The MPE IV Kernel : History, Structure and Strategies

John R. Busch
Member of the Technical Staff
Hewlett Packard Corporation
Computer Systems Division
19447 Pruneridge Avenue
Cupertino, California
95014

Abstract

The MPE IV kernel is the result of over three years of research and development undertaken at Hewlett Packard's HP 3000 R&D lab in Cupertino. It provides a new high performance, integrated, extensible foundation for the 3000 operating system, MPE. The project's history and the kernel's characteristics are described. Project objectives, investigation approach, implementation methodology, functional characteristics, resource management objectives and strategies, and performance results are presented.

1. Introduction

The evolution of the HP 3000 family towards large main memories, fast processors, and large and distributed configurations stressed the original MPE kernel design and implementation. It became clear that just supporting the evolution in terms of kernel data structure extensibility would become a problem. Moreover, the original algorithms could not be relied upon to exploit the performance potential offered by the larger configurations.

Project objectives for a new kernel were established. Research into the growth and performance limitations of the old kernel and into state of the art approaches to resource management policies was undertaken. Alternative designs were established, implemented and evaluated.

This process, culminating in the MPE IV C-Mit, is presented in the following sections.

2. Kernel Project Objectives

The primary project objectives for the MPE IV kernel were to provide :

(1). support for the evolution of the HP 3000 family;

(2). maximum performance across the family members;

(3). high reliability and improved fault detection and recovery;

(4). increased functionality as required by the other system components of MPE IV; and

(5). simple extensibility when unforeseen system requirements surface.

3. The Investigation

The 3000 architecture, workload, and evolution were to be matched by the new kernel.

The investigation would procede by : identifying the characteristics of the workload; determining the growth limitations and performance problems of the existing kernel; researching and consulting to determine promising approaches to kernel design; and formulating alternative designs.

Instrumentation was placed into the old kernel, and the system was measured under reproducible, representative environments.

Service requirements induced by the workload on the various system servers were determined. Distributions of segment sizes, processing requirements, and access requirements to secondary store were observed.

Resource utilization was measured. Disc, main memory and processor queue lengths, request type distributions, and overall utilization were determined.

Migration among the servers and service delays were characterized. Process stop type (eg segment fault, disc I/O, terminal I/O, time sliced, etc.) distributions and service delays for each stop type were measured.

These measurements were to be used not only in isolating invariant workload characteristics and performance problems but would also be used as the base for later comparisons. (The specific growth and performance limitations of the old kernel are addressed in later sections).

The literature was researched and academicians consulted to ensure that the lessons of the past and the academic investigations whose results offered potential were taken into account. Although extensive literature was available on resource management policies, very limited literature was to be found which directly related to the segmented architecture of the 3000 family. What was acquired from the research and consulting was a set of principles, measures and ideas which could be incorporated into the design and implementation.

The investigation phase resulted in an understanding of the problems and limitations of the old kernel, and a set of alternative strategies which eliminated the limitations and problems and offered potential in satisfying the overall objectives. Rather than coming up with the eventual design, the investigation came up with a commitment to try out the alternatives and select the best strategies for the architecture and enviroment based on measurement rather than intuition.

4. Performance Goals and Strategy

In this paper, a transaction is considered to be a step in an interactive session which begins when carriage return or enter is hit and terminates when the system is ready to accept further input form the session.

The global performance indices for the intended application environment are :

(1). transaction response time;

(2). transaction throughput;

(3). fairness;

(4). batch throughput.

The desired system behavior is as follows :

- For a given workload and configuration, the system should provide minimum transaction response time with maximum transaction throughput. Batch performance should be "acceptable."
- Under increasing load, the system should be stable. As the load increases, transaction response time should degrade gradually and fairly. System throughput should stay high even under very heavy loads.
- The system should dynamically tune itself to optimize performance for the current workload with the given configuration. However, explicit control over the relative service between transactions and between interactive and batch should be available to the operator and system manager.

It must be kept in mind that the bottom line performance of the system is measured by the global performance indices and not by the factors which may influence them. This suggests that the performance strategy should be directed towards optimizing the global indices and not towards optimizing indices local to each system component.

The selected overall performance strategy was to achieve maximum system performance by having the system components cooperate to optimize the global performance indices. This approach is fundamentally different from having each system component attempt local optimization and hoping the result will be good overall performance.

Measures and associated instrumentation were defined for the global and local performance indices and supported by the measurement interface. With these measures, the effects of alternative strategies could be understood and evaluated. The measurement interface through performance tools would be made available in the field so that on-site trouble shooting and tuning could be performed.

5. Implementation Methodology

In order to achieve the desired high reliability and natural extensibility, the implementation would have to be highly structured.

Interfaces between system components would be explicit, general, and adhered to. Access to kernel services and internal information would be available only through the use of explicit messages or the invocation of kernel interface intrinsics. An adequate set of interface intrinsics and a general, efficient internal message system would be required to support this structured interfacing.

Within the kernel, a structured implementation was absolutely necessary so that alternative resource management policies could easily be incorporated, coexist, and eventually be deleted.

Performance considerations at the instruction level would be of secondary concern in favor of a structured implementation. The sought after high level of system performance would be achieved through integrated, parallel policies rather than by relying on highly optimized code sequences.

The algorithm selection process and the support of MPE IV performance tools would require that complete instrumentation and an instrumentation interface be carefully designed into the new kernel.

6. The Internal Message Facility

A high speed memory resident message facility was defined and implemented. The facility is intended for the transmission of operating system status and control messages. This facility eliminates the need for supporting multiple ad hoc communication mechanisms.

The message facility associates a message harbor with each process. Each message harbor contains 32 message ports. Each message port contains a FIFO queue of messages, where a message is up to 5 words in length (maximum length is configurable).

Message intrinsics are provided to send a message to any port of any process, to determine the status of any or all message ports of a process, and to receive in a destructive or non-destructive manner the message at the head of a specified message port.

Use of the internal message facility is limited to operating system code. User level inter-process communication is available through MPE IV message files.

7. The Measurement Interface

In order to evaluate alternative strategies and to support the envisioned and yet to be envisioned MPE performance tools, an extensible measurement interface was designed and implemented.

The existing MPE measurement tools were highly dependent on the kernel implementation. They were knowledgeable of internal data structures and called very low level kernel routines or exerted direct control over resource management. Modifying the tools to support the new kernel would be an inadequate solution since the tools were inadequate for the evaluation task at hand, and future changes would create the same problem over again. The decision was made to attempt to centralize support of measurement requirements within the kernel itself, and to make the tools independent of the kernel's implementation.

The basic requirements of the existing and envisioned tools were investigated. An interface was defined which would provide the mechanisms, support structures, and the access and control intrinsics so that the information needed would be obtainable through intrinsic calls without knowledge of internal structures or policies.

The key objectives of the interface were : service (provide what's needed by the current tools); transparency (eliminate dependencies of performance tools on system internals); extensibility (meet future requirements by natural extension of the initial specification); and low overhead (so that use of the interface would minimally effect the performance of the system under test).

The resulting measurement interface supports complete system global and process local statistics gathering, selective measurement class enabling/disabling intrinsics, statistics class delivery intrinsics, and complete cleanup upon abnormal termination of a process which had enabled statistics gathering. Tools using the measurement interface require no privileged code so that system reliability is improved and the sought after independency from the kernel implementation is achieved.

8. MPE IV Kernel Resource Managers

Each resource manager operates independently through clean interfaces so that strategy or data structure changes of another resource manager will not effect him. Each resource manager is built from structured, general pieces so that alternative strategies can be easily implemented.

The management of the disc, main memory, and processor resources has the primary impact on system performance. The approaches taken towards resource management for these key resources is sketched in the remainder of this section.

8.1 Disc Management

Disc management policies have an extremely significant effect on system performance due to the workload characteristics. Transaction processing applications on the 3000 are characterized by several disc references per transaction with short processing requirements between references. In such an environment, good disc management is essential in achieving good system performance.

The goal of disc management should be to provide maximum disc subsystem throughput while minimizing the service time for the most important requests. The selection of policies for the management of disc space and the scheduling of accesses to secondary store should be based on achieving this goal.

The disc management systemi nterfaces with the file system and the memory management system when allocating disc space and servicing requests to access secondary store.

The major problems identified with the old disc management policies revolved around virtual memory management, disc access scheduling, and serial seeking.

Disc space for data segments was restricted to a single volume (i.e. virtual memory limited to the system disc). This restriction has serious detrimental effects on system growth and performance. The effect on system growth is the obvious limitation on the amount of disc space available for data segments by the size of the system disc. The performance impact of this restriction is due to the long queue length created at the system disc when the system is under memory pressure, and the resultant service delays for access requests to that volume.

Disc access scheduling did not perform requests in the order of their urgency. The scheduling policy for disc requests directed at a device was preemptive for all memory management requests and FIFO for all other requests.

The memory management replacement policy selected segments deemed not likely to be needed in the near future. In the case of data segments, a write to disc of the segment was requested, the motivation being that the write may complete before the region occupied by the segment was required so that the delay in fetching the new segment would be reduced. These "anticipatory writes" were not urgent and often unnecessary, yet the scheduling policy selected them for service before disc access requests required for the completion of important transactions. Segment fetches on behalf of batch jobs were also serviced before transaction related service requests under the old scheduling policy.

The FIFO policy within process initiated disc access requests resulted in the accesses of less urgent processes being performed before those of more urgent processes. This increased the service time for the more urgent processes requests and thereby increased the response time for the related transactions.

Disc service was entirely serial for each disc sharing a common controller (i.e. no ovelapping.) Although the controller supports overlapped seeks, this feature was not exploited. This resulted in a disc throughput limitation per controller to one access per (avg cylinder positioning delay + avg rotational latency + avg transfer time).

MPE IV disc management solves these problems. Additionally, the general approach to disc management gives broad flexibility in scheduling policies.

In MPE IV, disc space for data segments can reside on each system volume. This multi-spindle virtual memory eliminates the limitation on total virtual disc space, and helps to balance the disc queue lengths. (Balanced disc queues are required to take advantage of parallelism in I/O offered by overlapped seeks or multiple controllers).

The MPE IV disc queues are priority ordered. The priority of a disc request is determined by the priority of the process that requires the transfer. This holds for segment transfer requests issued by the memory management system on behalf of a process as well as for file system initiated transfer requests. Anticipatory writes are given the worst priority and sit at the back of the queue so that they are performed as background activity when the device would be otherwise idle. This priority queue management integrates the disc management policies with the goals of the rest of the kernel, since priority assignments reflect the global performance goal of the system.

The feature of the disc controller which allows a seek command to be sent to a unit other than the unit owning the controller is exploited in MPE IV. The seeks for units waiting for the controller are issued during the execution of the channel program for the unit currently owning the controller. This results in the heads being in position over the proper cylinder when the next unit gets the controller. The net result is a potential maximum disc throughput per controller of 1 access per (avg rotational latency + avg transfer time). Since the disc 1 access time is dominated by the head positioning delay, this overlapping approximately doubles the maximum throughput per controller. (The overlapping seek software will only be available for the Series II and III on the C-MIT).

To achieve this maximum throughput, the disc queues for the units on the controller must be kept non-empty and balanced. This requires a sustained high level of multi-programming and a proper spreading of data across the volumes. To make response times short, the more urgent disc requests have to be performed first. It can be seen how the interrelations between memory management, processor management and disc management impact system performance.

8.2 Memory Management

Memory management requirements for the 3000 architecture consist of free space allocation, segment replacement, and garbage collection.

Free space allocation is required when a segment fetch is to be performed. The free space allocation algorithm selects the hole into which the segment should be read. Alternative strategies include first fit, best fit, and buddy schemes.

Segment replacement must be performed when a segment fetch is required but a hole of adequate size is not available. Alternative strategies include working set type policies and least recently used type policies.

Garbage collection is required in a segmented system to combine holes into larger holes. A variable sized allocation policy tends to produce small, unusable holes scattered throughout memory. This is known as external fragmentation. Garbage collection attempts to minimize the external fragmentation by combining the small holes into larger usable holes.

The major problems identified with the old memory manager were its serial nature, high fault rate caused by the per program working set replacement policy, restricted garbage collection performed during critical periods, and an inefficient free space allocation policy.

The old memory manager was entirely serial. Once the memory manager was started on a process swap-in, he couldn't begin on a second (or more important) swap-in until all the disc transfers required to finish the first swap-in completed. This serial memory management service forced artificial limits on the multiprogramming level. For large main memories this limitation restricts the system from achieving its potential performance.

The working set per program policy caused processes to release each others localities resulting in a high fault and recovery rate.

Garbage collection could only be performed locally within a bank, and performed during allocation time, so that memory management service time was further increased.

MPE III free space allocation selected the first fit hole causing large holes to be used up before they were needed. This resulted in excess invocation of the replacement policy.

The memory management policies were entwined with the rest of the system so that minor strategy changes would require extensive development.

MPE IV memory management solves each of these problems and in addition presents a general, structured implementation which allows major strategy changes with minor development effort.

Free space allocation is implemented by a best fit policy using size ordered free lists. This scheme is very fast, and saves the big holes until they're needed.

The resulting external fragmentation is eliminated through background garbage collection. Main memory garbage collection is performed as a background activity using cpu cycles during which the processor would have otherwise been idle. Garbage collection attempts to move small assigned regions located between large holes into small fragmented holes. The large holes are combined into even larger holes, and the small holes are eliminated. This skews the distribution of hole sizes towards the large holes and eliminates the external fragmentation thereby reducing the frequency of application of the replacement policy. The garbage collection code is responsive to the system state, and returns to the dispatcher when more urgent activity becomes pending.

The memory replacement policy is a very low overhead implementation of a global least recently used (LRU) policy. When a hole of the required size is not available, segments not needed by the current multiprogramming set (as determined by a global LRU algorithm) are selected for replacement on a memory ordered basis. In the segmented architecture of the 3000 family, this replacement policy proved to be superior to the working set policies. The memory scanned LRU approach tends to release unneeded segments in adjacent regions of memory, thereby creating large holes with few replacements. In contrast, the working set policies were found to require many more segment replacements to satisfy placement requests since they freed up space randomly through memory when releasing a working set of segments.

Segment fetching is an unblocked parallel operation in which memory management code invoked directly by the dispatcher sets up the operation and
the disc management code finishes it off as the required transfers complete.

8.3 Processor Management

Processor management consists primarily of selecting the activity to which the cpu should be devoted. The major cpu activities include running system and user processes, swapping in processes, and garbage collection.

Processor management is implemented by assigning priorities to the pending activities and giving the cpu over to the activity with the most urgent priority. This function is performed by the dispatcher.

Priority assignment in the old kernel had a problem with batch jobs.

Batch jobs would migrate up in priority to compete equally with interactive processes during busy periods.

Activity selection was restricted in the old kernel due to the memory manager being serial. The consequence of this limitation was that even if plenty of free space was available, memory management could not be performed when needed for a process if a more urgent process was waiting for disc I/O to complete. It couldn't be risked to swap-in a less urgent process since the more urgent process might need memory management service soon and the memory manager would be busy. The dispatcher was forced to pause the cpu rather than to work on increasing the multiprogramming level.

The MPE IV processor management scheme is very flexible. Priority assignments and activity selection are directed towards optimizing the system performance and can be tuned by the operator.

Priority assigmments are made to reflect the performance goals of the system. Each scheduling class (C,D,E) has a base priority and a limit priority. When a transaction begins or a job is introduced into the system, the related process gets its class' best priority, the class base priority. As the process uses more cpu time than that required for an average member of the class, the process is considered to be less urgent and its priority drifts towards the class's limit priority. The limit priority is the worst priority that a process in the class can get assigned to it.

The priorities of processes placed in the A or B scheduling classes are kept static over time. The filtering parameter which determines the migration rate for a C, D or E scheduled process from its class base to class limit is dynamically tuned for C scheduled processes only. Bounds on the filtering parameters for C, D and E classes are set in the :TUNE command.

This priority assigmment scheme enables the dispatcher to apply a scheduling policy which approximates a "shortest processing time first" algorithm. This gives maximum system throughput and best response time for short transactions while slightly delaying the longer transactions.

The C, D, and E classes can be made to overlap so that the processes in the various classes compete with each other, or they can be made disjoint. By making them disjoint, D and E processes will always be preempted for C processes. This tuning causes batch work to be performed as background activity between bursts of interactive transactions. This is the default tuning setting.

The operator or system manager can control the base and limit priorities of each class, and the rate at which a process' priority moves from the base to the limit through the :TUNE command.

CPU activity selection procedes by inspecting the priority ordered queue of processes requiring cpu service. When a process is encountered which is ready to run, the process is launched. If the process requires some memory scheduling, the swap-in procedure is invoked directly by dispatcher. If there's nothing better to do, main memory garbage collection takes place.

In MPE IV, swapping-in of a process is performed by nested procedures on the dispatcher's stack. The fetching of a segment on behalf of a process is a low overhead, unblocked operation which allows an unlimited degree of parallelism in memory management. The swap-in code is responsive to the system state, and returns to the dispatcher when a process more urgent than the one that's being worked-on becomes ready or requires scheduling attention.

If the queue of ready processes is empty and there are no processes requiring memory scheduling , or increasing the multi-programming level has been determined to be dangerous at this time, the dispatcher invokes the background garbage collection code which returns when more urgent activity becomes pending.

9. Performance

The performance of MPE IV as measured by system transaction throughput and mean transaction response time is published in the "HP 3000 Performance Guide for Installed Systems." Performance tests were conducted using a standard application workload which represented a general-purpose EDP environment with a mix of online data base and program development sessions and background batch jobs.

The measurement results from these tests indicated that under light loads relative to system configuration, MPE IV showed slight performance improvement over MPE III. This was anticipated, since the MPE IV kernel seeks its performance improvements through the exploitation of parallelism, and the potential for parallelism is small under light loads. As the workloads were increased, MPE IV showed substantial improvement in both transaction response time and transaction throughput over MPE III. The performance improvements were realized across the family under the configurations and workloads measured.

The behavior of the system was exactly that which was sought. The system exhibited stability and good performance across the range of workloads, processor speeds, memory sizes and system configurations examined.

10. Conclusions

The approach to kernel design and implementation undertaken by the MPE IV kernel project resulted in an operating system foundation which naturally fits the evolving 3000 computer family to the environments it supports. The structured approach permitted alternatives to be easily implemented, and the measurement interface permitted them to be thoroughly evaluated. The final implementation consists of integrated resource managers who cooperate to provide the best performance with the given system configuration under the current workload. The validity of the approach taken to kernel design is demonstrated by the resulting kernel's reliability and performance.

Acknowledgements

Special recognition is due to those who significantly contributed to the project's success. Professor Wesley Chu of UCLA and Professor Forrest Baskett of Stanford impacted the process through their valuable consultations. Alan Hewer, Howard Morris, and Neil Wilhelm kept things on course through their careful reviews. Ron Kolb, Ray Ventura and Bruce Blinn, through their design and development help on seek-ahead, the measurement interface, and multi-spindle virtual memory respectively, helped to speed the kernel to completion. Chris Moeller with his tuning help and Marcia McConnel and Carl Sassenrath with their debugging assistence contributed to the system's performance and reliability. Ken Spalding, through his coordinating function in the late stages, helped to get the system out the door.