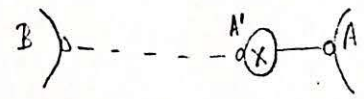
To	Lampson, Deutsch, Mitchell, McCreight	Date	June 8, 1972
From	Ed McCreight	Location	Palo Alto
Subject	MPL Debugger	Organization	PARC

MPL MEETING NOTES 6/6/72

1. Since there is no stack in MPL, each time the debugger gets control there is a place from which control came. The debugger must be able to explore the world with only a pointer to its source of control as a starting point. This is possible since each process has a context enough of whose format can be known a priori. There must be a basic set of primitive functions permitting one to follow pointers and determine types. From these primitives debugger macros can be built.

2. What MPL concept corresponds to the LISP backtrace? McCreight suggested a history list of "xfer" operations, truncated to avoid using excessive space. Lampson observed that a flurry of uninteresting function calls and returns would cause earlier interesting transfer information to fall off the end of such a list. He suggested matching function calls and returns on such a list so they would not use excessive space. McCreight proposed that a list of all processes kept in least recently executed order might be a useful debugging tool. (Control anarchy breeds debugging anarchy--Ed).

3. How should breaks be handled? The right mechanism for a break seems to be to attach a break process to a port in such a way that the process is transparent. Any re-binding of the port will keep the break process in the line of control.

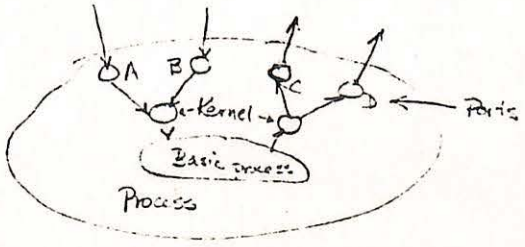


In the figure above, X is the break process and A' is its port. JOIN A TO B would now be interpreted as JOIN A' TO B. This creates a linguistic dilemma: how can A be joined by the debugger (or anyone else) to a new break process? REALLY JOIN? (Ugh!--Ed.) The user should be able to access and alter the state information kept by the debugger. Should one be able to break all port calls, or should the specific port breaks have to be set individually? Lampson pointed out that to break all calls will swamp one with system call breaks. Deutsch suggested that since all ports are known by the loader, sets of ports can be specified, and then broken and unbroken together using debugger macros.

discussion digressed to the question of accountability. Suppose two processes, A and B, "use" a process X. If it happens to be meaningless to assess resource usage against X, how should its resource usage be assessed against A and/or B? Deutsch suggested assessing against caller for resources consumed during call. Mitchell suggested that when a process X is destroyed, its creator should be assessed for its resource usage over its lifetime.

Returning from the digression, Mitchell suggested the use of a standard breakpoint process with standard ports. When an instance of this process is created to set a breakpoint, these ports are connected to ports in the debugger. On a break, the instance of the breakpoint process supplies the parameter list (from-port, to-port, message) to the debugger.

Lampson proposed the following model of a process, any of whose links can be broken:



Mitchell commented that this model includes a level of indirection which may not be necessary.