ソフトウェア工学 13－4
（1980. 3. 17）

THE CONTROL OF PROCESSORS AND ITS IMPLEMENTING ENVIRONMENT ON THE SIMULATOR FOR PARALLEL PROCESSING

Kiichi YAMATO, Kenji TODA and Nobuo SAITO
Department of Mathematics, Keio University,
Yokohama, 223 Japan

Abstract-A simulation system for estimating the performance of parallel processing system is now being developed. The design and implementation efforts are concentrated on 1) the full flexibility of the network architecture connecting all of the elements, 2) high precision simulation about conflict of the bus and the shared memory.
This simulator will be developed on the time-sharing system UNIX installed on PDP-11/60.

I. INTRODUCTION

The study of parallel processing systems with effectively constructed multiple processing elements has been studied since the appearance of digital computers. These systems are mainly used to deal with a large amount of information in a short time. In the last decade, the progress of the technology of the LSI produces many types of micro processing unit(MPU)s, with high control mechanisms. If a number of homogeneous or different kinds of MPUs are connected together to compose a computing system, it is possible to outweigh the weak points of a computer system through use of the increased number of MPUs. The application fields of the MPUs will be spreaded if they are connected to construct a multiple processor systems or networks. From these points of view, we are developing the simulator in order to estimate the efficiency of the parallel processing system on the practical applications. The principles of this simulator are:

1) finding better bus architecture of the simulated system for a given problem;
2) finding more effective structure of the protocol between processors;
3) checking the correctness and saving the history of the data access on the shared memory.

On this simulator, a processing element is simulated by a generated process. The synchronization of signals among simulated processors is implemented by proces switching, which occurs through time slicing or system call. The multi-processor environment is simulated by the way. We use the UNIX time-sharing system on PDP-11[1,2] for implementing the environment with many modifications.

Through use of this simulator, we can get many information about the simulated system for a given problem. For example, the behavior of the data transfer on the network are reported when a part of an operating system is executed on a multi-processor computing module.

In this paper, section II describes each elements of the simulation system. In section III, we treat the conflict and its analysis appearing in the simulated system. And in section IV is described how this system can be implemented on the UNIX.

II. CONSTRUCTING ELEMENTS

This system simulates a group of the processors with a single kind of an architecture. Each element has therefore the same processing power. The kind of an elementary processor to be simulated is not restricted to a specific one except for the bit-length of the operand of its instruction set. As each processor plays a rather important part of a given problem for controlling and calculating, the 1-bit processor or the 4-bit ALU can't be used because of the weak processing power.

2.1. The processor as an element of the simulated system

In this simulation system, each processor works asynchronously as an element of a computer complex. And the whole system can work as a MIMD machine[3]. Therefore, no system clock exists to synchronize the whole processing elements, and the data

transmission between two processing elements is asynchronous as a general rule[4]. The unit of operation managed by a processor has a variety in various application field, and the typical applications[5] of the multi-processor system are as follows:

a) The calculation of matrices is widely known as a typical utilization of the parallel processor. In this case, the unit of operation is an arithmetic operation on an element of a matrix. In this field, the processor is required to have the following abilities. 1) Rather high precision arithmetic operation, 2) high speed floating point arithmetic operation, 3) the ease of treatment on the occurrence of abnormal state(overflow,divide-by-zero etc.), 4) the capability of inter-processor transmission. As these items tend to be separated from the function of MPU itself, and if these functional element is recognized as a processor, the simulated processing element must have the same or more capability as modern 16-bit micro-processor.

b) In the application field that is mainly used as a process controller or a scheduler in the operating systems, the processor aims at the bit manipulation, the switching of processes, the controlling of the I/O devices and so on. It is important to resolve the conflict among accesses from multiple processors. Some computer system has special instrucion repertoire for solving these problems. And a few modern micro-processors also include these instructions to construct poly-processor system.

c) In the application having a large amount of data transmission among processors like as the sorting problem or as the controlling of multiple processors, the performance is not affected by the ability of numerical data processing. But the control mechanism for sharing the buses among processors needs to be simple and reliable.

Concerning about these points, we explain the architectures of Z-8000[6] and PDP-11.

1) Z-8001

This processor has 16 general 16-bit registers and 8M-byte addressing capability. The memory consists of 128 segments with 64k byte per segments. As the floating point instruction does not exist in the instruction set, this processor isn't suitable to operate floating point value. But the rich instruction set make it possible to implement powerful software for controlling a bus. For example, the instruction TSET(Test and SET) can be used as a semaphore. The MBIT, MREQ, MSET, MRES are used to construct multi-micro processors, and the signal μ(for synchronizing MPUs) assigned on a pin of the MPU can be controlled by these instructions directly. In our system, the fairy high precision simulator of Z-8001 is used to estimate the capability and performance of parallel processing systems in high precision.

2) PDP-11

The typical mini-computer PDP-11 which is already used in C.mmp and Cm* as the element of real poly-processor, is used in practice and estimate the efficiency of the network. Few speciality exist caused by in using PDP-11. Since there is no instruction to implement a semaphore for exclusive access on PDP-11, the extention about the memory reference is also included in the simulation system.

2.2. The implementation of memory

The difficulty about the implementation of the memory system depends on the connection method in the simulated system. If the memory space of the simulated processor is greater than the space of the base system implementing the simulator, the virtual space needs to be realized. The following problems exist in the implementation of the memory:

1) The bit length of the unit of the memory reference;
2) The capacity of memories;
3) The problem about the access rights from processor;
4) The prevention of the simultaneous access.

The items 1) and 2) have no problem as concerned with this simulator. As 3) is concerned with the connection between processor and memory, it need not be treated in the simulator of the memory. The item 4) is the main problem of the memory processing. If more than two processors send reading or writing requests simultaneously, the mechanism of the memory is made to act on one of these requests, the rejected one being failed. Consequently, for correct management of the simultaneous access, these demands are queued in the memory ssytem and treat them in order. On simulation of these events, it is necessary to regulate these accesses so that they are handled in the order of the simulator's clock.

2.3. The linking between processor and memory
The exchange of the control signal between processors is used to gain correct result on parallel processing[7,8]. Many linking methods have been proposed. The communication speed with the bus connection depends on two factors. One is the practical procedure of transmission on hardware, as the confirmation of linking and the transmission of data and so on. Another is a number of processors linked with the common bus. One of the purposes of this system is to decide the design of the bus network; to arrange the processors on the network from the standpoint of the speed and the reliability. This simulator examines the correctness of bus utilization in each exchange of clustered data or signal. The simulator detects the conflict on the bus, tells it to the report generator and enter these requests in the queue of each transacting process. A table(we call it the linking table) is provided to express the linking of processors and memories in a simulated system. The linking table is shown in fig. 2-1. When the bus is requested by a processor, the request causes the simulation system to verify the bus utilization by traversing the pertinent pointer in the table. And if the bus is free, the transmission success immediately and the simulator reports the transaction.

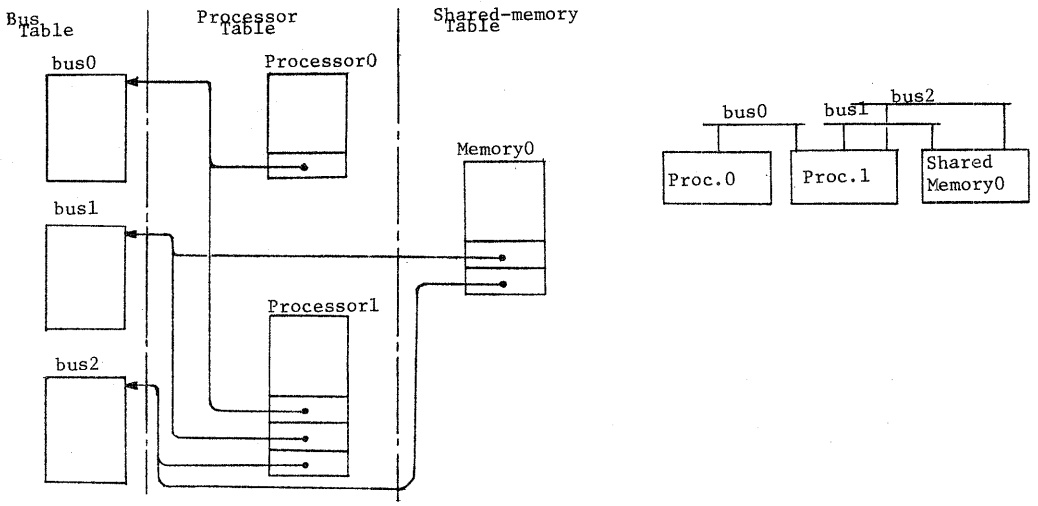

fig. 2-1 Linking Table

III. ANALYSIS OF CONFLICT

The conflict is the main problem on this simulator. It is defined as follows.

3.1. The definition of the conflict
The system, that simulates resource sharing between processors, inspects and manages the conflict of requests to estimate the drop of efficiency by the common use of the resource. The correctness about the simultaneous access can also be detected.

The judgement on the conflict in this system depends on the kind of the elementary processor to be simulated. These are defined as follows:
1) On the Z-8001 simulator
The simulator for the Z-8001 uses one clock cycle of the MPU as a tick of the clock of simulation system. The analyzer of the bus conflict first treats requests from the processors on the common bus. The conflict occurs on the memory unit, if the memory has multiple ports, and each port is connected to a different bus.
a) The conflict of different address reference.

If the preceding reference was read or write mode, the conflict does not occur.

If the mode is read-modify-write,

$$t(ref) \leq t(conf) \leq t(ref) + 18,$$

where t(ref) is the time when the preceding reference occurred, and t(conf) is the time when the conflict occurred. In order to keep the correctness of this memory, it is necessary to close the memory circuit during 18 memory cycles(the maximum 18 clock cycles needs to execute a TSET instruction).

b) The conflict of same address reference if both the preceding and conflicted reference are read mode,

$$t(ref) = t(conf),$$

if either of the preceding or conflicted reference is write mode,

$$t(ref) = t(conf),$$

and if the preceding mode is read-modify-write,

$$t(ref) \leq t(conf) \leq t(ref)+18.$$

If both the read and write operations are executed at one time, these accesses are logically invalid. The simulator reports following informations on the occurrence of this situation: a) the time when the conflict was occurred based on the clock of the simulator, b) the identifier of the processor requesting this data and c) the address and its contents.

2) On the PDP-11

The chief purpose of using PDP-11 itself as the element of the multi-processor system is to efficiently examine the behaviour of the simulated system and its program. Since the real processor is used as a processing element, high precision hardware clock is required to measure the time the request is generated. It requires nearly the same precision as the system clock of the PDP-11. Table 3-1 shows the cycle time of the PDP-11/60.[9] The minimum execution time for referring a memory is about 1.2 micro-seconds in mov instruction (e.g. mov r2,A). In this simulator, the count-up frequency of the clock for the measure of the estimation is about 100kHz to 1MHz.[10] The tick for confirming the occurrence of the conflict is therefore

1/(the count-up frequency) seconds.

The requests occurred within one tick are regarded as a conflict. This means that it simulates the pseudo processing device synchronized with a tick of the clock.(See fig. 3-1)

Mode	SRC Time	Read Memory Cycle	DST Time (A)	Read Memory Cycle
0	.00	0	.00	0
1	.51	1	.51	1
2	.51	1	.51	1
3	1.0	2	1.0	2
4	.68	1	.68	1
5	1.2	2	1.2	2
6	.85	2	.85	2
7	1.4	3	1.4	3

Table 3-1 The cycle time of the PDP-11/60

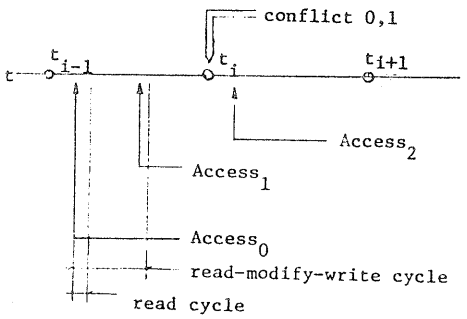

fig. 3-1 Conflict of the machine

3.2. The conflict about the shared memory

The assignment of the memory space in this simulator is as follows.

1) The local memory space : the local memory belongs only to a processor. The same address space of the local memory is assigned to each processor.

2) The shared memory space : the rest of the address space is assigned to the shared memory. The shared memory may be separated into some memory units. The bus address space is therefore,

$$L + \text{summention of } S(i) \leq A \quad (i = 1,2,\ldots, n),$$

where S(i) is the size of the shared memory i, L is of the local memory space and A specifies the whole space on the common bus.

The procedure dealing with the shared memory is as follows.

1) The access to the shared memory initiates the memory-process that treats the conflict in the strict execution and report the bus utilization.

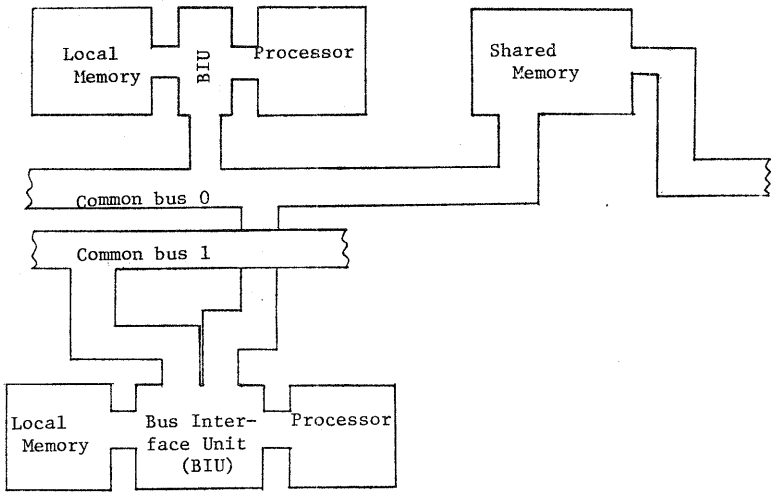

fig. 3-2 Processing Elements and its Metwork

2) The memory-process saves information about this access. And the process executes the actual access when no other access exists. (The reference is performed with read, write or read-modify-write mode.)

3) If one of these simulated processors doesn't pass the simulated clock time, the execution of the caller processor is suspended, and the information about this reference is held. The least recent processor on the time scale of the simulation clock is resumed.

4) The simultaneous access is processed in the order of occurrence with the report of the conflict, if all of the processors concerned have already passed the time.

3.3. The conflict of the bus

The conflict of the bus occupation is an unavoidable problem when the system uses the common bus architecture. On the real hardware, the bus management mechanism called the arbiter controls the occupation of the bus. On this simulator, the supervising program to control the memory and the processor does this work, when processors or memories request the same bus. Its procedure is as follows:

1) look up the utilizing situation of the bus linked to the processor. If the bus isn't free, suspend the execution until the bus occupation finishes. Otherwise,

2) search the least recent processor. And if the least recent is the caller, it occupies the bus until the access cycle ends. The caller processor passes according to the kind of the request (and it may cause new conflicts to other suspended processor).

3) Look up the destination element that the signal is transmitted. If it isn't found on the bus, the fail signal is returned to the caller processor and the occupation is teminated. Otherwise,

4) if no conflict occurs at the termination of the occupation, continue the least recent processor. If the conflicted processor is found, this procedure is applied.

IV. THE INPLEMENTATION ON UNIX

The basic elements of this simulator are, 1) processor, 2) memory, 3) clock, 4) network and signal. As it is difficult to describe complex simulated system only with these low level elements, we append the following functions: 1) wait, 2) signal, 3) kill, 4) packet. The following notes are the detailed description about them. (As the strict implementation is different between the Z-8001 and the PDP-11 simulator, both of them are described occasionally.)

4.1. Processor
Each processor is implemented by a process. The process creation on the UNIX is performed with the fork system call (we call these processes as the P.P.(Pseudo-Processor)). As the created process(called child process) is a copy of that of the caller of fork, a lot of areas are wasted in the main memory and in the process swapping device when creating many processes. It also increases the switching time of the processes. In order to restrain the size of these processes, it generates a individual executable module to be executed in a child process. These modules are executed by the execv system call(in the UNIX). This call replace the contents of a caller process with the specified executable module. Following problems are caused when employing this way:
1) The method how to make the individual executable modules written as a procedure in the program.
2) The input/output channels opened by the parent process are commonly used in the child process, and enables the pipe inter-process channel among child and parent processes. But execv closes all of the I/O channels when it is requested.
3) The whole amount of parameters of the execv system call are 512 characters. It is not enough to copy the data in shared memory or in the local memory of the parent process as the initial value of the local variables of the child process(for decreasing the reference of the shared memory).

The discussion of these problems are as follows: 1) all the procedures executed on each processors are build as the individual functions. When the child process is created, each process executes one of these functions. Each body of the functions is built as the individual executable module on UNIX. At this time, as each module is loaded as the reenterant program for common use, rather many (hundreds) processors can be simulated on this system.
2) and 3) the common file mechanism used by the UNIX (called pipe) can't be expected since the child process is initiated by the execv system call. The new protocol mechanism is provided for the inter-process communication. This simulator uses the above protocol implicitly.

4.2. Memory
The management of memory is left to the UNIX supervisor and language-C except for the shared memory scheme. In language-C, the variables allocated to a function are dispatched on the stack and are referenced by the relative address reference. This induces the pureness of the function data space, and disabling the external reference to the local space.

The reference to the shared memory generates a system call with : 1) the address in the shared memory, 2) reference mode and 3) the data (only write). The way of implementing the shared memory on the PDP-11 simulator, in implementing the memory management failure violation of the processor seems to be used, but it needs the interpretation of the violating instructions.

4.3. Clock
1) Z-8001 : The clock is implemented by software together with the execution of the processor simulator. The time slicing is also executed by using this clock.
2) PDP-11 : For partial interpretation, real clock is used for synchronizing the P.P. and for the time slicing. This clock is set to zero when the P.P. is initiated and is counted up on each processor. And this clock is suspended when the control fall into the simulator or into the UNIX system. The clock is resumed when the execution of the P.P. is restarted. It is set to the most recent suspended time when the processor is switched. Furthermore, this clock is used for time slicing to prevent for a processor from being simulated too long. It decrease the difference between simulated and the real execution.

In the UNIX, the line clock is used to schedule processes and to get time information. But as the tick of this clock is 20ms, it is useless to examine conflict. The interval timer is used for this aim. The count up pulse of the interval timer is strict internal 100kHz or higher external shimidt-triggered pulse.

4.4 Network
In section II, we discussed the bus network structure that can be simulated on this simulator. There need to be one process inspect the situation of the bus occupation. This process gives following data to the report generating process; the data through the network are:
1) address reference mode on referring the shared memory
2) received signal on communicating between processors;
(The receiver identifier can be treated as a special address.)
this process appends the following data:
1) referring time, processor-ID., bus-ID. being used,
2) referring time, processor-ID. of the sender, bus-ID. being used.

4.5. The processor control functions.
The following tools are provided to control the P.P.s.

1) wait(processor_id)
Through this function, the calling processor waits until a signal is brought from the processor specified with the 'processor_id'. If the id is -1, the sender processor id is not checked. Therefore a signal from any processor restarts the suspended execution. The return value of wait system call is the received signal value(16-bit unsigned integer value on this system). The wait is considered to be the conflict, generated by the caller processor, and the simulator treats it in same way as as the conflict.
2) signal(signo, func) , request(signo, processor_id)
These functions are used to transmit signals between two processors to synchronize them. The function signal is analogous and is widely extended from the signal in the UNIX. The function request is used to load a signal 'signo' on the common bus and send it to the processor specified by the parameter 'processor_id'. The conflict and the link of the bus has been described in section 3.3. The signal generated by the request is received by the wait or by the function specified by the signal. The signal cause an inerrruption to a specified function entry 'func', depending on the condition of the interrupted processor. This interrupt becomes effective when the interrupted processor is resumed at the next activity.

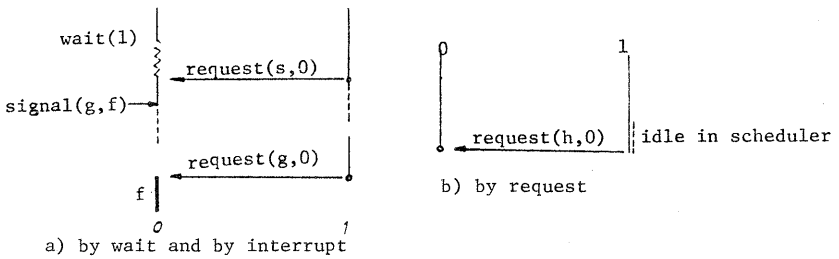

fig. 4-1 The relation of signal, request and wait

Fig. 4-1 explains the relation among signal, request and wait. The use of the wait increase the precision of the simulation, but it decreases the flexibility by interrupt programming.
3) kill(processor_id, range)
The kill system call cause termination of the P.P. specified by the 'processor_id'. The slight difference between the UNIX kill and ours is on the effective range. The kill on the UNIX terminates only the specified process. Therefore the termination of the parent process does not affect the execution of the child processes. The programmer of the simulated system must supervise the relation between parent and child processors. The kill in this system kills all of the decendant P.P.s if the parameter 'range' isn't zero. This call can be realized the algorithms that use the restricted numbers of processors efficiently.
4) packet
The inter-processor packet protocol is implemented to transfer the data set between arbitrary linked processors. The driver implements a full-duplex communication. There is no need to check errors on transmission. The necessary function for this protocol is that all arriving packets are queued in the input area, of the packet driving process. The long time bus occupation is caused by the packet transmission. The packet transmission is processed when the scheduler of the P.P.s on the simulator becomes active. The transmission of the packet is held up and queued in each bus while a processor is simulated. The system calls about the packet are as follows:
1) pkopen(proc_id) : open packet channel to 'proc_id'.
2) pkread(pk_id,buffer,count) : read 'count' bytes into 'buffer' from channel 'pk_id'.
3) pkwrite(pk_id,buffer,count) : load 'count' bytes from 'buffer' and transfer to 'pk_id'.
4) pkclose(pk_id) : free the channel 'pk_id.
The packet driver is initiated by the simulator and sleeps until the pkread or pkwrite system call is executed. The mechanisms of sleep and wakeup are analogous to the same UNIX system functions.

Many functions of the UNIX are available to implement this simulator. The following system structure is specially favorable to construct this simulator:

1) Multi-process environment: The users can create restricted numbers of processes by executing the fork system call. The communication is avaliable among parent and child processes. And as the communication procedure is written explicitly, there is no implicit communication effect.

2) The facility to implement the reentrant pure procedure program easily and efficiently : The program written in language-C becomes pure without rewriting, if each variable is defined as the local variable. And the execv system call enables to use the reentrant procedure efficiently.

3) Since almost all of the system is written in language-C, it is easy to modify the system.

4) As it is the small size operating system, it is easy to append functions.

CURRENT RESEARCH

This system is used not only for the estimation of the linking of processors but also for the software tool which is used to verify the correctness of the simultaneous access and is used as the debugger of the system software. As the information about the processes and the bus are held in the system, the number of simulated processor and the complexity of the network depends on the size of the simulation system. Therefore in the UNIX, the increasing of the system size causes the decreasing of the capability. In order to prevent this problem, the virtualize of the process information is now investigating. And moving some parts of the simulator into the user area is also being persude. Furthermore in this paper, we do not concern about the description language, but the use of the extended version of language-C is also being studied.

REFERRENCES

[1] "The Bell System Technical Journal," Jul-Aug 1978 ,vol.57, No. 6, Part 2

[2] "Documents for Use with the UNIX Time-Sharing System," (Sixth Edition)

[3] M.J. Flynn ,"Very high speed computing systems," Proc. IEEE, vol.54, pp.1901-1909.

[4] "VAX11/780 Architecture Handbook," vol. 1, 1977-1978.

[5] P.A. Gilmore, "Lecture Notes in Computer Science," No.24 pp.272-290, 1974

[6] "Am Z8000 Family Reference Manual," Advanced Micro device inc., 1979

[7] J.R. McGraw, G.R.Andrews, "Access Control in Parallel Programs," IEEE Trans. on Soft. Eng. vol.SE-5, No.1, Jan. 1979 pp.1-9.

[8] R.B. Kieburtz, A. Silberscatz, "Capability Managers," IEEE Trans. on Soft. Eng. vol.SE-4, No.6, Nov 1978 pp467-477

[9] "PDP-11/60 Processor Handbook," Digital Equipmemt Corp. 1977

[10] "PDP-11 Peripherals Handbook," Digital Equipment Corp. 1976

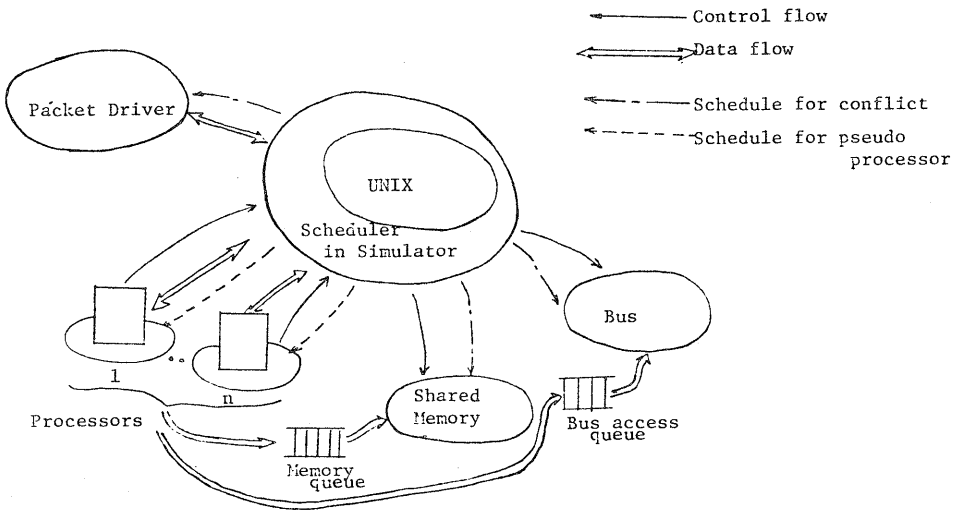

fig. 4-2 Simulation system on the UNIX