITY OF TEXAS AT AUSTIN

May, 1983

ABSTRACT

The Operating System for a Reconfigurable Network Architectured (RNA) system must be capable of virtualizing the execution environment of the task which runs on a Logical Processor. This execution environment of the task is reflected in the virtualization of:

1. Communication.
2. Logical Processors.
3. Address Space.
4. Synchronization.

The varistructure property of RNA's require for the Operating System to support the unique characteristic of intra-task communication. This property will virtualize the architecture upon which a task executes. Once this is accomplished, the Operating System is able to virtualize the support of Logical Processors, Communication and Address Space. The Operating System will virtualize Logical Processors by providing the facility to create jobs and tasks, Communication by

providing the proper inter-task communication interface and Address Space by supporting virtual memory.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

An Operating System implements processes, address spaces, interprocess communication and execution environments of logical devices. This research is concerned with the design and specification of process and interprocess communication on a varistructured reconfigurable network architectured system (RNA).

The concept base mechanism sets for executing functions for conventional static processor structure are not yet fully established. The reason for this is the problem faced in the development of Operating Systems for statically structured multiprocessors and for distributed systems. Previous research into execution functions for concept base mechanism sets have been limited [BAC 81, KAR 77, QUA 78, SUL 77].

A logical computer system (LCS) consists of one or more logical processors executing an instruction stream in

an address space. It also provides for communication between its processors and to other logical computer systems. A logical processor is a computational engine capable of executing instructions against data elements of some fixed precision. The Texas Reconfigurable Array Computer (TRAC) [SEJ 80, PRE 80] instantiation of an RNA allows for equivalent processors to be constructed from different sets of physical resources (the varistructuring concept). For example, a logical processor of 4 bytes of precision can be constructed from 1,2,3 or 4 physical processors. RNA's create logical computer systems from primitive processor and memory units by binding communication paths between pairs or sets of processor/memory components. This binding may be changed during the execution of a process (reconfiguring concept).

A top level schematic representation and a breakdown of the structure of a job for the complete Operating System for an RNA is shown in Figures 1-1 and 1-2. As shown in Figure 1-1 the System Configurator (SC) partitions the system resources into Logical Computer Systems. The Job Monitor (JM) controls the execution steps of an individual job. The Processor Resident Monitor or PRM associated with each physical processor must create the execution environment seen by each physical processor

[BRO 81]. The PRM must be able to bind an almost arbitrary logical structure to a physical structure or, equivalently, to virtualize almost any execution structure with some set of resources.

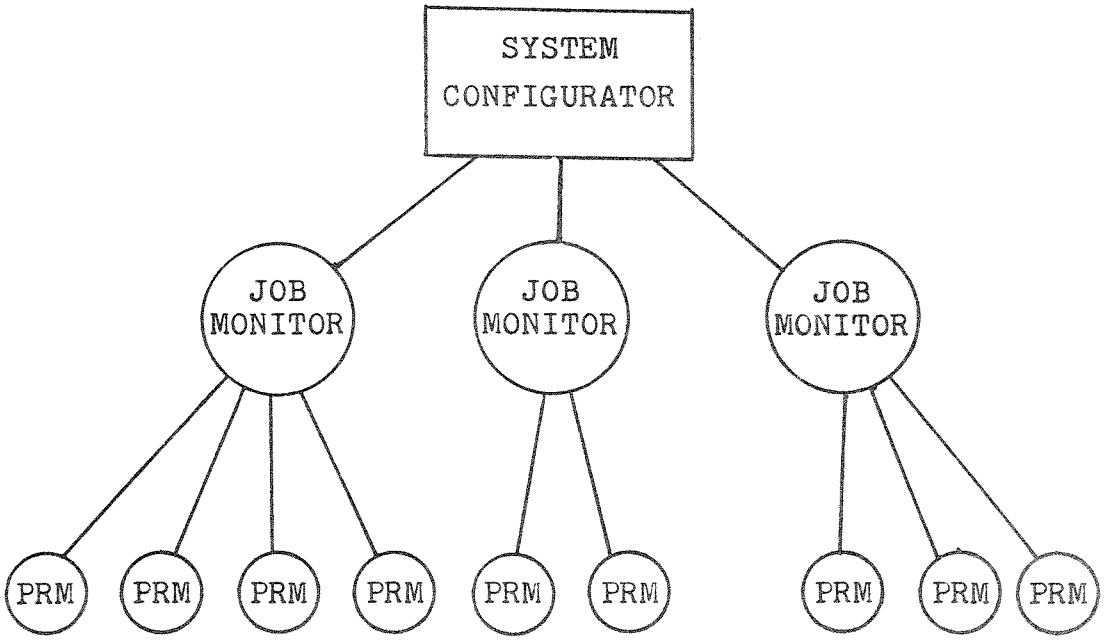

Figure 1-1: Operating System for RNA

Once the set of resources to be composed into a logical Computer System has been selected and the Logical Computer System has been physically established, the programs which define processes must execute correctly on any possible representation of a Logical Computer System. This implies that the node kernel or Processor Resident

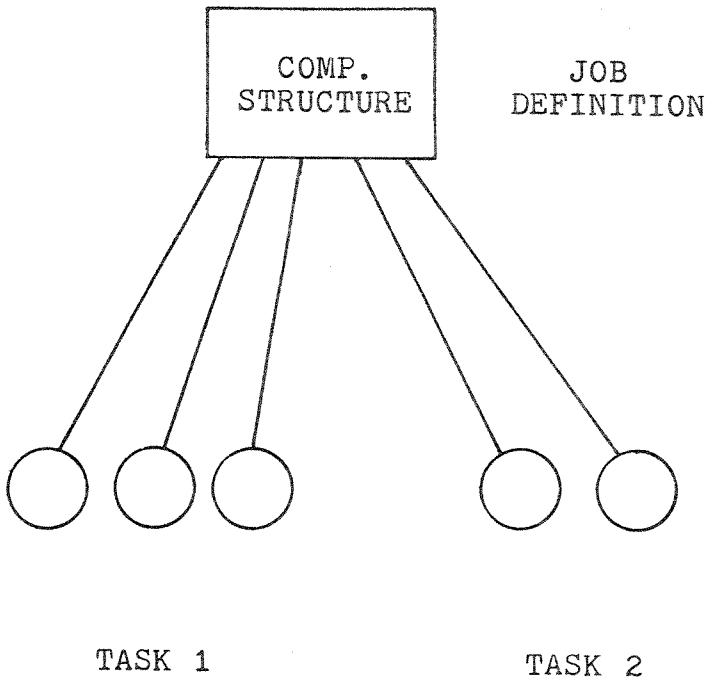

Figure 1-2: Job Structure

Monitor of each physical processor must be capable of correctly mapping all communications between the Logical Processor of a Logical Computer and between the Logical Processor of a Logical System. These communication requirements arise from explicit cooperation between these elements and from device virtualization such as virtualization of memory. The programs executing in a physical processor must perform correctly independent of configuration. This implies that the Processor Resident Monitor must be able to map any communications between

Logical Processors correctly independent of physical configuration. Thus, the PRM must have explicit knowledge of the program executing in the Logical Processor of which its physical processor is part.

Logical Computers are mapped to a set of processors and memory modules which are interconnected by a network as shown in Figure 1-3. At its highest level the Operating System must be capable of assigning a set of processors and memory modules to constitute the Logical Computer. It must establish the proper physical paths through the network as required by the Logical Computer.

Since a Logical Computer consists of one or more Logical Processors these physical paths must map the Logical Processors to the assigned physical resources. For example a Logical Computer may consist of two Logical Processors each four bytes wide. Network blockage may restrict one of these Logical Processors to a two physical processor configuration as shown if Figure 1-4. Processors 1 and 2 could not be assigned to Logical Processor A because of blockage in the network.

The execution of a task in this Logical Processor must be independent of configuration. Therefore

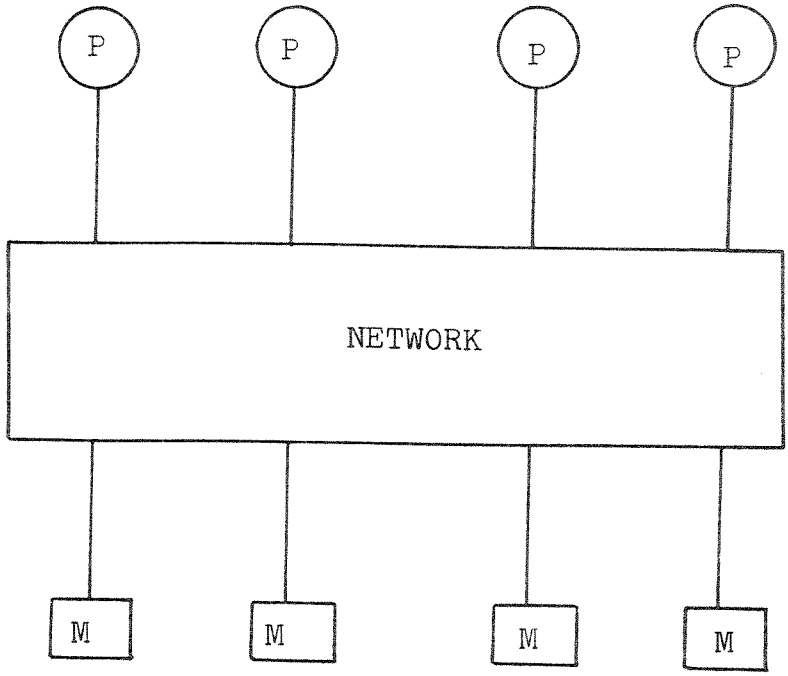

Figure 1-3: Physical Computer System

communication and data sharing must be handled by the Operating System regardless of physical configuration. Communication between tasks executing in a Logical Processor is handled by the Job Monitor while communication within a task is controlled by the TH-PRM

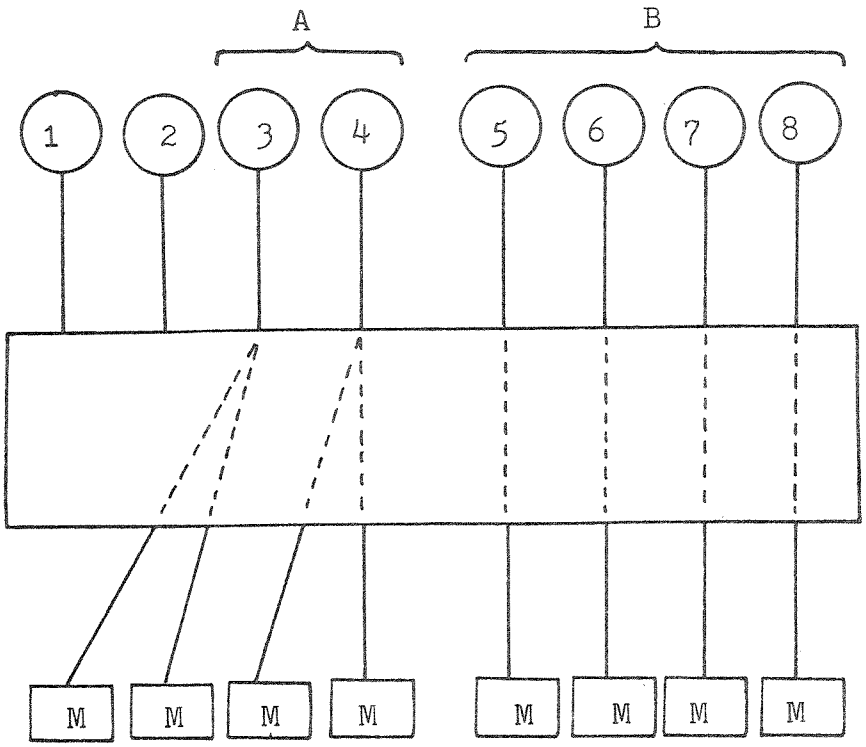

Figure 1-4: Logical Processors

Chapter 2

Operating System and Virtualization

2.1 Introduction

The Operating System for RNA must be capable of virtualizing the execution environment of a task which runs on a Logical Processor.

The architecture of RNA's calls for a hierarchically distributed Operating System. By hierarchical we mean that Operating System functions are assigned to different layers or modules and each layer relies on lower layers to carry out more basic functions [DIJ 68]. By distributed we mean that every job will execute in an environment without being preempted by any other job. There will be no processor or memory sharing among jobs. Inter-task communication is accomplished by a message based system.

The three hierarchical layers of the Operating System are: the System Configurator (SC), the Job Monitor

(JM) and the Processor Resident Monitor (PRM) [BRO 81], as shown in Figure 1-1. A brief discussion of the first two layers, namely the System Configurator and the Job Monitor are presented in Section 2.2 and 2.3 respectively followed by a detailed description of the Processor Resident Monitor which is the main topic of the research presented herein.

2.2 System Configurator

A major problem of Operating Systems of reconfigurable computers whose processors and memory-IO modules are connected through a blockage type network [LIP 79, KAP 80], is that of allocating resources to satisfy the computational needs of different tasks to be concurrently executed in the system. The SC is responsible for this task.

There is one SC in the system which will run in the processor which has a data bus to the memory-IO node which contains the switch [PRE 80, LIP 79]. that main functions of that SC are:

- Allocation of resources to satisfy requirements of the computational structure of the different jobs.
- Reconfiguration of assigned resources in order

to avoid blockage situations and to provide better utilization of system resources.

The functions of the SC are kept to a minimum in order to provide efficient response to the requirements for switch configuration management.

2.3 Job Monitor

The top level unit of work for RNA's is a job. Every job in the system is controlled by a Job Monitor, which is responsible for local management of that job. that jM is logically divided into two parts: the Job Control Policy Module (JCPM) and the Run Time Executive (RTE) [BRO 80a]. The JCPM is a user written part of the Operating System. It establishes flow of control, data flow, communication requirements among tasks and computational precision [FED 80]. The RTE is responsible for the coordination among tasks upon interpreting commands from the JCPM, and of communicating with the SC. The main functions of the RTE are:

- Analysis of resource requirements and configuration for the different tasks of the job.
- Negotiation of configuration and reconfiguration requirements with the SC.
- Inter process communication

4.2.1.1. Pipeline Communication

The way to implement pipeline communication is through the use of a shared memory. Let us suppose that we want a task, T1, to leave information of variable X to task T2 (See Figure 4-1). We have the following JCL commands:

```
EXECUTE (T1);
SEND X (T1) TO CH;
RECEIVE Y (T2) FROM CH;
EXECUTE (T2);
```

Task T1 is to execute and produce a value for X, which will be used by task T2 (which refers to it as Y). The following mechanism is used:

1. before executing T1, the RTE will change the descriptor of X, to point to a specific location of a shared memory (SM) which is being attached to T1. The RTE will send a modified descriptor to the respective PRM of T1 so that X can be updated to point to the SM, and not to a local memory. Execution of T1 can now be initiated.

2. task T1 is executed and will produce a value for X in SM. After execution the SM is disconnected from the processor of T1.

3. the SEND statement will cause the RTE to queue the modified descriptor for X into the buffer allocated to channel CH. the RECEIVE statement will cause the RTE to connect the SM to task T2. The descriptor for X will be passed from the channel queue to the respective PRM of task T2 and will replace the descriptor of Y for T2. So the descriptor for Y will now point to the SM space.

4. task T2 will be executed upon encountering the EXECUTE command, and will now have access to X of T1 as intended.

4.2.1.2. Active Task Communication

When communication is to be established between two or more active tasks, it can be achieved through message or data channels depending on the amount of information to be transmitted and the availability of shared memory modules.

Message channels are used to transmit low volumes of data among the tasks, and is accomplished by sending packets. Data channels are used to send high volumes of data among tasks and will be implemented through the use of shared memories.

1. Communication through Message Channels

   The use of message channels will move data to the address space of a task through packet switching. The action is as follows:

   a. the RTE sends packets to the SENDing task, instructing it to SEND the variable declared in the SEND command to the task(s) named in the CHANNEL declaration.

   b. the PRM of the SENDing task decodes the packet message, locates the required variables and its descriptor and initiates the packet transfer to the RECEIVEing processors.

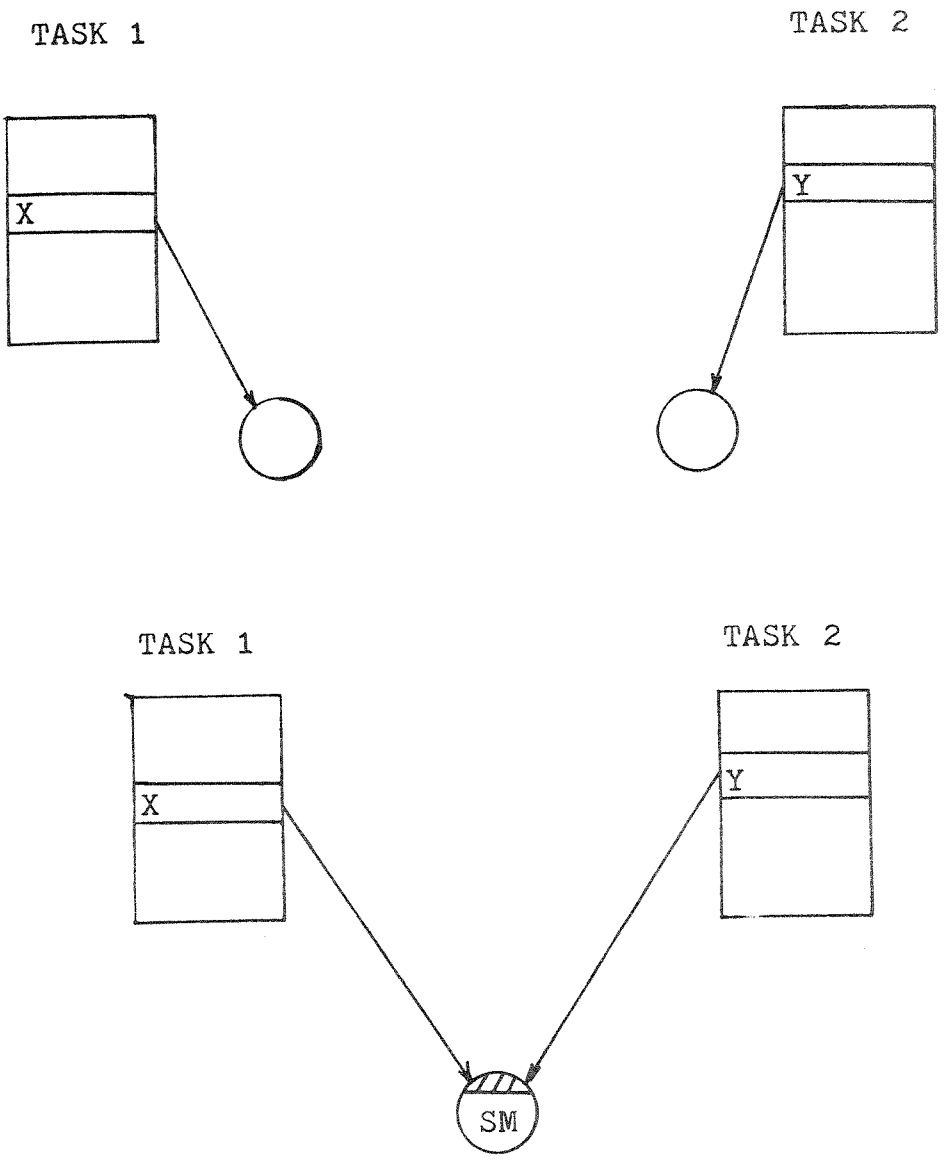

Figure 4-1: Pipeline Communication

c. the RTE SENDs a packet message to each RECEIVEing task naming the variable it should expect to RECEIVE.

d. the PRM's of the RECEIVEing tasks match the descriptor of the variable sent, with the descriptor of the variable it expects to receive and loads the variable into the appropriate storage location.

2. Communication through Data Channels

The PRM's of the SENDING and RECEIVEing tasks must also be coordinated in the movement of data along data channels. The Data Channels will first try to use shared memories, but if unsuccessful will use packets. The shared memory implementation goes as follows:

a. the RTE sends a packet message to the PRM's of the SENDing processors, instructing them to move data into the shared memory buffer associated with the channel.

b. the PRM's attach the shared memory and move the data to shared memory.

c. the PRM's (task group head) send a packet to the RTE noting that the transfer is complete and give the address of the just loaded buffer.

d. the RECEIVEing tasks are sent a packet message directing them to attach the shared memory and fetch the oldest occurrence of the named variable which is in the shared memory buffer.

e. the PRM's of the RECEIVEing task move data from the shared memory to the local data storage for the named variable.

The use of shared data objects requires coordination.

Execution of a memory reference to an address in a

non-attached shared memory will cause a trap to the PRM. The PRM will attempt to attach the shared memory by communicating this need to the JM. The requesting task will be blocked until the JM signals the PRM of the task that the required shared memory is available. The shared memory will be released upon exit from the block using the shared variable. The PRM will use the provided system procedures RESERVE and RELEASE to attach and release the shared memories holding the variables given as arguments. Execution of RESERVE blocks the calling task until the attach is made, as explained above.

4.2.2 WHEN/WITH Statements

During the execution of a Job, any of its tasks may require the exclusive use of a shared object. This can be requested with the use of the WHEN/WITH/DO statements or the WITH/DO statements. The first will provide exclusive use of a shared object WHEN some condition is met, and the second gives unconditional exclusive use to the shared object.

The occurrence of a WITH clause will cause the RTE (which knows which shared variables are in which memories) to place the request for the shared variables on a FCFS

queue. Whenever a given task reaches the head of the queue the RTE notifies the task that it can attach the shared memory which contains the desired variables. The PRM of the processor executing the WHEN/WITH/DO statements notifies the RTE of release of the shared variables when it exits the block containing the statements requiring exclusive use of the shared variables. The RTE then selects another task to receive the shared memory, notifies that task it can attach the shared memory and notifies the holding task to release the shared memory.

4.3 Intra-task Communication

The basic unit of computation structure is a task which is executed in a collection of processors, memory-IO units configured to create a partition for the task. The processors in the domain of a task operate in a SIMD mode, thus providing synchronous parallelism.

During the lifetime of a task, processors within the task will need to communicate with one another. This communication will be done by sending packets through the PRMs of each processor.

Since all processors operate in a SIMD mode, every

time there is a a need for communication, all processors must stop. This communication among the PRM's must be coordinated by one of the PRM's. This special PRM, called the Task-head PRM (TH-PRM) will coordinate this activity and, as we will see in the following sections, will also be responsible for task-wide decisions. The main functions of the TH-PRM are:

1. SEND/RECEIVE packets to/from other PRMs of the same or other tasks.
2. Synchronize the processors of the task.

Intra-task packet communication (or non-interrupt packet communication) differs from inter-task packet communication (or interrupt packet communication) in that interrupt packets will cause an interrupt upon arriving at the receiving processors, while non-interrupt packets must be expected by the receiving processor and do not cause interrupts upon arrival.

The broadcast bus is used to broadcast a signal to all processors of a task and is a good means for establishing synchronization among processors of a task.

Interrupt packets will be sent and received only by the TH-PRM of a task. In general, when a packet is

received by the TH-PRM, some information must be distributed among the processors of the task receiving the packet. This means that a one-to-many communication link has to be established between the TH-PRM and the other processors of the task [BAS 77]. In order to achieve this, the processors must enter an asynchronous mode of execution and later be synchronized again. There are two basic problems related with realizing the one-to-many communication link. First is the de-synchronization of the processors so each one can execute its own code, and second, the communication mechanism to be used once the processors are executing asynchronically.

1.- This first approach will take full advantage of the existence of the broadcast bus, and will use it as a means of processor communication as well as for synchronization. To use the broadcast bus, privileged instructions for manipulating the bus are required; namely, instructions for controlling the broadcast of information through the bus. It should be possible for the PRM's first to inhibit the broadcasting of information and also to be able to direct messages to specific processors of the task. This way asynchronous execution and communication among processors is achieved as follows.

When an external packet arrives at the head processor (i.e. the processor where the TH-PRM executes) an interrupt is generated. The microcode will vector execution to the External Packet Handler of the PRM. The External Packet Handler will execute the necessary code to inhibit broadcasting through the instruction bus. The TH-PRM will continue to analyze the packet, while the other processors of the task will enter a wait-broadcast--signal loop. The TH-PRM can now distribute information as required to the other processors of the task by being able to communicate with each processor individually through the instruction bus. When all processors are ready for re-synchronization the TH-PRM will signal all processors of this event, full broadcasting is re-established and the processors will initiate their synchronous execution.

2.- This second mechanism does not take full advantage of the broadcast bus, but will use it only as a means to start asynchronous execution. Communication will be accomplished through the use of non-interrupt packets. This approach requires that the broadcast of information through the bus may be inhibited when the fetch is done on page zero, that is no broadcasting is done upon execution of the operating system, if so required. The mechanism is as follows:

When a packet arrives and the interrupt is generated, the microcode vectors execution to the External Packet Handler. Still executing in lockstep mode the External Packet Handler will request for broadcasting to be inhibited. The TH-PRM will analyze the packet while the other processors will voluntarily enter a microcode wait-local-packet where they are waiting for the arrival of a non-interrupt packet. The TH-PRM is now free to communicate with each processor on an individual basis through non-interrupt packets. As each processor is ready to be synchronized it will enter a microcode loop waiting for a synchronization signal through the bus. When all processors are ready, the TH-PRM will request full broadcasting capabilities and the synchronization signal is sent through the bus to all processors. Lockstep execution is resumed.

3.-This third approach uses the reconfiguration capability of the system. Since instructions fetched are broadcast to all processors of a task, the only way to realize asynchronous execution is to break the instruction tree and re-configure the task into several one processor tasks. By doing this a one-to-many communication link can be established.

Reconfiguration is accomplished as follows. When an external packet arrives at the head processor all processors will enter the microcode due to the interrupt caused by the arrival. Control will be passed to the External Packet Handler. Still executing in lockstep mode the PRM's will de-activate the instruction tree, and all processors will wait for the hardware to signal that event. At this point all processors (except the head processor) will enter a microcode wait-local-packet expecting to receive a packet. The TH-PRM will analyze the packet and can then communicate with each individual processor through non-interrupt packets. Each processor will signal the TH-PRM upon completion of the requested task and will wait for a synchronization signal. The TH-PRM will record this, activate the instruction tree and then wait for the synchronization signal. When the processors are signaled that the instruction tree has been created lockstep execution resumes.

In the next sections we will discuss the three situations where intra-task communication is required:

1. Channel signals for SEND/RECEIVE CSL statements.
2. Paging situations.
3. Acquire/release of shared memories.

4.3.1 Channel Signals

A packet coming from another task or from the JM, will always be received by the TH-PRM. Before processing the packet the TH-PRM must first stop the other processors which are working in a SIMD mode. Once the TH-PRM has decoded the information it knows the type of action it needs to take (SEND or RECEIVE). The channel through which the information is to be SEND/RECEIVED and the type of shared object to which reference is made is also known since the PRMs have the descriptors for the shared objects.

The packet is decoded and, by examining the channel through which data is to be SEND/RECEIVED, the TH-PRM can determine if the transmission is to take place through a shared memory or by means of packets. Both cases will now be discussed.

4.3.1.1. Through Shared Memories

If the channel to be used is a data channel, then data will be exchanged through shared memories. The following intra-task communication will take place:

1. The TH-PRM will know which shared memory module to use since that information must have been supplied by the JM. The PRM's which will

participate in the transfer of information will be informed by the TH-PRM that they must connect themselves to the shared memory.

2. Processors which are unable to perform the connection to the shared memory will so inform to the TH-PRM.

3. The TH-PRM will tell the PRM's of the shared object they are about to SEND/RECEIVE.

4. The PRM's are instructed by the TH-PRM to read/write from a specific location in the shared memory depending on whether the operation was a RECEIVE or SEND.

5. Once all processors have obtained their information from the shared memory the TH-PRM will read the information for those tasks which were unable to connect to the shared memory and send them the information via NIGM.

6. The PRM's will again be synchronized and proceed in a SIMD mode.

Figure 4-2 shows the reception of a 32-bit word by a 4 processor task where processor No. 4 is unable to connect to the shared memory.

4.3.1.2. Through NIGM

In this case the TH-PRM will SEND/RECEIVE the contents of the shared object through NIGM. The steps involved in this procedure are the following:

1. The TH-PRM will inform the PRM's of the shared object they are about to SEND/RECEIVE.

2. The TH-PRM will either

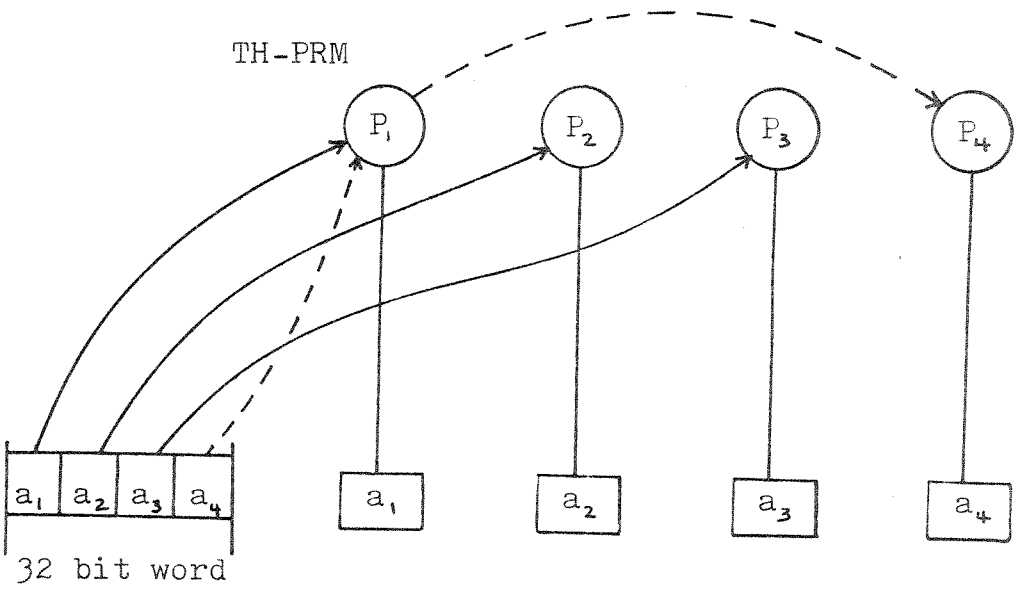

Figure 4-2: Packet Disassembly

a. receive information from the PRM's in which case it must assemble the information to produce an object of the desired length and send it to the requesting task.

b. send information to each PRM, for storage in the appropriate locations of their local memory.

3. The processors may now be synchronized to proceed in a SIMD mode of execution.

Figure 4-3 shows the assembly of a 32-bit word by the TH-PRM, from data received from each of the 4 processors of the task.

4.3.2 Paging

When a paging situation occurs the task must be reconfigured into smaller tasks so paging can be accomplished. This is explained in detail in Section 5.3 and in Chapter 6. Communication is limited to the distribution of information required by each of these smaller tasks in order for them to be capable of performing the page out/in operations.

Upon detection of a paging situation, the TH-PRM will obtain the type of page fault, the page causing the fault and the pages to be swapped out. This information is

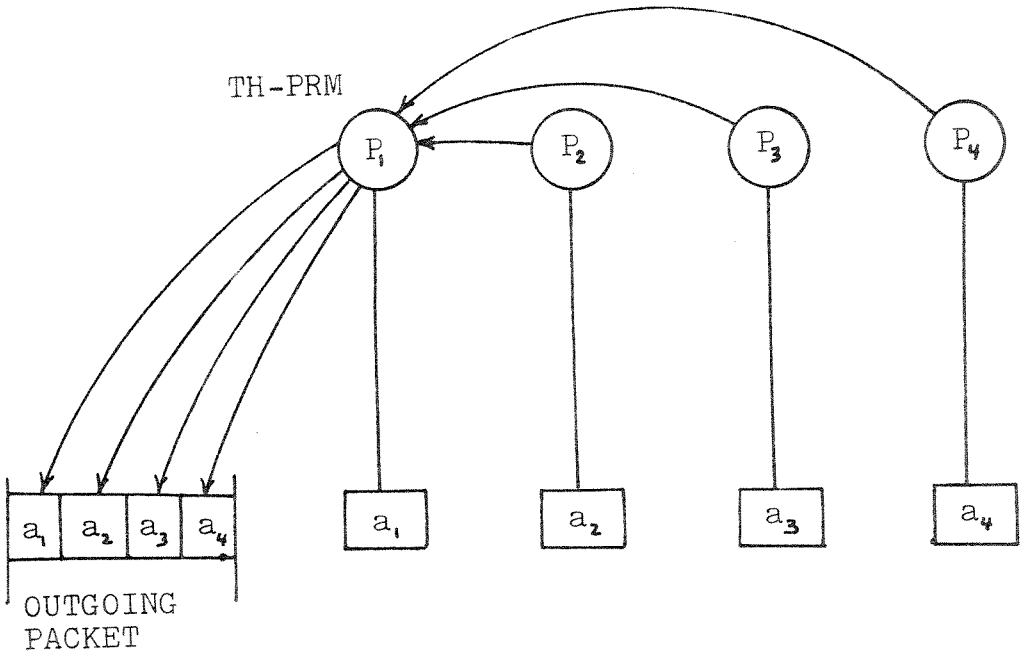

Figure 4-3: Packet Assembly

distributed among all processors of the task. After this has been accomplished the task is reconfigured into smaller units of execution so paging can take place.

Communication is limited under paging situations because the required reconfiguration is done locally by the task without intervention of other Operating System modules. This will be explained in detail in Section 5.3, Local Reconfiguration.

4.3.3 Shared Memories

When processors of a task try to access a specific location which is stored in a shared-memory module not attached to the processor a page fault will occur. As explained in the previous section the TH-PRM will take control at this point. The TH-PRM is able to recognize that the fault is due to the absence of an attached shared memory module, since at load time the JM had passed information as to which modules are shared by different tasks.

The TH-PRM will request the SM to the JM and the task will be blocked until the SM is available. Once the SM is available, the JM will assign the modules to the

task and inform the TH-PRM that the SM is available. If there is one memory module for each processor in the task, the acquisition of the SM can proceed in lockstep. If the module must be shared by all or some of the processors, the access will proceed as explained in Section 4.3.1.1. As each processor completes its use of the SM, the PRM will release the module and tell the TH-PRM that it has finished using it. Once all the PRM's have completed their transfer of information the TH-PRM will let the JM know that the task no longer needs the shared memory modules. The JM can now make this module available to other tasks. The task can now resume its execution in SIMD mode.

By having the TH-PRM do the negotiation with the JM, the JM will not have to keep track of which processor is currently using which shared memory module. The JM will assign the modules to the task, and is the responsibility of the TH-PRM to know which processor is currently using which module.

4.4 Communication Failure

There are two instances where Process Communication can fail in a Reconfigurable Network Architectured System. The nature of the Operating System,

distributed among a network, will make deadlock and process destruction potential communication failure situations. The next sections will describe how these two situations are handled by the Communication System.

4.4.1 Deadlock

The most common cause of communication failure is that of deadlock. Deadlock occurs when a process is waiting for an event that will never occur. A deadlock would occur if an Operating System module should be waiting to receive a message which will never be sent.

There are two instances where a Processor Resident Monitor would be blocked expecting to receive a message. The first occurs when the TH-PRM is waiting for a reply to a need-SM which was sent to the Job Monitor. The other case would be when a Processor Resident Monitor enters a micro-code wait loop expecting to receive a message from the TH-PRM. It can be easily seen that neither case can cause a Processor Resident Monitor to enter a deadlock state.

Before a task is created the Job Monitor negotiates a configuration for that task with the System

Scheduler. All the needs for shared memories are known a priori by inspecting the user's needs declared at the Job Control Language level. The user dictates the policies by which shared memory modules are going to be used by the tasks and explicitly expresses the acquisition and release of these modules. These two circumstances avoid the deadlock due to shared memory requests by a Processor Resident Monitor --first because a task is guaranteed to have physical access through the network to the shared memory modules, and second because a task cannot hold a shared memory module indefinitely, since tasks are forced to release shared modules by the Job Monitor when communication is to initiated.

These process by which a task is guaranteed access to its shared memory modules is similar to the deadlock avoidance technique in which all resources a task will use are assigned in advance [COF 71, HOW 73]. The only difference is that the resource in this case is shared among tasks of the Job.

The second instance of a blocked Processor Resident Monitor occurs in the micro-code wait loop. A Processor Resident Monitor enters this loop through a command given by th TH-PRM when it needs to communicate on