JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM
SFB: 366-91-3
July 8, 1991

TO: Time Warp Group

FROM: Steve Bellenot

SUBJECT: The Increased Use of Signals in the Sun version of TWOS 2.5.1

The Sun 2.5.1 version of TWOS uses signals in a different fashion than earlier Sun versions. Only the use of SIGIO (indicating input is available to read) and SIGALRM (indicating the real time clock has expired) has changed. Both of these signals were used in earlier versions, but their use is expanded in TWOS 2.5.1.

Implementation:

1. SIGIO:

A new global flag "maybe_socket_io" to indicate the possibility of readable data on some socket was used. The assertion is (outside the read routine) if maybe_socket_io is zero (false), then there is nothing waiting on any socket. (The converse statement is false, the flag could be true when there is not any readable data.) The SIGIO signal handler just sets the flag maybe_socket_io to true. The routine that actually does the reads from the sockets is now get_msg in SUN_Hg.c. At the start of this routine, maybe_socket_io is set to false. If a message is found on a socket, the flag maybe_socket_io is set to true and the message is returned. It must be done this way since there might be another message still waiting on this socket. Thus for maybe_socket_io to stay off, every socket must have had no ready data to read. Note that it is important for the flag to be cleared only at the beginning of the read routine. This is so that I/O doesn't get lost in the following sequence of events: Read routine finds socket x empty; input arrives for socket x; read routine finds socket y empty and returns claiming there is not any messages to read. The signal handler will reset the flag in this case and eventually the read routine will be called to read this message on socket x. (The routine get_tester_input also reads from a socket, the one connected to the host program, but maybe_socket_io is always set on exiting get_tester_input.)

2. Idling with sigpause.

In several places the code determines it has nothing to do and so it

sleeps waiting for the next signal. The easiest example is in the host program which we describe in detail by looking at the following code which is at the top of the hosts infinite main loop:

```
sigblock ( mask );
if ( maybe_socket_io == 0 )
     sigpause ( 0 );
sigsetmask ( 0 );
```

This uses "reliable signals". The sigblock call blocks all signals given by the parameter mask. Blocking a signal "holds" a signal until the blocking is turned off. The sigpause (0) call temporary clears all signal blocks and ether pauses until a signal arrives or returns immediately if there is a signal on "hold". The sigsetmask (0) call clears all signal blocks. The sigsetmask is needed for after the return from sigpause, the signals in "mask" are still blocked. Old style unreliable signals would use something like the following code:

```
if ( maybe_socket_io == 0 )
     pause();
```

which allowed the signal to happen between the test of maybe_socket_io and the call to pause, resulting in maybe_socket_io being set when the call to pause happened. The resulting program could hang waiting for an event which may have already happened.

There are other places were the pair sigblock(mask) and sigsetmask(0) are used to give "atomic" access to a critical section (the variable maybe_socket_io in the above case). As an exercise, the reader can show that the call to sigsetmask (0) is not needed in the host program, but is needed in the main timewarp loop.

3. SIGALRM:

A new global flag "timed_out" is set whenever SIGALRM occurred. The main time warp loop calls "check_timeouts" which clears the timed_out flag, checks to see what has timed out, calls the required routines and possibly resets the alarm. (Note: the usual UNIX alarm calls cannot used with this system!) Currently there are three possible routines to call, the gvt interrupt routine, the dlm load interrupt routine and a spare interrupt which is currently unused. Since there is only one real time clock per process in UNIX, there is only one alarm and its use must be multiplexed with other timing data needed for dynamic load management. The variable last_timer in the file SUNtime.c becomes a critical section which must be protected with sigblock--sigsetmask pairs. The signal routine for SIGALRM updates the variable node_cputime as well as setting

the timed_out flag. Currently this code is protected by "ifdef MICROTIME" wrappers, since it is designed to work on the butterfly and the hypercube with little additional work.

4. The sigpause in the main loop of timewarp.c

Care should be taken to make sure the sigpause is not called if there is anything left for timewarp to do. The long list of conditions checked before the sigpause is an indication of hard won experience. There is a chance that all are currently checked. (Calling sigpause can speed the arrival of signals.)

5. Deadlock solution

There was a deadlock condition in the old Sun socket code which has been removed in version 2.5.1. In the old code, both the read and write routines went into hard (i.e. possibly infinite) loops to complete partial reads or writes. Deadlock occurred when, for example, node 1 was trying to complete a send to node 3 while node 3 was trying to complete a send to node 1. The send from node 1 to node 3 to would complete only if node 3 would read from the socket node 1 was writing. This example deadlock occurred on every 4 node run of stb88.

The solution requires another global flag "partial_send" and periodic calling of the routine resend_msg until the flag is cleared. All possibly infinite loops in time warp need to do this. Currently, there are only two such loops known, the time warp main loop and the hard read loop mentioned above. The partial write hard loop has been replaced. On a partial send, data structures are filled which allow resend_msg to complete the send. Since there is room for exactly one partially sent message, no other message can be sent until the partial sent is completed. (It has always been ok for the low level send routine to return complete failure, it is the partial failure case which causes the problems above.)

6. Difficulties

The main problem in implementing the signal driven I/O was to multiplex the tester messages with the time warp messages. This required playing with the low level message layer and the tester layer which are not the finest code in TWOS. This multiplexing was necessary since they needed to use the same signal. In the end, the mercury like function calls were implemented for the Sun (in SUN_Hg.c of course), and low level message headers were added to tester messages. It is no longer the case that a tester call on the host will interrupt an object in an infinite loop. If this is desired, it possible to use SIGURNT to re-implement this behavior.

7. Justifications

Dynamic load management requires both real time measurements of object execution time and two alarms. UNIX with its single alarm and Mercury with its single malarm require supporting code to multiplex the two alarms. To support dlm, both something like MicroTime and a routine like check_timeouts are needed.

The signal driven socket I/O was required by performance. In version 2.5, all timewarp messages were read by polling all the sockets on every pass of the main loop. On average, each read of a socket was about 0.25 milliseconds which isn't much until one realizes how often this is called. Profiles of code execution showed that between 20 and 50 percent of execution time was spent in read calls.

8. Object timing modes

There are 3 object timing modes in the Sun version: None (NOOBJTIME) which doesn't time objects (this was the only option for the Sun before 2.5.1). Real or wall time (WALLOBJTIME) which may include time which UNIX runs other programs but wall time is needed for the flowlog. User time (USEROBJTIME) which is UNIX's measure of the time spent in the process. The simulator object timings are in USEROBJTIME mode. Because of the number of system calls to get the time, there is a significant overhead. Going from no timing to user timing increases the run time of pucks by 5%, stb88 by 3% and warpnet by 2%. Going from user time to wall time increases the run time again, for pucks it is 4% (9.5% over none), stb88 it is 2% (5.5% over none) and warpnet it is 2% (4% over none). At least the user timing mode is needed to support dynamic load management. Setting the object timing mode is a configuration file option.

9. If I had more Suns

Rather than reading every socket it is possible to do a "select" system call which finds out exactly which sockets have data available. The select call seems slower than reading four empty sockets, but could make a big difference if I had more Suns (or even if the 3/50's had more memory). The select code is still in SUN_Hg.c where it will be either a run time or compile time option.