CONFIGURING YOUR SYSTEM

Jeff Kell
University of Tennessee at Chattanooga

What? Me Worry?

System configuration is a highly technical specialty within the realms of systems programming for which there are rarely any concrete guidelines. At times it seems more an art than a science, as any basic configuration will function; to the casual observer there is no readily apparent difference between similar configurations. Only the systems person, eyes aglow like a proud father, seems concerned with such technicalities. With the HP3000, most configuration changes are unnecessary; so why bother?

One unique feature of the HP3000 is the lack of the requirement for an absolute system generation (from scratch) to bring up an operating system. There is no need for the time consuming generation step, the user must only supply the necessary parameters relevant to his installation. If the I/O configuration is omitted from consideration, the remaining items which may be changed rarely disable a system completely. With a few exceptions, the system will run if the I/O configuration is correct. To strengthen the apathy toward configuration changes, the default configuration (or SE supplied configuration) usually works.

A well configured system will outperform a poorly configured system. This benefit is derived from allowing sufficient resources to prevent shortages while trimming the surplus space to a minimum. In this manner, the resident (frozen) memory area is reduced, allowing more memory areas for segment swapping. Slight modification may not produce an obvious result, but overall configuration enhancement will, especially when the system becomes loaded. The results of unoptimized configuration fall into certain categories:

* Table sizes too large:
  - May overflow bank zero; system will not run
  - May waste main memory if table is resident; available memory is reduced
  - Will increase size of table if not resident, and swap time is increased proportionately

* Table sizes too small:
  - If critical, system will not start
  - If serious, increased job load may cause:
    + System failures (critical tables)
    + Job overloads (cannot start new jobs)
    + Failure to :RUN a program (out of resources)
    + Program aborts (unable to obtain resources)
  - If shared, programs are suspended until resources are free

Perhaps the easiest way to present configuration guidelines is to give the usage of each table in a clearer manner than the standard manual definitions. Most tables are user and/or program dependant. In this case the utilization (per unit) will be given for you to adjust to your particular installation. Other tables are relative to other factors, and their definitions will be more concrete.

Code Segment Table (CST)

The primary CST area is used to identify code segments which can be shared by more than one unique PROGRAM, not process (if three users are running EDITOR, there are three processes, but only 1 program). The only entities which can be shared between programs are SL segments (group, PUB, and system). MPE requires approximately 128 entries for itself, depending upon the installed software and I/O devices. I/O device drivers are also given a CST entry (one per driver). The remaining entries are used for user SL segments, one per segment per SL file. If a segment is present in multiple SL files, the same multiple number of entries will be required to support it. Only segments which are present in the system SL are truly shared across accounts. Insufficient CST entries can block the system from coming up (if very small), cause JOB OVERLOAD messages, and block additional users or programs from execution.

Code Segment Table Extension (CSTX)

The CSTX area is used to identify code segments which can be shared by more than one PROCESS, and are used to map the segments of a program. The CSTX is divided into blocks, mapped by the CST block table, one for each unique program (see Maximum Number of Concurrent Programs). The block for a program contains the same number of entries as there are segments in the program. Subsequent users of the program share the same block. NOTE: if the program uses non-system SL segments, these segments generate entries in the CST table - the CSTX holds only the segments which are physically contained in the program file.

When a program terminates and no other users are executing the same program, the CSTX block is released. Allocated programs are given permanent CSTX blocks which are never released (this is the actual function of :ALLOCATE). The number of CSTX entries required can be calculated as:

```
CSTX = (number of unique program segments)
     + (number of allocated program segments)
     + 15% (to allow for fragmentation)
```

Insufficient CSTX entries block additional programs from execution, usually resulting in a message referring to the CST table, not CSTX. The loader error messages are very vague on this point (see also Maximum Number of Concurrent Programs).

Data Segment Table (DST)

The DST is a rather complicated beast, consuming more entries than anyone seems to expect or can explain. It has been impossible to justify all active DST entries at any given time, but the general tendencies and known quantities of this table can be explored. By definition, there is one entry for each data segment in the system; however, many data segments are generated by the system which are not common knowlege. Approximately 50-60 data segments are used by MPE for system tables, depending on the configuration and MPE version. There is one stack data segment for each process in the system (including MPE system processes), and one entry for each segment acquired through extra data segment capability by user programs. This is where obvious rules terminate.

For each user on the system, without any UDC files (user, account, or system) in use, 4 entries are required just to log on. Of these 4, one is for the command interpreter stack, one for the job directory table (JDT), and one for the job information table (JIT). The other is not immediately obvious, unless it relates to the buffer areas for $STDIN and $STDLIST. If only one UDC is present, two more DST entries are required: one to hold the UDC command directory and one file system buffer segment for the UDC file. Each additional UDC file at any level requires an additional DST entry as a file system buffer segment for file itself. The typical UDC setup with a system UDC and an additional user UDC will require 7 DST entries per user JUST TO LOG ON (64 users = 7*64 = 448 entries).

Along the file system line, one segment is required for each open buffered file. No segments are used by NOBUF files, and some may be saved by MULTI or GMULTI files (which supposedly share buffers). KSAM uses one segment per open file per process, and IMAGE uses one global segment for each open database plus one local segment per DBOPEN per database per process. Since IMAGE works in NOBUF mode, the number of files in the database does not affect the count.

Communications subsystems also use data segments, but the usage will vary between subsystems. DS/3000 uses two segments for each line opened by the operator initially (one stack for DSMON, one control segment), but subsequent closes and opens only affect the control segment (DSMON remains active). Each DSLINE or remote session done by a user requires two segments (possibly a control segment and a file system buffer segment). In an informal approximation, MRJE used 10 entries to support one printer and one punch. Additional printers and punches cause extra processes and extra entries. It is reasonable that the output processes require more segments than just a stack, as they reference the configuration, job, and directory files as well.

In summary, the number of DST entries known is:

```
DST entries = about 50 for MPE tables
            + 1 per system process (stack)
            + 1 per system process extra data segments
            + 4 per user
            + 1 per user with any UDC (including system)
            + 1 per UDC file per user
            + 1 per opened buffered file per process
            + 1 per opened KSAM file per process
            + 1 per opened IMAGE database
            + 1 per DBOPEN per process
            + 2 per DSCONTROL OPENed CS line
            + 2 per DSLINE per user
            + 1 per user process (stack)
           + xx User extra data segment capability
           + xx other communications subsystems
```

Insufficient DST entries can block the system from coming up, cause JOB OVERLOAD messages, block additional sessions or programs, or cause "OUT OF VIRTUAL MEMORY" message on file open attempts (which is another misleading message, although it can be caused by virtual memory shortage).

Process Control Block (PCB)

The PCB table is used to hold information about processes in the system. One entry is required for each MPE system process, one entry per user (session or job), and one for each user process. Spooling processes are included along with MPE system processes, there is one spooler process for each spooled device. There is also a monitor process for each opened communication line, plus any additional control processes required for the communication subsystem in use. The number of PCB entries is:

```
PCB entries = Basic system processes
            + 1 per spooled device
            + 1 per open communication line
            + 1 per MRJE host
            + 1 per MRJE output device (printer/punch)
            + 1 per user (job or session)
            + 1 per user process (:RUN & process handling)
           + xx additional communications processes
```

Input Output Queue (IOQ)

The IOQ is used to hold I/O requests for all non-disc devices. If the number of entries is too low for a given configuration, the system may halt. One entry is used for most terminal read and write requests. One or more entries is used per communication line, and in some instances, by pseudo devices directed to the line. Spooled output devices may use up to 15 entries (10 for input spooling). A good approximation of IOQ limits is:

```
IOQ entries = 1 per non-disc assigned LDEV
           + 10 per spooled input device
           + 15 per spooled output device

256 maximum; may impose limit on spooled devices
```

Disc Request Table (DISKIO)

This table parallels the IOQ for disc devices, but the entries are of different length and format. One entry is required for each process disc request. A minimum of 60 entries is recommended. A process may issue more than 1 request (swapping, file system read ahead, NOWAIT). Insufficient entries will cause a system failure.

Terminal Buffers per port (TBUF)

Older MPE versions asked for this value in terms of a total, but the new configurator multiplies the given number by the number of ports (terminals) currently configured. On non-HPIB systems the buffers hold 30 characters each, HPIB systems hold 62 characters each. In character mode, only one buffer is initially allocated for a pending read; in block mode, all required buffers must be allocated. Note that the specification 'per port' is only for defining a variable total based on the current terminal configuration - the buffers themselves are shared by all terminal devices.

Block mode operations (such as V/3000) are seriously degraded by an insufficient number of buffers, particularly when concurrent read operations take place. A full screen transfer (1920 bytes) will take 64 buffers for non-HPIB, 31 buffers for HPIB. Note also that V/3000 is not limited to the physical screen, and transfers may be larger. This is particularly true of FORMSPEC, which reads the entire screen at once, not just the unprotected fields. A two-page form (50 lines, 4000 characters) requires 134 buffers on non-HPIB, 65 for HPIB. If block mode is used, the maximum buffers should be configured (the maximum is 255, regardless of the number given per port). Otherwise one or two buffers per port is sufficient.

System Buffers (SBUF)

System buffers are rather vague. Older versions of MPE used these buffers for logon and logoff messages, :TELL buffers, and other things. The current versions use these buffers for I/O error log parameters (intermediary to logging system) and for certain communications line operations. Although the recommended number is 8 in most manuals, some loss of I/O error reporting can occur when tape retries are encountered. Insufficient entries generate warning messages and abort the process responsible for the I/O operation. A better estimate would be 16-18 entries, plus 1 for each MTS line.

Swap Table (SWAPT)

The Swap Table replaces the memory manager table in older ver- sions of MPE, but its function is similar. The number of entries is based on the working set of a process (code and data segments) and is very application dependent. The manual suggests 4 per PCB, although other sources suggest 8, 12, or other values. This table should be watched in order to adequately approximate its size. The suggested value of 4 is considered somewhat marginal, but again, this is application dependent. Insufficient entries will cause a system failure.

Primary and Secondary Message Tables

These tables are used for communication between system processes, the primary table being used for critical messages and the secondary table being used for noncritical messages. Entries in both tables are 5 words long, and the suggested value of 25 is usually generous. A shortage of primary entries is obvious (causing a system failure), but a shortage of secondary entries only impedes the requesting process (which is not obvious or easily measureable).

Special Request Table (SPREQ)

This table is used "to temporarily buffer the parameters for segment expansion and to form the queue for devices waiting for a segment to arrive" according to the System Manager Reference Manual. The memory manager uses this table rather than its own stack to allow size changes. Now that the memory manager overlaps processing with segment I/O, certain things about special segment transfers must be saved (somewhere) for use after the transfer completes. The first case occurs when a data segment expands (a positive ZSIZE or normal expansion during the early life of a program). This occurs infrequently and rarely do requests arrive concurrently unless extensive use of ZSIZE is present in application programs. The second case is not immediately obvious, and the only clue is device related (possibly an I/O driver). The suggested value of 25 is usually generous.

Interrupt Control Stack (ICS)

The ICS is used to process most types of interrupts in the system, and versions of MPE beginning with MPE-IV use this area to process garbage collection by the memory manager. More space is used by these versions than was used previously. The suggested value is 512, and the monitored use of this area is often unpleasantly close to this value (from 475 to 496). One case of power fail recovery problems at UTC was attributed (but not concretely) to the ICS. UTC has subsequently used a higher value, although it has never been monitored over 512. The suggested 512 may be sufficient, but uncomfortable. Allowing some overhead is highly recommended for safety, and a value of 600 is suggested unless proven otherwise.

User Controller Process Queue (UCOP)

This table is used to pass information (requests) to UCOP by other entities. UCOP is documented to manage the initiation and completion of jobs and sessions, and the system device recognition process DEVREC presumably sends initiation requests to UCOP when :HELLO or :JOB is encountered. UCOP also performs other functions as well, such as certain :REPLY operations (a process waiting on a response from PRINTOPREPLY is in a UCOP wait state), deleting processes, changing priorities, and stack size manipulation. The suggested value is 2 per user, with more for process handling. This value has proven to be generous for our installation, and we do an exceptional amount of process handling. Two per user appears to be a maximum, but only 2 words are required for each entry anyway.

Timer Request List (TRL)

This table is used for all timer related functions such as memory logging, the PAUSE intrinsic, timeout and timed reads (terminal and IPC), logon time limits, modem turnaround, and certain timeouts for communication lines. The suggested value is 1 per terminal plus 1 per user, but 1 per user is usually sufficient unless there are a large number of modem ports or communication lines. Standard hardwired configurations can use 1 per user plus 10% for most applications.

Breakpoint Table (BKPT)

This table is used by DEBUG for setting breakpoints in programs. A minimum of 8 to 16 entries will allow for adequate debugging, but more should be configured if DEBUG is used extensively by more than a few users.

Maximum User Logging Processes/Users per Process

These two values determine the number of user logging processes which can be defined and the number of users which may access a given process. Usually one process per application area using logging is sufficient, and the number of users per process is the maximum number of users of any single process (not the total number). This really limits the number of processes, not users, which open the logfile. A large number of users per process increases the space required for user logging control (once initiated).

Number of RIN's

This value determines the number of available Resource Iden- tification Numbers (RIN's) in all (global, local, and file). One per user is usually sufficient to allow for file locking (more if MR applications exist or global RINs are used). IMAGE no longer requires RIN locks to accomplish database locks, thus the 'one per user' is especially true.

Maximum number of concurrent sessions/jobs

These values determine a maximum for the :LIMIT command, and their total determines the size of the Job Process Count table, used when a CPU limitation is imposed on a job. Although not obvious by its query, this value does allocate space.

Logging status (general)

Most installations either allow all logging, or disable all logging. In practice, I/O errors and certain other infrequent status events should be enabled for diagnostic purposes. Logging everything for no purpose clearly burdens the system with unnecessary work. In those cases where logging is important for accounting or security, many logging types are beneficial. Frequent events such as file close and process termination should be questioned for merit, especially in transient environments. File closure, in particular, generates a great deal of data at the completion of every process. System related programs such as EDITOR and the compilers may generate anywhere from 4 to 10 or more records for each completion (a compiler has a record for text, USL, list, master, and new file; COBOL also has work files, symbol table overflow files, and copy libraries).

Virtual Memory

Virtual memory allocation is best spread across volumes, although one could argue merits of other schemes. The technique is not the purpose of this discussion, so it will be left to rest. If virtual allocation is spread, however, it should be reasonably close to the amount actually used; otherwise the used area may reside on a single volume and probably will be on the system disc. Virtual memory is only used to store data segments, not code segments. Code segments are swapped directly from program and SL files.

The amount of virtual used will be, at a minimum, the size of all data segments. Each segment requires a number of sectors equal to the segment size in words divided by 128, adding one unless the remainder was zero. The size of a data segment is fixed by GETDSEG, and the size of a stack is the MAXDATA value plus the system defined area at the beginning of the stack (PCBX). Some shops use a large default stack size, and MAXDATA is allowed to default (See section on Standard Stack Size). Insufficient virtual memory causes any process requiring an additional segment to be aborted (:RUN, process handling CREATE, and file opens).

Maximum Number of Concurrent Running Programs

This value determines the size of the CST Block table (CSTBLK) used in conjunction with the CST Extension. This is NOT related to the number of processes or users, but refers to the number of unique programs which may execute concurrently (additional users of the same program do not count). Insufficient numbers cause :RUN to fail, usually with 'Unable to obtain CST entries' error, which does not immediately indicate the problem. This value does not necessarily have to be large enough to prevent the message - it is frequently necessary to limit this value for performance purposes. Additional users of a program already loaded do not impact the system as much as new programs do. Note that :ALLOCATED programs take up an entry, as :ALLOCATE simply assigns a permanent entry to the designated program. The system will add 8 to the value you define to account for system programs used by MPE.

Maximum Code Segment Size/Maximum Code Segments per Process

These values place restrictions on programs. Once upon a time, HP said 4K segments were adequate for all of their products, but this value has increased to 8K. Arguments can be made for and against larger segments, but it should be noted that there are contributed library programs which have segments larger than 8K. In a controlled environment, it may be adviseable to set this value higher and limit the size by individual control. In an uncontrolled environment, this is not practical, and the limit should be set to 8K. The number of segments per process is suggested to be 63, smaller values may cause problems with large products (compilers, etc.). This only limits the eventual size of a program, and has little if any impact on performance.

Maximum Stack Size

This value limits the size of a stack, not to exceed 31232. Smaller limits may be argued to help performance by decreasing swap time, but many products (V/3000, SORT) are either adversely affected or cannot run altogether with smaller stacks. As with code segments, it is adviseable to set this value to its maximum and control sizes by some manual guidelines.

Maximum Extra Data Segment Size

If this value is large, and little control is exerted over data segment management, this may lead to performance degradation. This value affects IMAGE directly by limiting the buffers available in the global control block. If this size is too small, IMMAGE may not be able to allocate enough buffers to be helpful (the segment allocates a certain amount for control information and lock information - this amount must be subtracted from the maximum size to determine the space available for buffers). If this segment limit is too low, it will be impossible to open buffered files with large block sizes. HP recommends 8K for some reason, but IMAGE and extra data segment capability are the only areas which can misuse this limit. As with the previous limits, it may be adviseable to configure the maximum limit and control the actual use by other means.

Standard Stack Size

This parameter has perhaps the greatest misuse potential of all in terms of performance. It is used to supply a system default for the initial stack size (Q-Z) if the user does not override it. Some installations will set this to a large value (4K or more) to avoid run time STACK OVERFLOW and SORT: INSUFFICIENT STACK SPACE errors. Most programs, except for V/3000, SORT, and MERGE applications, will run with a default stack between 1600 and 2000. This situation deserves clarification.

When a program is loaded (with default STACK and MAXDATA), the data area defined by the program globally is already allocated. This data goes into the DB area, and the final position, plus one for PARM and a variable area at run time for INFO, determines the location of Q. Below DB, 128 words are reserved by default unless a DLSIZE is given. Below this, the system has a reserved area. This system area is not included in any specifications, but is subtracted from the 32K possible words to derive the 31232 maximum stack size possible. The area above Q is used for procedure storage (or for any called subprogram data in COBOL). Typical FORTRAN, SPL, or PASCAL procedures will not use much local storage, but COBOL is more likely to use a bit more. Most standard MPE intrinsics do not need a great deal of storage either. FOPEN is a typical case taken as a maximum, requiring around 1200-1400 words (more for labeled tape and laser printer, and remote file access). The point to be made is that a default stack size of about 2000 words (suggested in manual) is more than adequate for most applications.

If MAXDATA is not specified, it defaults to:

```
MAXDATA = DLSIZE (128 if not specified)
        + DB-Q area (from program)
        + STACK (uses Standard Stack Size if not given)
```

The Z register is planted at the end of the STACK area, and the final size MAXDATA + (system reserved area) determines the data segment size of the stack. This same area is reserved in virtual memory for the stack. Swapping occurs from the beginning of the segment to the location of Z. If MAXDATA is specified, Z is still placed at the same location, but virtual memory is reserved for MAXDATA. In other words, the segment swapping is directly related to STACK, not MAXDATA itself. During execution, if the stack reaches Z, it will expand automatically if more space is available, but will cause overflow if MAXDATA is exceeded.

SORT requires additional stack space for a work area. V/3000 uses a good deal of space in the stack as well, but also uses the DL area. Both products may fail with a default stack and no override MAXDATA. Changing the STACK parameter will help SORT, and usually help V/3000, but MAXDATA does the job also. If the installation standard stack size is changed to help these areas, everyone suffers. If a program does not need this extra space, it is not only wasted in memory, but the unused space is also swapped.

MAXDATA is better than STACK, and should be used instead. although not recommended for everything, it would be better to have a system UDC to PREP everything with a large MAXDATA than to change the default stack. MAXDATA wastes only virtual memory, but STACK wastes virtual memory, real memory, and swap time. We firmly believe that a standard stack size of 2000 (or slightly less) is adequate. In other terms, it is better to override small defaults for large sizes than to override large defaults for small (programmers will not go to the trouble to change something that doesn't abort their program).

Maximum Number of Open Spoolfiles

This parameter is asked before any tables, but listed here to keep the obvious tables together. Usually no connection is made between spooling parameters and memory usage, but this parameter does indeed allocate permanently resident storage. Most installations will use a high value for spool file kilosectors so that spooling is limited only by available disc space. This practice is acceptable, but the value for open spoolfiles should be minimized. Spool files in the READY or ACTIVE state are unaffected, only those OPENED at a given time are affected by this parameter.

When a spool file is opened, MPE sets up a dummy device in the location of an unused logical device number. This parameter tells MPE how many dummy device numbers to reserve for spooling. Each entry requires 8 words, and the maximum is determined by the number of real logical devices on the system:

```
                  16244 - (# of LDEV's * 4)
Maximum value = -------------------------
                           30
```

Configuration Table Parameters Summary

Table Name	Entry Size	Resident	Min	Max	Insufficient
DST	4 words	Yes	192		Oflow; load
CST	4 words	Yes	93	192	Can't boot
CSTX	4 words	Yes	32		Oflow; load
PCB	16 words	Yes	?	256	Overflow
DSKIO	16 words	Yes	?	255	SF601
IOQ	11 words	Yes	20	256	Impeded
TBUF	16/32	Yes	?	256	Impeded
SBUF	129 words	Yes	?	?	Impeded
SWAPT	5 words	Yes	?	2048	SF602
Primary Msg	5 words	Yes	?	255	SF620
Secondary Msg	5 words	Yes	?	255	Impeded
SPREQ	5 words	Yes	?	255	SF600
ICS	given	Yes	?	?	SF/Halt
UCOP	2 words	No	?	?	Impeded
TRL	4 words	Yes	?	?	Fail
Breakpoints	5-12	No	?	?	Can't set
# User log procs					
# Users per/log					
Rins	2 words	No	5	?	Can't use
Global Rins	12 words	No	?	?	Can't use
Max Sess/Max Job	1 word	Yes	?	?	Overflow
Max Programs	?	?	?	?	Can't run
Max open spoofles	8 words	Yes	?	?	Can't open

Brief Biographical Sketch of Jeffrey R. Kell

Jeff Kell, 25, has been in data processing for eight years, with a background in systems programming on IBM equipment for three years, and five years with Hewlett-Packard equipment. Jeff has been with the University of Tennessee at Chattanooga for six years, and serves as senior systems analyst for Administrative Computing Services. In addition, he maintains configurations, backup, and recovery for the university's five HP-3000 computer systems.