SOFTECH MICROSYSTEMS

A SUBSIDIARY OF SOFTECH

SOFTECH MICROSYSTEMS UCSD p-SYSTEM™ VERSION IV

C. A. Irvine

Vice President, Engineering

15 October 1980

SOFTECH MICROSYSTEMS UCSD p-SYSTEM VERSION IV

The UCSD p-System Version IV, is a major step forward in the development of the p-System. It is a result of about a half million dollar investment, and clearly demonstrates the dedication of SofTech Microsystems to the future of the p-System. The promise of the UCSD p-System lies in portability. A portable system in which FULLY portable applications programs can be developed promises to reduce software development costs, increase the real productivity of programmers, significantly reduce redundant programming efforts thus making programmer's work more interesting, and most importantly it can mean that more really useful programs can reach the hands of end users at lower cost than ever before. In order for any of this potential to be realized, the p-System cannot be allowed to rest on its achievements in portability. It must grow and develop so as to keep pace with the microcomputer industry's needs. Version IV was developed by SofTech Microsystems in order to restrengthen the p-System's portability (which was allowed to weaken somewhat during the development of version II.1 and III.Ø at UCSD), and to move strongly into important new areas such as multitasking applications programs. Version IV represents the first move towards a modern and powerful operating system. Preceding versions of the p-System included only a rudimentary operating system, which could perhaps be more accurately characterized as a "run-time support package". In Version IV the Operating System has taken on new responsibilities in the areas of task control, scheduling and memory management services. It reestablishes the dedication to portability, it provides important new facilities to the application developer, and it lays the foundation for the future growth of the system.

The primary objective of the Version IV development effort was to resolve the existing incompatibilities among the Versions II.Ø, II.1 (distributed by Apple Computer Corp.), and III.Ø (distributed by Western Digital Corp.). In order to accomplish this objective the design of the p-Machine had to be changed and, as a result, object code portability from preceding versions to Version IV could not be achieved. However, upward compatibility of source programs from versions II.Ø, II.1 and III.Ø was considered to be an absolute necessity.

Users of a preceding version of the system will find Version IV to be a major improvement and that there are some important new features and characteristics. The documentation was almost completely rewritten and an Internal Architecture Guide and Installation Manual have been added. The system is now provided with an integral Screen Control Unit and GOTOXY is now a UNIT which no longer requires the use of BINDER. The portablility of object code files has been improved by eliminating the byte-sex dependencies, and problems associated with real number constants. The librarian program has been rewritten and the user will find that it is much more understandable and easier to use. A couple of important utility programs, a p-Code Debugger and a powerful cross-referencer, have been added. The "new" features arise out of several sources. First, each of the preceding versions provided features not found in one or more of the others. Second, some important capabilities were added in order to provide a stable, integrated system containing all of the features of the preceding versions.

‾he new features provided by Version IV.Ø fall into two categories: program segmentation; and multitasking (concurrency). The important new operational characteristics of Version IV result from the use of these features within the operating system itself.

SEGMENT PROCEDURE's in Version IV, may be used much more conveniently than ever before. The programmer has much greater control over their residency, and the operating system provides much more effective management of them than in earlier versions of the system. For example, the new p-Code Debugger makes use of the new segment management facilities so that it is only loaded when an execution error or breakpoint is encountered. Version IV has eliminated the need for using the Linker to construct programs which use separately compiled UNIT's (except assembly language procedures), and thus has made program construction and checkout more efficient. The Pascal Compiler has been reorganized into a one and a half pass compiler and the 12ØØ byte limit on the size of procedures has been removed. The stifling limits regarding the number of UNITs and SEGMENTS which could be used within a program have, for all practical purposes, been removed. All of this means that much larger and more sophisticated applications systems can be written than can be with Version IV's predecessors.

The concurrency features of Version IV permit applications systems with "foreground" and "background" processes, and the development of applications programs with more logical and understandable structure. The foundation for the expansion of the operating system to multiprocessor configurations is also provided by the concurrency features.

The new segmentation facilities and the concurrency features provided by Version IV, have been used in the Operating System in order to produce a system which provides a great deal more useable space to the application programs while providing even more services than preceding versions of the p-System.

The development of this new version of the p-System required that all of the p-Machine Emulators be substantially revised, and that almost every major component of the p-System be heavily modified. Those that were not so drastically changed still required substantial testing. Many thousands of lines of validation code were written and used to ensure the reliability of Version IV. These validation programs will ensure that future releases of the p-System will be much more thoroughly tested than ever before.

UNITS and Program Segmentation

The facilities associated with the use of UNITS have been substantially improved in Version IV. UNITs may now contain SEGMENT procedures, processes and functions. They may now contain INITIALIZATION sections which are automatically invoked before the first invokation of any procedures within the UNIT. Files may now be declared in the IMPLEMENTATION section of UNITS.

Versions II.Ø and III.Ø of the UCSD System provided for separate compilation of program units of two kinds: ('regular') UNITS and SEPARATE UNITS. The use of the Linker was required to bind in either kind of UNIT to the host (main) program and since a unique copy was bound into each host, substantial space was sometimes consumed on disk volumes to house multiple copies of the same code segments. The advantage of SEPARATE UNITs over 'regular' UNITs was that when a SEPARATE UNIT was bound in by the Linker, only those procedures were bound in which were actually called by the host program .

In Version IV.Ø, the need for SEPARATE UNITs has been satisfied by allowing UNITs to contain SEGMENT PROCEDUREs. Since SEGMENT PROCEDURES are only loaded when called (and released from memory when exited), the functional capabilities of SEPARATE UNITS can be attained with 'regular' UNITs which contain SEGMENT PROCEDURES. This strategy was chosen for Version IV.Ø because it not only meets the need but makes the UCSD Pascal language definition more uniform. It in fact results in providing capabilities not present in any of the preceding versions, and simplifies and improves the separate compilation facilities of the system. In addition the limits regarding the number of segments which may be used in a program have been substantially relaxed in Version IV.Ø. In Version IV.Ø a compilation unit may contain up to 16 segments, up to 256 segments may be referenced by a compilation unit, and up to 256 compilation units may be referenced by a program. These relaxed limits have, for all practical purposes, removed all serious constraints to the use of separate compilation and segment procedures.

Version II.1 provided, in addition to 'regular' UNITS, the INTRINSIC UNIT. The advantages of INTRINSIC UNITs were that they did not require the use of the Linker, and that disk images of INTRINSIC UNITs are shared among all of their host programs. The primary disadvantage of INTRINSIC UNITs, as implemented in Version II.1, was that a "segment number" had to be assigned at the time the INTRINSIC UNIT was compiled, and each host program had to use the unit with that segment number. This constraint severly limited the utility of INTRINSIC UNITs. For example, two INTRINSIC UNITs which were assigned the same segment number at compile time could not both be used in a single host program.

Although the Version IV.Ø Compiler accepts declarations for all three kinds of UNITs (UNIT, SEPARATE UNIT, and INTRINSIC UNIT), it treats all units as INTRINSIC UNITs, with one major improvement. In Version IV.Ø the segment number assigned at compile time (if any), is not fixed at compile time of the host program. Therefore the primary constraint of Version II.1 with regard£to INTRINSIC UNITs has been relaxed in Version IV.Ø.

In addition to the changes discussed above, Version IV.Ø has taken a radical departure from the memory management strategies employed in earlier versions. Prior to Version IV code segments were loaded onto the stack as they were invoked. In Version IV.Ø code segments are managed in a "code pool" which is separate from both the stack and the heap. The code pool management strategy allows considerably more flexible operation than was ever before possible. Segments may be dynamically loaded, unloaded, and relocated as execution time needs demand. The code for a segment may be shared among several processes which use it. A "activity count" is maintained for each segment. The activity count reflects the frequency with which the segment has been used and how recently it was used. Segments which are not memory locked are removed from memory when necessary, and the segments with the lowest activity count are the first to be removed under such circumstances. All segments, including those resulting from the main body of a UNIT or PROGRAM are managed in this fashion. In previous versions only segment procedures and functions were, in a very constrained manner, loaded dynamically.

Perhaps most importantly, the Code Pool approach is the basis for new facilities which permit the programmer and the program an opportunity to manage the code segment "working set". Two .PAGE 8 new system calls have been introduced in Version IV.Ø:

```
MEMLOCK(<seglist>);

MEMSWAP(<seglist>);
```

where <seglist> is a string (constant or variable) containing a list of segment names.

The MEMLOCK call requests that the operating system should ensure that the segments specified in <seglist> should, once they are loaded, remain in memory even when they become inactive. The MEMSWAP call declares that those segments specified in <seglist> should return to normal segment status, i.e. competing on the basis of activity counts with other segments for space in the main memory. Quite independently of their importance in programs which use concurrency facilities, these two calls provide a means for the program to exercise much greater control over memory utilization. A program can, for example, under Version IV.Ø dynamically adjust its segmentation and thus reduce disk accesses for frequently used segments, and free up memory for rarely used segments.

Significant improvements have been made in the area of heap storage management. Three new intrinsic functions (VARNEW, VARAVAIL, and VARDISPOSE) which permit much more natural control of heap storage allocation than the formerly common practice of sequential calls to NEW. DISPOSE is now fully supported and in fact can be used in programs which employ mark/release strategies.

In summary, Version IV.Ø accepts (for compatibility) all three kinds of UNIT declarations. It treats them all as INTRINSIC UNITs, whose segment number may vary from one host program to another. Furthermore a UNIT may, in Version IV.Ø, contain SEGMENT PROCEDURE declarations, private file and USES declarations, and the residency of all SEGMENTs can be dynamically controlled through the use of MEMLOCK and MEMSWAP calls.

Multitasking (Concurrency)

Version III.Ø provided facilities for the construction of multitasking and/or interrupt driven application programs. The facilities were provided through£the addition of:

1. A new kind of procedure called a PROCESS;
2. Some system calls to control the execution of a PROCESS;
3. A new predefined data type, the SEMAPHORE; and
4. Some system calls for manipulation of SEMAPHOREs.

A PROCESS is essentially a special kind of procedure. PROCESSes provide a means for dividing an application program into 'logically concurrent' tasks. PROCESSes are all invoked by the main program. The easiest way to think about PROCESSes, is to consider a multiprocessor system where there are enough processors so that each task may be assigned to an available processor when it is invoked. Such tasks all may access the global data of the main program and thus may communicate through that data. If there were not a sufficient number of processors, then the operating system would sometimes have to place a new task 'on hold' until a processor became available. It would also, if more than one task was on hold when a processor became available, choose which task should next be assigned to the available processor. PROCESSes are handled in the UCSD System in this manner, however the system is prepared to handle only one processor and thus only one task is active a time.

SEMAPHOREs are a predefined data type which permit tasks to synchronize their activities in several important ways. In essence they provide a reliable way for tasks to ensure that they can use shared resources safely and consistently.

The SEMAPHORE manipulation calls provided by the system provide the means for a task to SIGNAL another task and for a task to WAIT for such a SIGNAL to arrive.

The MEMLOCK and MEMSWAP calls discussed above have been added to the facilities of Version IV.Ø, in order that the program can ensure that 'active' tasks (which are awaiting the arrival of a SIGNAL, perhaps provided by an interrupt) are not removed from memory.

In order to provide a greater degree of protection, the Operating System contains mechanisms which effectively lock the file system so as to prevent inconsistent file activity.

The multitasking features provided by Version IV.Ø provide the means to construct multitasking application programs. Examples of such programs include:

> programs which print reports via one task while accepting input data via another task;
>
> interrupt driven programs, such as real-time process control applications;
>
> multiaccess application programs which provide services to users at more than one terminal; or
>
> programs which are organized as a set of "co-routines".

The MEMLOCK and MEMSWAP calls discussed above have been added to the facilities of Version IV.Ø, in order that the program can ensure that 'active' tasks (which are awaiting the arrival of a SIGNAL, perhaps provided by an interrupt) are not removed from memory.

The multitasking (concurrency) facilities provided by Version IV.Ø are described in detail in the Version IV User's Manual.

ATTACHMEN

"UPWARD COMPATIBILITY"

This appendix surveys various aspects of the compatibility of Version IV of the UCSD p-System with preceding versions of this software system. Although an important objective of the Version IV development effort was to achieve full upward compatibility from versions II.Ø, II.1 and III.Ø, some aspects, such as object code compatibility, had to be sacrificed in order to expand the capabilities of the system. Furthermore, it is possible to write non-portable programs (by accident or intent) even in the UCSD p-System. Certain implementation dependent and hardware implementation practices when used in programs running under earlier versions may cause difficulty under Version IV.

The following sections of this attachment will address several dimensions of the upward compatibility problem.

Source Level Pascal and FORTRAN Programs

In general this objective has been achieved. In fact Version IV has also achieved a somewhat higher level of object portability than earlier versions. All 'normal' UCSD Pascal and all FORTRAN 77 programs which run under Versions II.Ø, II.1 or III.Ø can be compiled, without change, and run under Version IV.Ø. In general terms, a 'normal' program is one which has not made use of implementation characteristics of the compiler, operating system or P-Machine. The implementation dependent Pascal programming practices which will require program modification are defined below.

Object Programs

Some changes and extensions have been made to the P-Machine, and as a result Version II.Ø, II.1 and III.Ø object programs cannot be executed under Version IV.Ø. Source level programs must be recompiled under Version IV.Ø in order to execute properly under IV.Ø.

Full portability of object programs and disk directories is supported among Version IV implementations. That is, the byte-sex, floating point constant, and directory "flipping" problems have been solved in this version.

Assembly Language Procedure and Functions

Assembly language PROCEDURES which accept parameters of type STRING, and assembly language FUNCTIONS will need slight modification in order to be used under Version IV.Ø.

Media Conversion

External storage media logical formats have not changed in Version IV.Ø. Therefore, no file or media conversion is required for program text or data.

Version Dependent Programming Practices (Pascal)

The following paragraphs define those Pascal programming practices which will cause programs which execute correctly under Version II.Ø, II.1, or III.Ø to fail when compiled and executed under Version IV.Ø. The use of any of the following programming practices will require that a program containing them be modified before recompilation under Version IV.Ø:

System Data Structure Dependencies

Some of the definitions of system data structures have changed under Version IV.Ø. Therefore programs which directly access such structures (e.g. SYSCOM, SIB's, etc), will have to be modified in order to execute correctly when recompiled under Version IV.Ø.

Heap storage utilization

Under Version IV.Ø the program cannot assume that the memory immediately following that obtained by a NEW is unoccupied and therefore available for use by the program.

Similarly, the program cannot assume that the storage obtained by consecutive calls to NEW will yield contiguous storage. The practice of indexing across the boundary separating storage obtained by consecutive execution of NEW may therefore fail under Version IV.

Calls to MARK and RELEASE, must, under Version IV.Ø be paired correctly. The pointer value obtained by calling MARK must not be modified prior to calling RELEASE with that pointer value. Furthermore that pointer value obtained by calling MARK cannot be used as a base pointer for storage references.

In previous versions segment procedures comprising a large program could be compiled separately and then stitched together using the LIBRARY program. This was possible because segment numbers were assigned only to explicitly declared segments or units. Thus it was relatively easy to arrange dummy segment declarations in the separate compilations to assure consistency of segment numbers. In Version IV this a good bit more difficult and therefore UNITS (which are designed for separate compilations) should be used instead.

"Tightly Fitting" Programs

The user has a great deal more control over the amount of memory required in Version IV.Ø for the operating system during run-time than in any of the preceding versions. However, Version IV.Ø does, in general, use more memory during execution of a user program than Version II.Ø, II.1 or III.Ø. Therefore those application programs which have been carefully tailored (using SEGMENTS, INTRINSIC UNITS, SEPARATE UNITS, etc.) to fit "tightly" under one of the preceding versions, may not fit, without some modification under Version IV.Ø.

The collection of memory management facilities have been increased substantially under Version IV.Ø, and as a result "tightly fitting" programs can be modified so as to have larger amounts of available memory than in preceding versions.

In summary, programs which were carefully tailored to fit the available memory in preceding versions may need to be modified to fit under Version IV.Ø, but Version IV.Ø can accomodate larger programs than preceding versions.

On-Line Volume Management

Since the Version IV.Ø operating system is now segmented, and segments can be released from memory when they are not actively executing, the disk containing the operating system segments must remain mounted at all times. Therefore programs which required that the system disk be removed and replaced by other volumes, may need some modification for correct execution under Version IV.Ø.