AT891-0259 RMC
W. E. Baker and J. S. Peek

92A062235

IBM Technical Disclosure Bulletin Vol. 35 No. 3 August 1992

SUPPORT FOR EXECUTION OF PROGRAMS WITH LARGE DATA REQUIREMENTS ON AIX 3.2

On AIX* v3.1 for the IBM RISC System/6000* it is not possible for processes with large data areas to run. This is not a restriction on the amount of paging space in the system, rather it is a restriction on the size of the process private data segment. This segment is 256 MB with some space for the kernel region (1/2 MB), the user stack (probably about 256 KB), and the rest for user data, user BSS and shared library data. However, there exist now some programs with extremely large data regions on the order of 800 MB. These simply do not run on the current system.

The solution is to allow these large programs to load by extending the user's data address space with other shared memory segments which will effectively give the user about 2 GB of space for data, BSS and malloc heap.

There are several current solutions in the industry. For instance, Cray uses machines with up to 4 GB of real memory, and does not have the notion of virtual memory. Therefore, this large program is loaded totally into real memory and allowed to run. The memory on a Cray is very expensive. Other UNIX** based systems have a flat address space with virtual memory and will only allow processes to run if there exists the paging space to handle all the memory requirements of the process. This requires a lot of paging space.

A binder option is proposed that indicates that this program is "large". A word is needed that will the size of the total data area needed by the process. This includes initialized data, BSS, AND malloc space. The binder option will be similar to the -S option which allows the user to specify a large stack size which is used by the system loader to increase the limit of the process to match the size given. The shared library data will not be included in the data that is located out in the new segments. Since the shared library data is prerelocated to match addresses in the process private segment, another relocation is not necessary and would have an adverse effect on performance. The loader will open the executable analyze the above mentioned word to determine if the data should begin in segment 3. Fork will have to copy the address space and exec/exit will need to be able to throw it away. The loader will then attach to the needed shared memory segments and do the relocation. The entry point to the module is in the header file, and the loader adds the appropriate relocation amount and off we go.

* Trademark of IBM Corp.
** Trademark of UNIX System Laboratories, Inc.