SIGMA SERIES COMMUNICATION

SUBJECT: Sigma 5/7 Multiprogramming System DOC. NO.: 384

AUTHOR: Tom Melton, Jim Gaines DATE: December 10, 1969

DISTRIBUTION: Sigma Notebook Holders FILE NO.: 14

OBJECTIVE

The objective of the Sigma 5/7 Multiprogramming System is to provide for the full solution to a user's data processing problems. It stresses the availability of a spectrum of services--multiple batch streams, remote job entry, real time and interactive time-sharing. Even though the user may never attempt to use all of these services, their availability guarantees him a growth path regardless of which services satisfy his immediate needs.

GENERAL DESCRIPTION

The planned facility for the Sigma 5/7 multiprogramming batch system can be described as an unmapped, fixed partition system supporting any given number of partitions as determined at system generation. The maximum number of jobs that can be run concurrently is fixed at system generation. The system is describable in the same general terms as the MFT version of OS/360.

Partitions will be general in nature. Processors, utilities, and user programs alike will be executable in any partition large enough to contain them. In practice, each program can be formed to run specifically in one of the partitions (i.e., each can have its own "home partition"). The design allows any particular program to be directed toward its home partition. When scheduling conflicts occur and the program can be scheduled to run elsewhere, the system uses the program's relocatable form to relocate it into another designated partition. The user can be charged for this extra system facility; it is used to his benefit within the rules set up by the DP manager at system generation.

The multiprogramming facility will be available under BTM as well as in a batch only version. Symbionts are an inherent part of the system. It will be a completely compatible superset of existing BPM or BTM batch facilities. That is, jobs prepared for batch operation under BPM/BTM can be run without modification if they require only the default allocation of resources. Additional resources will require additional control commands. It will be a completely compatible subset of the UTS based Sigma 7 multiprogramming system. It will neither require nor be able to use the map.

Monitor residency will require a minimum of 16K words and BTM residency requirements will add to this at least another 8K. The high speed RAD will not be required. Additionally, the system will support real time and remote batch, concurrently as required by the user.

Principally, this system provides improved batch throughput due to over-lapped use of the CPU during disk and tape I/O as well as improved use of critical resources such as disk packs and magnetic tapes. When one program is waiting for an I/O request to be completed, the CPU will be directed to a program which can use its services immediately.

Jobs entering the system will be selected for processing according to "Class" and priority, relative to available resources. The relationship between "Class" (see details below) and physical resources is determined at system generation. Class is added to the scheduling system to provide the DP manager with the tool needed to relate his needs to the total resources of the system. A general point to be made is that once a job is scheduled and begins execution, it will not be rolled out if a job of higher priority enters the system. In a normal situation, once a job begins it proceeds to completion.

Jobs will be scheduled on a job rather than on a job step basis. Resources of a special nature that will be required for a given job must be specified at the beginning of the job and must be the maximum of each such resource that the job will use.

It will not be possible to allocate added resources after job initiation. Peripheral resources can be released for the remainder of the job by user program and/or control command between job steps.

All jobs input into the system (via BTM, Remote Job Entry, etc) will be scheduled on the same basis regardless of their point of entry. Preferences must be handled through assignment of class and priority.

The job originator can specify that one or more other jobs must have been completed with specified termination type before this job is run. This "predicate" relationship allows the breaking up of jobs with widely varying resource requirements into more economical processing units.

Private volumes required for the execution of a job must be stated so the scheduler can avoid otherwise unresolvable conflicts among jobs which require exclusive use of the same private volume(s). Also, all files on public volumes for which a specific job will require exclusive use must be specified so the scheduler can avoid lockout conflicts. Job scheduling will include consideration of exclusive file usage requirements complete with file names and perhaps mode of use. Jobs with the same account number will not be scheduled concurrently.

Run-time tasking will not be supported.

The minimum configuration will be:

Sigma 5 or 7 CPU
40K of Core
6 Megabytes of RAD
And the remainder of a minimum BPM configuration.

The minimum core requirement of 40K yields a maximum partition of 24K, satisfying the core requirements for any standard processor currently available. A 32K system (16K maximum partition) will be feasible for those installations that only use processors which operate in 16K or less (Fortran IV-H, Symbol, etc.).

PARTITIONS

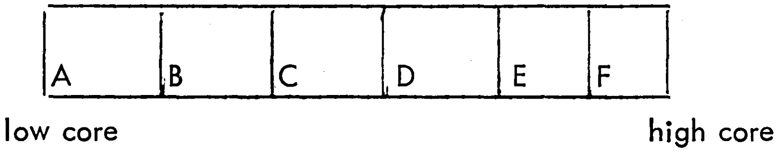
The number of partitions and the characteristics of each are specified at system generation. The number of partitions that are defined determines the number of jobs that can be concurrently executed. The attributes of a partition are the specific area of core residence, the set of processors that are biased within the core residence area, the number of non-sharable resources that are guaranteed to jobs that will execute in this partition, and a prioritized list of "job classes" that are allowed to execute in it.

core considerations

The size of a partition places it into one of two categories -- scheduling or nonscheduling. If it is 8K or larger, it can physically accommodate the Job Scheduler (an expanded CCI) so jobs can be scheduled for it at any time; that is, it is a scheduling partition. If less than 8K (i.e., nonscheduling), the partition must wait until the next use of the job scheduler in a scheduling partition. Any time the Job Scheduler becomes active, all schedulable partitions, large or small, will be serviced by it including the partition in which it resides. The minimum partition size is 2K. The operator has the facility to disable, enable and append to partitions and can reallocate peripheral resources. He can, by disabling a partition, cause it to accept no more jobs at the end of the currently operating one.

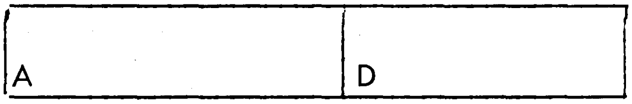
He can, by enabling it, cause a partition to begin accepting jobs. Those jobs that can run only in a currently disabled partition are "blocked." The operator can reallocate the core resource of a disabled partition by appending it to the next lower partition. The operator can later restore the partition and return it to its former enabled condition. Any number of contiguous, higher in core partitions can be appended to a partition. More than one enabled partition can be appended to in the above manner. For example:

Partitions Before



If the operator were to append B, C to A and E, F to D, the enabled partitions in the system would be:

Partitions After



Only those classes and resources of A and D would now be active. In particular, these facilities can be used to build up the BTM partition from several smaller ones. This allows a very flexible use of the BTM user space when BTM has been activated.

biased processors

The system provides for the processors to be biased at more than one partition; this implies multiple copies of the absolutized processors. When set up to do so and when scheduled, relocatable versions of the processors will be used in partitions for which no absolute copy exists. The user may be chargeable for the relocation time and resources so the system must keep separate track of them for accounting use.

non-sharable devices

A set of peripherals of each type available can be dedicated to a partition at system generation. This is an assignment of the number of such devices required, not the physical devices. Those devices remaining (i.e., the total number of a type in the system minus the total number assigned to all partitions) form a pool from which the scheduler draws to satisfy additional peripheral requirements.

The operator can change these assignments. He can reallocate to and from any partition to any other partition and/or the peripheral pool. These reallocations are permanent as if they had been assigned during system generation though they are not written to the system device to supplant the system generated copy. In the event of a crash, an attempt is made to use the current allocations. If this is not possible, a complete system reboot returns the system to the system generated version.

prioritized job classes

Fixed core partitions imply the need for scheduling jobs into those partitions that fit as nearly as possible to avoid wasted resources. For example, if core were partitioned into areas A(8000 words) and B(4000 words), all large jobs (larger than 4000 words) should be run in A (a jobs) and all small jobs in B (b jobs). If no a jobs were available, it would be very efficient to run b jobs in partition A. That is, in partition A, jobs should be selected according to the priority; a jobs first, b jobs second. In partition B, only b jobs would be selected.

In general, one could say that though jobs of the same type might have preference for one partition, they should be executable in more than one partition. This idea forms the basis for providing job selection in this system from a prioritized list of job classes that can be executed in a given partition. Class makes it possible to group jobs having similar requirements. By associating, at system generation, the prioritized list of the job classes with each partition in which they can be executed, the DP manager has complete control of system efficiency in the overall use of system resources. Additional control is available with operator key-ins to enable or disable a job class for a specific partition.

JOB CLASS

Class is the quality assigned to a job that reflects how important it is relative to all system resources. Though the operator has many controls that affect class, its properities are principally defined during system generation. The classes can be ordered for selection priority across the whole system or within each partition. The latter implies that for partition M with classes a and b and partition N with classes a, b, c, class a could be higher than b in partition M but lower than b in partition N, or class a could be a very important class and be of highest selection priority in both M and N. Service priority (i.e., which job gets CPU service next) between active jobs is class dependent, too. So if class b jobs are the most urgent or if they must be serviced often because they produce frequent I/O calls and thus help balance the system, then class b can be assigned the highest service priority in the system. Then no matter in which partition a class b job might be executing, it has the highest service priority. The next most important classes can be assigned priority as required.

Jobs are selected for execution with the ordered triple: (Class Selection Priority, Job Priority, Time in the Queue). Class Selection Priority is a number (as is Job Priority) that shows the selection preference among schedulable jobs for a given partition. Consider the unordered set of partitions, M and N, and an unordered set of classes, a, b, c, associated with them representing three kinds of jobs to be handled in the system. To choose the class selection priority, the DP manager will assign priorities for each class to be associated with each partition.

M	a, 2; b, 1; c, 3
N	a, 1; c, 2

In this case (assuming 1 of greater priority than 2), when partition M becomes empty the next job selected is of class b if available, a if b is not available and c if neither is available, job priority and then time in the queue are determinants for scheduling.

Once selected for execution, the service priorities of the executing jobs determine the order in which they receive CPU service. The run-time scheduling of the CPU will be on an I/O request to I/O request basis. A maximum compute time quantum will be used to force redirection of the CPU from compute bound jobs to maintain peripheral utilization. If more than one job of equal priority is awaiting CPU service, preference will be given to those jobs who requested I/O during their last activity.

The system generation class parameters include:

- Job priority minimum level; time duration for promotion considerations.

  After any job above the specified priority level has been in the scheduling queue for longer than the specified time duration, it will be raised in priority during each subsequent scheduling cycle until it is scheduled or reaches the highest job priority. At this time its condition will cause the system to begin shutting down an appropriate partition until it is the only job that can be scheduled. The shut down will be of the nature of temporarily removing all other classes from the selected partition and so on. The promotion parameters that will be used are a function of the job class.

- Service Priority
- Compute Quantum

  Associated with each class is a time limit used to prevent the tieing up of system resources by totally compute bound jobs. Exceeding the quantum causes a new, normal service cycle to begin.

- Default and Maximum Limits for the Class

  Maximum limits give the DP manager control over the type of jobs that are allowed for a given class. Default limits state the typical demands made on the system by jobs of this class.

  Cards in, out
  Pages out (LO, DO, UO)

Job duration (sum of CPU and I/O time)
Devices required for dedicated usage (magnetic
    tapes, disk packs, etc.)
Core size
Blocking buffers (default only)
Public file granule usage
    Total space allocated
        RAD
        Disk packs
    Permanent space used
        RAD
        Disk packs

JOB SCHEDULING

Job scheduling is initiated whenever the system finds a partition in an enabled, inactive state. Job scheduling goes through the process of selecting the next job that will be executed in an inactive partition based on the following considerations:

1. Job class
2. Priority and age
3. Predicates' status
4. Resources required
5. Conflicts with currently executing jobs

Normally, job scheduling for a partition will be initiated by job termination in that partition. Once service of the job scheduler begins, jobs will be considered first on the basis of the job class priority associated with this partition. If no jobs exist in the queue for any of the job classes assigned to this partition, the partition becomes inactive. If any jobs are available the first one considered is the oldest job with the highest job priority (from the job command or modified by promotion) and the highest job class available to this partition. If this job cannot be scheduled because of resources (dedicated peripherals, core, account number or volumes conflict with current jobs), then the next oldest job is considered for execution. If none of the jobs in the highest job priority level can be scheduled, then jobs of the next highest job priority level are examined for a schedulable job. This process is continued until all of the jobs in the highest job class priority associated with the partition have been exhausted. The above search for a job that can be scheduled will be repeated down the prioritized list of allowable job classes for this partition.

Jobs will not be swapped out of core by the system to provide facilities for another, higher priority job. Consideration should be given for making available, under operator control, core roll out of specified partitions.

Jobs can be in one of the following categories:

Arriving Jobs

These are jobs which have been introduced into the system since the last scheduler operation.

Jobs awaiting job predicates

These jobs are awaiting the completion of predicate jobs. Those jobs in this category are considered "blocked" if one or more of their predicates are not currently in the system.

Jobs blocked by system

These jobs have core and peripheral requirements that are within the allowable limits of their associated class but cannot be satisfied by the system in its current configuration without operator control.

Special priority jobs

Jobs in this category are those that have been promoted (either by time or operator control) to the level that resources are being collected for them at the expense of system performance.

Schedulable Jobs

These jobs make up the normal selectable job mix for the job scheduler.

Executing Jobs

These jobs represent the ongoing, concurrent workload in the system. All of the necessary resources have been allocated to each job in this category.

JOB PREDICATES

The predicate relationship between jobs is necessary to allow the entry of two or more data related jobs into the job stream on an independent basis and still assure a specific order of processing. An example of the type of situation for which the facility is necessary is the weekly production of payroll checks which is dependent on the introduction of the weekly time cards into the system. Another important use of the predicate relationship facility in this specific system will be to allow breaking up of a job which is comprised of a number of job steps which require very different resources for their execution. For example, a job which consists of two serial job steps, the first of which requires a large amount of computation and no magnetic tape units and the second of which involves a small amount of computation and 8 magnetic tape units. The reasonable thing to do would be to break the job into two job steps with the second predicated on the first, so that the magnetic tape units would not have to be tied up during the processing of the compute bound job step.

The predicate relationships will be indicated on a control command. A unique identification must be supplied on any job that is a predicate to another job. The control command will allow reference to the identification of each job predicate and an option specifying the mode of completion of the predicate job that will be required to satisfy the relationship.

The installation will be capable of setting a limit to the length of time that status on predicates will be maintained by the system. In addition, the capability to delete the status information will be available by either operator key-in or control command.

CONFLICTS WITH CURRENTLY EXECUTING JOBS

A job can require files, data or peripherals for exclusive use that would conflict with the requirements of other jobs. The situation with files is one in which a given job is going to update a file of related information such that exclusive record use would not, in general, allow logical, concurrent updating of the data base by two or more independent programs. An example of this is the possible independent updates of the same file by programs from two sources, where a predicate relationship is important but simply connot be implicitly determined ahead of time. Since there will also be jobs which will legitimately wish to update the same file concurrently, the discrimination between the cases will have to be resolved by an explicit indication of the names of all files and volumes for which each job will be demanding exclusive use. The scheduler will then avoid scheduling jobs with exclusive use requirements for the same files and volumes. It would not be possible to resolve all conflicts as they occur at run-time because of possible hooking problems, where each of two jobs would require a file that the other already had.

In addition to the file access interference problem, two jobs that require utilization of the same removable disk or tape volumes obviously cannot be scheduled for simultaneous execution. The same is true of any jobs which utilize the same dedicated specific piece of hardware, such as a remote batch terminal or a specific line printer.

These problems of conflict will all be solved by requiring user inputs on control cards.

ACCOUNTING

The accounting system will have to be expanded to take advantage of the fact that different peripherals should be charged to the people who keep them tied up, in order to allow installation control, by billing, of conservatism in stating requirements and to improve the efficiency of utilization of the system. The data that must be added to the existing accounting information includes the product of core size used and the time it was used, the length of time private disk pack and tape volumes are required by a given user, and the total amount of CPU time used. This information is all output information which will not affect the user's program or job stream in any way.

When the user-specified class priority causes the system to relocate a processor, utility, or user module, this increment of system facilities must be billable in the user's accounting charges if desired by the installation.

OPERATOR CONTROL, SYSTEM CONTROL MESSAGES AND MONITOR CALLS

Because a multiplicity of jobs will be executed concurrently in the multiprogramming system, all operator input and system output messages will have to be job oriented with job identification included in the message. This will affect operator input message formats, but will not affect user programs or job streams in any way. All job associated control commands will have to be recognized as such by the system which will have to be able to cope with a variety of request sources for debug commands, assignments, etc. The system will also have to cope with monitor calls which must be job associated to be meaningful, such as ABORT.

SYMBIONTS

The symbiont catch up facility will have to be disabled in the multiprogramming system. The facility under discussion involves the ability of the symbionts to allow direct device I/O when the symbiont queues are empty. That means that the symbionts try in the existing system to output directly to the line printer, for example, while a job is running if the symbiont queue for the line printer is empty, rather than waiting for the executing job to complete the symbiont output file before anything goes out to the line printer. The capability to initiate the output of a running job to a device will be available only by operator key-in. This will allow the operator the ability to control peripheral utilization over those jobs producing voluminous symbiont output.

In the multiprogramming system nothing should be automatically done with other than complete symbiont files, and no direct I/O through the symbionts should be provided. That statement is necessary because in a multiprogramming system there are a number of sources for the creation of output. Peripheral initiation for an arbitrarily chosen job, because the queues are empty at one point, may mean running at reduced output speed for the duration of that job. The alternative is to do nothing until a file is complete and then run the peripheral at full speed through that file, and then the next, etc.

In addition, it will be necessary to be able to define the destination address of the output device for a symbiont file, at least in a categorical manner. For example, in a hypothetical future system the three line printers may be of different types with variations in speed and print quality. It will be necessary to allow a user to optionally specify to which printer the symbiont output of his program should be directed.

It will also be necessary to provide the facility within the system which will allow the user program to specify that a given form be loaded into the printer before a specific symbiont output file is printed. That feature should be implemented so that an operator message is generated when the symbiont output file is selected for printing and assigned to a given printer.

The normal mode of operation of the symbionts should be to pool symbiont output files and assign printers on an as available basis. A desirable extra would be the scheduling of output of the files in the pool based on the priority of the program which created them.

The symbiont capability of the BPM system must be expanded to include the facility to handle generation of output for the same device from multiple sources concurrently without merging of the data. Each of the sources would be in a separate job. An example of the facility required is illustrated by the necessity to allow output through the symbionts to a line printer from a number of jobs which are being processed concurrently.

FAILURE AND RECOVERABILITY FEATURES

The development of error detection and recoverability features in the BPM/BTM software (and UTS where applicable) must result in the same facilities being made available in the multiprogramming system, including both the BTM and Batch-only systems.

REAL-TIME IN THE SYSTEM

Real-time jobs will have to be supported in the multiprogramming system in the same way that they will be supported under BPM/BTM. Such jobs will be pre-emptive in nature and can be introduced only with the understanding by the installation manager that they will have an effect on performance.

SCHEDULES AND MANPOWER

The multiprogramming system must be available to the field in the second quarter of 1971.

XDS Memorandum

TO	Bob Spinrad			DATE	21 April 1970
FROM	Dan Cota	EXT 1441	MAIL STA A1-02	REF	PR-70-1029
SUBJECT	Multiprogramming				

1. Introduction

This memo presents Programming Development's analysis of alternative ways of obtaining a multiprogramming batch capability for Sigma 7. There are several approaches, of which no one is really "correct". But we must choose one, implicitly resolving several important -- yet conflicting -- issues.

There are two major, competing views with respect to availability of some form of batch multiprogramming. One view is that all we need is something called multiprogramming. The other is that any system called multiprogramming must satisfy a number of BDP requirements. There is such a thing as "scientific" multiprogramming. We can produce a multiprogramming batch system rather quickly. But such a system will be absolutely useless (and not salable) in any commercial environment. It will take much longer to produce a multiprogramming system with even a modicum of customer acceptability for BDP environments than it will to produce one for a scientific environment. Also we must remember that diverting resources to produce a scientific multibatch system will delay proportionately the delivery of a BDP-oriented multibatch system.

Serious doubts exist about the need for scientific multiprogramming. The people who really must have more than real-time (non-resident, operator initiated, externally scheduled multiprogramming) plus batch -- which we currently have -- probably need the features and capabilities that we ascribe to a BDP system. No quick and dirty multiprogramming system is going to help us get into new markets. Such a system would have minimal value in solving BDP problems posed by our current markets.

Next, there are questions regarding the levels of disc pack support and file management enhancements that must be included in the first release of batch multiprogramming. Examples are CRAM and removable disc pack support, the keyed file speed-up, and other major portions of BTM/F00 or UTM/B00.

Finally, there is the question of undertaking a dead-ended development. Any short-term, quick and dirty system will not be compatible with either future releases of UTS or with any of the releases of XMS. Producing such a system inexorably forces us to continue its development. A dead-ended effort ought to be avoided, because we will have it forever.

Copy to: G. Boyd, E. Bryan, B. Doeppel, F. Haney, D. Heying, D. Keddy, B. Mallonee, C. Martin, T. W. Martin, B. Reid, D. Reilly, B. Sharpe

The next four sections present four ways of obtaining multibatch capability; each alternative forces a resolution of the issues raised above. This exposition is concluded with a specific recommendation.

2. Alternative One: Quick and Dirty Multiprogramming

This is what we can cook-up in order to provide multiprogramming in the shortest possible time. It is the absolute minimum form of multiprogramming that can be tacked onto UTS.

FEATURES:

a. No resolution of file contention.

b. Job (not job step) oriented.

c. Only one job per account permitted in batch at one time.

d. Conflicts between batch and on-line use of files treated as they are now treated in both BTM and UTS (no shared access if update).

e. Jobs that reference files that are logically unavailable are supplied an abnormal return. (This occurs upon any OPEN of a file already opened in an update mode or upon an OPEN for update if the file is already open.)

f. Disc pack and file management support restricted to BTM/E00 level.

g. Jobs will be aborted if disc space is requested but is not available.

COMMENTS:

This is the quickest system that can be built. It is probably dead-ended. It diverges from UTS and from XMS. It doesn't offer removable disc pack support, and carries into the field the UTS/A00 level of reliability. It also would extend UTS problems into the batch domain, with concommitant pressure to extend, enhance and generally improve the system. It delays XMS in whatever its first form will be by about half a year.

SCHEDULE:

Development would take about 14 weeks from project start to Q.A. turnover. Earliest start is August 1, inasmuch as the extensions must be built upon the final Development turnover of UTS/A00 to Quality Assurance. We could, of course, delay all of UTS/A00 by three or four months and release this as part of UTS/A00. Adding a minimum of two months for Quality Assurance, field delivery becomes January 1971.

3. Alternative Two: A Severely Restricted First Version of XMS

This is similar in scope to alternative one. However, it presumes that a minimum subset of XMS can be defined with respect only to multiprogramming. It is undertaken on a "crash" basis, and is built upon UTM/A00.

FEATURES:

a. Resolution of file contention as required by XMS.

b. Clean operation of multiple jobs under a single account.

c. Job (not job step) orientation.

d. Clean handling of disc space allocation.

e. Disc pack file management support restricted to BTM/E00 level.

f. No UTM/B00 reliability or functional enhancements.

g. No symbiont or remote batch enhancements.

COMMENTS:

This is a terribly weak first version of XMS. In fact, since XMS must be advertized and sold as a BDP monitor, this system cannot be labeled as XMS. With respect to the first system, this one has the distinct advantage of not being dead-ended. It can evolve from what it is -- UTS/A00 with multiprogramming -- into XMS. But it isn't what we's call XMS. There are large drawbacks. If undertaken at all, the first release of the real XMS is delayed by at least nine months. Since it is undertaken on a crash basis, no time can be spent worrying about X1, symbionts, and the like, which almost guarantees incompatiblities and obstacles which must be overcome when the initial system is extended into XMS.

SCHEDULE:

Surprisingly, this project can be developed on roughly the same schedule as the first alternative, but it would use more people. The Q.A. effort would be significantly increased, as would the Q.A. time. Development turnover to Q.A. would occur about January 1971, and field delivery would occur in April 1971.

4. Alternative Three: Phase One of a Completely Specified XMS

This system is the release of phase one of XMS as interpolated from the extant draft planning specification. The schedule assumes that the functional specification for all of XMS is worried through and written down and that then the releases are designed, built and delivered serially. It has the apparent advantage of having a complete specification which would govern an implementation effort spanning three years. It also pushes the first appearance of multiprogramming far into the future.

COMMENTS:

This is a clean, first release of XMS whose design accounts completely for future, planned releases. The function and scope can be gleaned from the existing XMS planning document. However, there is a great deal of evidence pointing to the unpredictability of product requirements that far in the future. We might lock ourselves into a product that could not be made to meet market needs three years hence.

SCHEDULE:

The functional specification should be completed late in 1970, with Q.A. turnover occuring around January 1972, and field delivery occuring in the second quarter of 1972.

5. Alternative Four: Incremental Design and Implementation of XMS

This approach differs from the one previously outlined in that first XMS release is trimmed back in scope from that currently implied by the planning document. Further, this method is based upon a series of incremental planning, specification, design and implementation tasks. We would not attempt to specify all of XMS before undertaking the first phase. We would always specify and implement by extending a released product and would specifically exclude any complete rewrite of keystone system elements.

FEATURES:

The first release of this system would offer multiprogramming of jobs and provide services, features and operational characteristics expected by BDP users. We would establish as a necessary condition (without guaranteeing sufficiency) the ability to satisfy some identifiable total application(s) within Xerox. In other words, we would attempt to produce a system for use by Commercial Systems Integration.

SCHEDULE:

Assuming a complete planning specification is published by 1 June, Q.A. turnover would occur in April 1971, with field release in August. This further assumes that we permit UTS and XMS to diverge into two systems, building XMS upon UTM/A00 with BTM/F00 disc and file management enhancements. Alternatively, we could delay UTM/B00 by several months as well as phase one of XMS, and satisfy our UTM/B00 requirements with this first release of XMS. Because of external pressure, we will have to emphasize delivery of UTM/B00 with its reliability, real-time, and file management improvements (over UTM/A00). This forces the divergence of UTM and XMS until, perhaps, the second release of XMS in 1972.

6. Conclusion

We prefer alternative four for a number of reasons. It gets a first release of XMS quickly. It forces XMS to satisfy the known needs (or a known part of them) of a large BDP customer. It is realistic, in that it recognizes our inability to predict future needs with precision. It will admit of reasonably accurate scheduling because we will build upon what we have in hand. We will avoid any massive rewrite of the system.

This compromises both the features and the availability of our first multiprogramming system; yet it gives a BDP oriented multiprogramming system expeditiously. Choosing this strategy implicitly recognizes that our needs for multiprogramming are those of the BDP environment and not those of a scientific or real-time one. If we must produce a multiprogramming batch system for these latter uses, then we could use alternative one or, possibly the Vanderbilt dual batch system. This assumes that we don't attempt to sell this system or use it in BPD environments. Either of these interim solutions, however, commits us to even further product enhancements, maintenance, etc., and will delay both XMS and UTM. This happens because it uses resources that would otherwise be applied to these efforts. The long-term cost of a quick and dirty multiprogramming system will be very large indeed, especially when measured by the resultant UTM and XMS slippages.

We argue, therefore, for an all-out effort focused upon conversion opportunities within Xerox as the best available strategy to drive our multiprogramming development.

Dan Cota

bb

XDS memorandum

TO George Boyd

FROM Ron Thomas EXT 641-1952 MAIL STA C7-33

SUBJECT Marketing's Response to Xerox Multiprogramming System (XMS) Planning Specification

DATE Apr. 27, 1970

REF RKT-058

Cy → Keddy
Bryan
Doeppel
Haney
[illegible]
Mallonee
Martin Th
Martin C
Reid
Reilly
Sharpe

REF: Memo from George Boyd, PP-GB-1496, dated Mar. 18, 1970

The Marketing Division is, in general, enthusiastic about the XMS as described in the preliminary specification. Because of our near term needs, we would like to request that some consideration be given to adjusting the priority of implementation. We are also offering some comments in the functional aspects of the XMS specification and hope that they are considered in your final product specification.

I. Priority of implementation.

A. The following is a list of the functional capabilities Marketing feels is needed in the time frame approximately coinciding with the first phase of the XMS development. If the development resource is a constraint, Marketing feels that the fail soft may be postponed in the later phase of the development as a trade-off.

1. Basic multi-programming facilities:
   a. priority job scheduling.
   b. resource requirements specification in control language.
   c. resource pre-allocation at beginning of job step.
   d. memory management.
   e. job step predicate (for process sequencing).

2. File management:
   a. complete disk pack support.
   b. multi-volume file and multi-file volume facilities.
   c. ANSI standard labels.
   d. current file sharing capabilities.
   e. file concatenation.
   f. default peripheral assignment.
   g. user defined standard label.
   h. user error disposition.

i. retention cycle, not expiration date.

j. relaxation of limits for:

- DCB
- buffers
- file size
- file number per volume, etc.
- serial numbers, etc.

In general, file management must be designed to accommodate the commercial users with large data files.

3. Symbiont:

a. standard file format.
b. form control.
c. multiple copies of output.
d. operator control of scheduling.
e. device independence.

4. Remote Batch:

a. log on/off.
b. central accounting.
c. form control.
d. automatic dial.
e. error recovery.
f. full/half duplex.

5. Checkpoint/Restart:

a. symbiont checkpoint/restart.
b. COBOL checkpoint/restart.

6. Accounting:

a. consistency between batch, remote batch and time-sharing uses.
b. within a user account, separate usage accounting for chargeable processors such as COBOL and FMPS.
c. flexibility for installation options.

B. Phase II.

1. Multi-programming Extension:
   - a. job control language as described in the preliminary specification:
     - . conditional execution
     - . cataloged procedures
     - . forked job step generation
   - b. dynamic resource allocation.

2. File Management:
   - a. full file sharing: read, write, update.
   - b. cataloging.
   - c. sequential sub-file.
   - d. full user label support.

3. Basic Communication Management:
   - a. symbiont control of remote terminals.
   - b. file manipulation.

4. Checkpoint/Restart to include programmer initiated checkpoint.

C. Phase III.

1. Full Data Management.

2. Full Communication Management.

3. Program Fragmentation.

4. Multi-processing.

5. Fail soft.

II. Functional Considerations.

A. Dynamic Resource Allocation.

In conjunction with job class and priority scheduling, more overlap usage of system resources may be achieved if the resources are allocated at the actual time of attachment. A system with this allocation method is known to have been implemented with the two following fundamental allocation rules:

1. the remaining resource requirements of a requesting job must all be available at the time of allocation.

2. the specific resource being requested is free at the time of allocation.

This system is known to be system-jam proof under normal circumstances.

B. Command Language Enhancement.

Additional control commands should be added to the current command repertoire in the following areas:

1. resources allocation - REQUIRE and FREE.

2. file management - RENAME to allow renaming of file identification without copying the file.

3. fail soft - give user some flexibilities in file checkpoint and backup.

C. Multi-tasking.

The complexity of this problem is appreciated. However, we request you reevaluate this capability for the following reasons.

1. OS/MVT has this capability now. How competitive can we be without it thirty-six months from now?

2. ANSI COBOL specifies multi-tasking.

3. If we ever have a full PL-1, we need multi-tasking.

4. User level scheduling (using monitor as a host system) requires multi-tasking, especially in teleprocessing applications.

Ronald K. Thomas, Manager
MARKET REQUIREMENTS PLANNING

PREPARED BY:	APPROVED:
Edward S. Keh Software Specialist	L. B. Perillo Vice President - Marketing

RKT:ESK:bbw

xc: M. Gingrich, J. Hargrave, C. Hoffman, L. Miller, D. Shaw, F. Yee

XDS

Memorandum

TO Bob Spinrad

FROM Shel Klee EXT 1927 MAIL STA. A3-33

SUBJECT MULTIPROGRAMMING - COMMENTS ON COTA'S MEMO

DATE May 1, 1970

REF APD70-73

I've just reviewed Cota's memorandum on multiprogramming and felt some comments were in order. Since late 1967, (the earliest my chrono file reflects any data) the marketing forces of this company have been asking for a multiprogramming capability for Sigma 7. Two and one-half years later, the situation appears to be the same with one significant difference -- in late 1967 our competition by and large had only announced multiprogramming software, whereas now in 1970, virtually all of them have some form of this software in operation. Marketing is still asking for multiprogramming, but now their pleas are desperate because it may mean survival.

We must have an operational multiprogramming in short order and must announce one almost immediately. There are a number of procurements from government and industry (as we started seeing in late 1967) which we simply must "no bid". With multi-batch we might have a crack at some of this business. There is no question our market has changed somewhat since 1967; we are gravitating toward the heavier BDP usage. Our present market for Sigma 7's is not the scientific real-time market, but rather the general purpose market where there is a good mix of compute bound and I/O dominated jobs that are processed daily. We recognize that the requirements of some of the larger 100% BDP users in the area of multiprogramming may demand the equivalent of an OS-MVT system, but we don't believe this to be a necessity to satisfy either our present market or the major part of the BDP market. In these considerations, it is extremely important to note that the vast majority of IBM users utilizing multiprogramming techniques use MFT and not the MVT we appear to be trying to emulate in our XMS specification. In any event, it is folly to believe that a multiprogramming system alone, no matter how advanced and efficient, will deliver to us a segment of the BDP market save perhaps internal Xerox applications. We need other things as well: additional hardware, commercially oriented support people, applications software, to name just a few. Since many of these are long lead items, I suggest we address ourselves to two problems: the immediate one of providing our present market the desired capability and a second one of meeting the challenges of the "total BDP" market we choose to pursue.

Just a little aside about "quick and dirty" systems, a good example of which is RAD-75. This system, bootlegged in AP in early 1967, saved SDS a good deal of business while we struggled to get BPM working in any acceptable manner. We are now phasing this system out since RBM-2 has been released and BPM enhanced to the point where both these products are functional supersets of it. We will not be forced into supporting a short range multiprogramming system forever as long as the ultimate system we are planning can do everything the initial one could.

The question of resources not being available, we presume, is based on UTS-knowledgeable personnel. We believe it would be a serious mistake to provide our first multiprogramming system on a UTS base. An estimated field release of January or April '71 is a dream. Our experience with UTS thus far makes us wary about the status of a non-multi-batch system in the first quarter of '71. We all should recognize from our experiences with BPM/BTM what it will take to obtain a reliable, maintainable system. UTS is much too complexand six months just won't do it. BDP users, whether on their own machine or the engineering department's, demand this reliability. They get it from IBM.

We are just beginning to see the light in the BPM/BTM area. The system is reasonably solid now and most of the previous reliability problems seem well in hand. We believe building on a solid base to be essential to our ability to deliver and, therefore, recommend the company provide a fixed partition multiprogramming system utilizing a BTM base. The hardware memory map could possibly be utilized to handle the multiple batch portion of the system.

From our knowledge of the BTM system and the status of UTS, we believe this to be the most realistic, reliable way to achieve a multiprogramming capability in short order. We must have this for Sigma 6.

In summary, we do believe a multiprogramming system can be provided quickly and will be acceptable to the present and short range future (1 - 2 years) markets. We do not believe this system will be with us forever or would be dead-ended but rather a product which will be functionally superseded by our ultimate system. We agree with Cota that implementing XMS, if we do, should be in small steps with periodic reviews. This is the best approach even if it may initially seem to take longer to get to the end point in our development.

We have procrastinated long enough. Let's do something!

Sheldon Klee, Manager
Applied Programming

SK/sc

cc: G. Boyd, A. Bongarzone, E. Bryan, D. Cota, B. Doeppel, G. Eckley, F. Haney, D. Heying, D. Keddy, E. Kinney, B. Mallonee, C. Martin, T.W.Martin, M. Micheletti, B. Reid, D. Reilly, J. Romey, B. Sharpe

XDS

Memorandum

TO	Distribution	DATE	7 May 1970
FROM	Dan Cota EXT 1441 MAIL STA A1-02	REF	PR-70-1033
SUBJECT	More on Multiprogramming		

The attached points out the consequences of restoring the multibatch capability that now exists -- but which has been disabled -- in UTS. UTS in its present form provides batch multiprogramming, although not without problems. These problems can be overcome by relatively simple installation-controlled operating procedures and policies. These restrictions are similar to those which we have adopted on the Sigma 7T. They are not unreasonable.

We still predict that UTS will be available by the end of August. If we can sell around the existing difficulties by providing operating procedures to our customers, then we can offer a basic multiprogramming capability with E00 disc pack support and all the other features of UTS release by the end of August.

As pointed out in my earlier memo on multiprogramming, this is not a BDP monitor. It will not help us enter new markets or replace IBM 360's. But if multiprogramming is important to our present customers and in our traditional market, then perhaps we already have what we need.

Dan

Dan Cota

bb

Attachment

Distribution:

A. Bongarzone
G. Boyd
E. Bryan
B. Doeppel
W. Gable
W. Glavin
F. Haney
D. Keddy

S. Klee
W. Mallonee
C. Martin
D. McGurk
B. Reid
D. Reilly
B. Sharpe
R. Spinrad

XDS

Memorandum

TO Dan Cota

FROM E. Bryan, B. Doeppel,
F. Haney, B. Sharpe

EXT 1421 MAIL STA A1-02

SUBJECT UTS MULTIBATCH

DATE 7 May 1970

REF PR-70-3017

The UTS multibatch capability is currently disabled by several tests and switches in the code.

If we enable the multibatch code, the following difficulties can arise:

1. <u>File Contention</u>

Turning on the UTS multibatch switch increases the number of job aborts caused by file conflicts.

UTS has three types of files: 1) temporary files known throughout a job, 2) scratch files known within a job step, and 3) permanent files which can be shared by users. With minor variations for the different file types, the file contention problems in a single-batch UTS system are as follows:

a. If on-line user A tries to use a file that is unavailable because it is being used by on-line user B or by a batch program, on-line user A is notified of the conflict. He can do something else, or he can wait until the file is available.

b. If a batch program tries to use a file that is not available because it is being used by an on-line user, an error condition is returned to the program. The program may be able to do something else, or wait to avoid a conflict. If the program does not check the "in use" condition, the program is aborted.

If the multibatch switch is turned on, the timesharing - timesharing conflicts remain the same. The incidence of timesharing - batch conflicts may increase since there will be more than one batch program in execution. The possibility of batch - batch conflicts is introduced. These are handled like the case where a batch program tries to use a file being used by an on-line user. The first program to use the file retains it. A second program trying for access to the file gets an error condition. Unless the program tests the error condition, it is aborted.

2. Tape Contention

In single-batch UTS if a job tries to use more tapes than are available, the job is aborted. If one job uses all available tapes, then as each new job executes it will abort if it requests a tape. Turning on the multibatch switch aggravates the problem since more programs compete for tapes.

Another consideration related to enabling UTS multibatch is the fact that execution of batch programs is under control of an algorithm designed for on-line users. Scheduling is based on assumptions about response time for on-line users, think-time, etc. Programs are held in memory from the beginning of an I/O operation until its completion. This does not result in the type of device-use optimization normally associated with multiprogramming. A similar inefficiency can occur if a job occupies almost all of available memory. If the job does I/O it is held in core until I/O completion, at which time, if the quantum is completed, the entire job is swapped out if the other batch job is ready to run and core is needed. In this case, multibatch does not result in parallel use of devices. Only one I/O stream operates and normal batch efficiency is degraded by swap I/O. Of course, with sufficient core storage to hold all active jobs multiple I/O streams and concurrent CPU use will occur and no swaps will occur.

3. Operating Procedures

When the UTS multibatch code is enabled, the tape and file conflicts can be practically eliminated by the following operating procedures and restrictions:

a. Every batch job in the job queue must have a unique account number if it uses COBOL or Manage. This may also apply to other processors produced outside of PDD: FLAG, SL/I, FMPS, etc. This is being investigated by Shel Klee.

b. The number of tapes required at one time must be regulated by the tape limit associated with each on-line user and by operator supervision of tapes for batch programs. The total number of tapes that the system will require must be no greater than the sum of tapes used by on-line users and batch programs. The operator can tell by the !LIMIT card how many tapes are required by a batch job. As he puts jobs in the queue, he must guarantee that the tapes required by all jobs that can run concurrently, plus tapes used by on-line programs, do not exceed the total available. Suppose, for example, that a system has eight tapes, that two are being used by on-line programs, and that the maximum number of jobs is four. Six tapes are available for batch. Suppose the job queue contains only jobs requiring one tape. If a new job is submitted the operator must assume that three of the six tapes may be in use when the new job is executed. Therefore, the new job may use one, two, or three tapes but no more.

These restrictions will cause some revision of UTS installation operating procedures, but with some simple operator guidelines, installations should obtain improved throughput and device utilization because of the multibatch facility.

Ed
G. E. Bryan

J.E. Doeppel
J. E. Doeppel

Fred M. Haney
F. M. Haney

R. Sharpe
R. A. Sharpe

bb

xc: D. Keddy
C. Martin

SDS

Memorandum

E. Bryan
91-03

TO George Boyd

FROM Paul Hibbs EXT 2410 MAIL STA A1-77

SUBJECT XEROX MULTIPROGRAMMING SYSTEM
(XMS) PLANNING SPECIFICATION

DATE 5/7/70

REF CCD-70-8146

The XMS Planning Specification has been reviewed by the Management Information System Section and Utility Development Programming Group. In general, all concerned were of the opinion that this was a positive step toward our goal of competing in the business data processing field. The individuals who participated in this review are closely associated with the current 360 to Sigma 7 BDP conversion effort and have worked extensively with IBM's OS or DOS. The following comments arose from the review.

1. The last sentence of the second paragraph on page nine specifies that "Jobs with the same account number will not be scheduled concurrently." It is felt that this is an unreasonable restriction and the system should be able to identify jobs by some other method. For example, to make our own business production scheduling a simpler task, all of our business programs are executed under the same account.

2. The forked job concept will provide a powerful capability to the user. However, i. is felt that this capability should be expanded. Ideally, it would be desirable to be able to create a job fork that would sort a file that would be used by the main job at a later step. This means that now the execution of a job step would be conditional on the completion of a forked job step, and that forked jobs could pass files to a job step.

3. In the area of resource allocation, it is feasible that a high priority job could tie up enough resources waiting for an additional device that the machine would be processing only the job utilizing that device. During this time period it is possible that there are resources available to run jobs that would complete before the high priority job would have all necessary resources available to run. It is felt that the scheduler should be able to determine from a job execution time limit whether or not to utilize some devices out of the high priority job device pool for execution of shorter jobs.

   For Example:

   > Job 'A' needs three tape drives for execution but cannot execute because there are only two available.

Job 'B' is using the third drive and will not complete execution for ten minutes.

Job 'C' is a five minute job waiting to execute which requires one tape drive.

Rather than wait for jobs 'A' and 'B' to complete, start job 'C' since it will be done before job 'B' can release the third tape drive necessary for the execution of job 'A'.

4. The portion of the specification dealing with the symbionts generated a number of comments and suggestions.

A. From this department's investigation, it appears that key-board to tape equipment (i.e., CMC, Mohawk, etc.) is becoming quite popular in installations which have a large volume of keypunching. It is therefore recommended that the symbionts be able to process unit record input tapes. This should be implemented such that the data is read off the tape and stored as a part of the symbiont stream as opposed to waiting for the job to request the data.

B. Considering the volume of output generated by most business applications, the output should be processed at end of job step and not retained until total job completion.

C. The user should be provided with the ability to communicate form requirements to the symbiont system. The symbiont system should then react by "batching" all jobs requiring the same form, thereby eliminating as much forms change as possible.

D. The operator should have the ability to kill specific symbiont output by DCB assignment. This would enable the user to kill the printing of program output but still receive diagnostics and post mortem dumps.

E. The user should be provided with the ability to assign symbiont DCB's to separate (even non-existent) printers so that a separate symbiont file will be created for each DCB. This would enable concurrent generation of separate reports by the same job step or fork. At present the number of concurrent reports that can be generated is a function of the number of line printers available.

F. The fifth paragraph on page 25 seems to suggest that the symbiont files should be created and accessed via the file management portion of the monitor. We feel that this may slow the symbiont processing time. There is also an advantage to this method, in that it would no longer be necessary to allocate symbiont disc space at SYSGEN time, but rather obtain this space dynamically. If it is not intended that the symbionts work through the monitor, it is felt that the monitor and the symbiont system should work out of a common granule pool.

G. The solution presented for the problem of "what to do when the symbiont storage area is filled to capacity" looks a bit messy. We weren't able to come up with an alternative solution, but felt that the idea of spilling symbiont data off to tape presented operator intervention problems that could prove disasterous.

If you desire to discuss any of the points outlined in this memo, please feel free to call.

Paul

P. E. Hibbs

PEH:tt

xc: A. Bongarzone
E. Bryon
D. Cota
V. DeVine
G. Dobbs
B. Doeppel
P. England
R. Evans
B. Gable
R. Gold
F. Haney
R. Keddy
J. Mendelson
L. Miller
G. Myers
L. Perillo
V. Porizky
B. Seith
R. Sharpe
D. Shaw
R. Spangler
S. Spiegel
R. Spinrad
W. Todd
B. Wilson