This is a preliminary, unofficial description of the ECS "layer" of the CAL time-sharing system on a 6400 with central exchange jump and Extended core storage. This document precedes any attempt to implement the ECS "layer" of the system but it should describe the "user machine" defined by the ECS system. The following describes the major elements of the ECS system. Most of these elements are also ECS objects while the rest (subprocess and subprocess map) are components of ECS objects whose complexity warrents a separate description.

Special comment should be made about the nature of "operations" with respect to the overall system design. The system is to be constructed in layers. The first layer is the ECS system. The next layer will implement the disc as a backing store for ECS. Other layers are contemplated to simulate Scope and provide comprehensive debugging facilities. An operation is an ECS object whose function is to manage the transfer of control and passing of parameters between system layers operating in different environments of protection. Furthermore, operations are used to "pass the buck" when a layer finds it is unable to complete its assigned task. For instance, assume an operation has been invoked to ask the ECS system to access part of a file. If the ECS system should discover that the portion of the file requested is not in ECS, then F return (see Process) will cause the operation to be invoked again, transfer control to another layer of the system (in this case, the disc system which would schedule the needed transfer). Thus the intervening layer of the disc system adds overhead in the file access only when the access fails at the ECS level.

This example also illustrates another characteristic of operation, their ability to hide lower levels of the system. They make all levels below the level initiating the operation appear to be one grand system which will perform all actions for which there are operations at the higher level. This is done through the "buck passing" and by using operation to call a subprocess at a different level which in turn may compute or invoke additional operations.

One hsitates to carry the "layer" model too far. The process construct will be used to implement all layers except the system. The hierarchy of subprocesses provides varying protection within the process and should be studied to understand the major protection mechanism of the system.

The following, besides describing the major elements of the ECS system, also describes the actions which the ECS system will undertake on these objects. This description is not intended to be an exhaustive compilation of the ECS actions but rather, a list of most of the interesting ECS actions.

These documents were produced for internal use and thus may not provide an easily readable description of the ECS system. Questions should be directed to Howard Sturgis or Bruce Lindsay, Berkeley Computer Center (phone 642-1491).

Event Channel

1.1 The event channel is used to cause "wake ups" and to "block"

1.2 Structure of event channel

An event channel consists of 2 queues, only one of which may be nonempty at a given time. (not true - see below)

1.2.1 A queue of events (fixed size list)

An event consists of 1) D: a process name — what is a process name?

2) D: a data word

1.2.2 A queue of waiting processes (chained list)

This queue is a chain through the chaining word of the processes.

1.3 Actions on an event channel

1.3.1 Place an event on an event channel

1.3.1.1 Parameters of this action are

1.3.1.1.1 C: an event channel

1.3.1.1.2 D: a data word

1.3.1.2 The action may be divided into 3 steps

1.3.1.2.1 Form an event consisting of

1.3.1.2.1.1 The name of the process calling for the action

1.3.1.2.1.2 AP2

1.3.1.2.2 Dispose of the event

1.3.1.2.2.1 If the event queue is empty;

1.3.1.2.2.1.1 If the process queue contains no process which does not have a "wake-up-waiting"; then, place the event on the event queue.

1.3.1.2.2.1.2 If there is a process on the process queue without a "wake-up-waiting"; then, set "wake-up-waiting" for the first such process; pass the event to the process (using a local hidden variable "waiting event"); and if the process is not "running" then do 1.3.4.3.5. ← you mean 1.3.4.3.4?

this contradicts the assertion that only one queue is non-empty at a time. What is a process doing in the process queue if it has "wakeup waiting" and is therefore, by definition, ready — or doesn't this agree with Kampson's use of the term "wakeup waiting"

1.3.1.2.2.2 If the process queue of the event channel is empty, there is no entry in the event queue duplicating the new event, and there is more than 1 free place in the event queue;

then the new event is placed on the event queue.

1.3.1.2.2.3 If the process queue of the event channel is empty, there is no entry in the event queue duplicating the new event, and there is exactly 1 free space in the event queue;

then a special "you lose" event is placed on the event queue. (meaning?)

1.3.1.2.2.4 If the process queue of the event channel is empty, there is no entry in the event queue duplicating the new event, and the event queue is full;

then nothing is done on the event channel.

1.3.1.2.2.5 If the process queue is empty, and there is an event in the event queue which duplicates the new event;

then nothing is done on the event channel.

1.3.1.2.3 A number is returned to the calling subprocess in X6 to indicate which form of step 2 was taken.

1.3.2 Get an event from an event channel or "block"

1.3.2.1 The parameter of this action is:

1.3.2.1.1 C: an event channel

1.3.2.2 The action depends on the state of the event channel

1.3.2.2.1 If the event queue is nonempty;

then the event on top of the queue is removed and placed in X6 and X7 of the calling subprocess.

1.3.2.2.2 If the event queue is empty;

then the P-counter (of the calling subprocess) is decremented by 1, the process is placed at the tail of the process queue of the event channel (using the first of its chaining words) and the process is "blocked" (not running) until an event causes a "wake-up" ("wake-up-waiting"). The contents of the event (see 1.3.1.2.1) are copied to X6 and X7 of the calling subprocess, the P-counter (of the calling subprocess) is incremented by 1, the process is removed from the process queue of the event channel, and scheduled to run.

1.3.3 Get an event from the event channel or make "F return"

1.3.3.1 The parameter of this action is:

1.3.3.1.1 C: an event channel

1.3.3.2 The action depends on the state of the event channel

1.3.3.2.1 If the event queue is nonempty;

then the action is the same as in 1.3.2.2.1

1.3.3.2.2 If the event queue is empty;

then an F return is made.

1.3.4 Get an event from any one of a number of event channels or "block".

1.3.4.1 Parameters

1.3.4.1.1 D: number of event channels

1.3.4.1.2 D: relative location of a sequence of capability indices for event channels

1.3.4.2 Error

1.3.4.2.1 AP1 is negative or greater than the number of chaining words in the process.

1.3.4.2.2 Any one of the capability indices starting at AP2 does not specify an event channel with the necessary options.

1.3.4.3 Action

1.3.4.3.1 All capabilities specified by AP1 and AP2 are checked to insure that they do not satisfy 1.3.4.2.2. Also decrement the P-counter by 1.

1.3.4.3.2 For each of the event channels the following action is taken (as an atomic act)

1.3.4.3.2.1 If "wake-up-waiting" for the process, then go to 1.3.4.3.4.

1.3.4.3.2.2 If the event queue is non-empty, then transfer the head of the event queue to "waiting-event", set "wake-up-wating", and then go to 1.3.4.3.4.

1.3.4.3.2.3 Otherwise, place the process on the process queue using the next available chaining word.

1.3.4.3.3 If step 1.3.4.3.2. was completed for all the specified event channels, and there is no "wake-up-waiting" then block until there is "wake-up-waiting" (may be zero time). Then go to 1.3.4.3.4.

1.3.4.3.4 Remove the process from all event channels on which it is queued (i.e., unchain all chaining words in use). Copy the event in "waiting-event" to X6 and X7 of the process, increment the P-counter by 1, and schedule the process to run. NOTE: The process is given no explicit information as to which event channel (if hung on more than one) produced the event.

1.3.5 Create an event channel

1.3.5.1 Parameters

1.3.5.1.1 C: accounting block

1.3.5.1.2 C: location in full C-list to return capability for the event channel.

1.3.5.1.3 D: length of the event queue for the event channel

1.3.5.2 Errors

1.3.5.2.1 Errors may be caused due to insufficient resources in AP1 (see alloc block, 11.2.1).

1.3.5.2.2 If the location specified by AP2 does not exist or does not contain a "null" capability, an error will be generated.

1.3.4.2.3 AP3 is negative.

1.3.5.3 Action: a new event channel with an event queue of length AP3 is created by the authority of AP1 and attached to AP1. The event queue is initialized to "empty" and a complete (all options allowed) capability for the new event channel is placed in the full C-list of the calling subprocess at the location specified by AP2.

Capability List

2.1 The function of a capability list (C-list is to store capabilities to be used by subprocesses as needed.

2.2 A C-list is a fixed length sequence of capabilities.

2.2.1 Capabilities are used to name objects within the ECS system.

2.2.2 A capability is a pair

1. An object
2. A set of option bits

2.3 The "full C-list" of ~~an active subprocess~~ a process is defined dynamically by the subprocess tree and the state of the stack. The full C-list is a concatenation of C-lists of several subprocesses.

2.3.1 The full C-list is formed by concatenating the C-lists of all the subprocesses on the path between the current "top of path" and the current subprocess (see "process"). The C-list of the current subprocess is the first while the C-list of the "top" of the tree is the last in the concatenation.

2.3.1.1 Let N ← "top of path". Initialize sequence S ← empty.

2.3.1.2 S ← N,S

2.3.1.3 If N = head of subprocess call stack, S is complete; otherwise N ← father of N and go to step 2.3.1.2

2.3.2 Accessing the full C-list

2.3.2.1 Let the lengths of the C-lists which form the full C-list be $\ell_0,\ell_1,\ell_2,\ldots,\ell_n$. Let the index of the desired capability be I. Initialize working index: j ← 0.

2.3.2.2 If $I \leq \ell_j$ then the desired capability is the I-th capability in the j-th C-list of the full C-list.

2.3.2.3 $I \leftarrow I-\ell_j$; j ← j+1; if j < n then generate an error.

2.3.2.4 Go to step 2.3.2.2.

2.3.2.5 NOTE: The first C-list of the full C-list (i.e., the C-list of the current subprocess) may not reflect changes made to it by another process as soon as they are made.

2.4 Actions on a C-list

2.4.1 Display capability from full C-list

2.4.1.1 The parameter of the action is

2.4.1.1.1 D: an index to the full C-list

2.4.1.2 A representation of the capability specified by AP1 (see 2.3.2) is copied to X6 and X7 of the calling subprocess.

2.4.2 Move capability and mask options within full C-list

2.4.2.1 The parameters of the action are

2.4.2.1.1 C: any capability to be moved

2.4.2.1.2 D: an index to the full C-list

2.4.2.1.3 D: a mask specifying options to be preserved

2.4.2.2 First the options of AP1 are "ANDED" with the mask provided by AP3. Then the resulting capability is entered in the full C-list at the location specified by AP2.

2.4.3 Copy a capability from a C-list to the full C-list of the calling subprocess

2.4.3.1 Parameters

2.4.3.1.1 C: a C-list

2.4.3.1.2 D: the index in AP1 of the capability to be copied

2.4.3.1.3 D: a location in the full C-list for the capability to be placed (see 2.3.2)

2.4.3.2 If AP2 is negative or greater than the length of AP1, an error is generated. The capability in the C-list specified by AP1 and AP2 is copied to the full C-list at the relative location specified by AP3.

2.4.4 Copy a capability from the full C-list to a C-list

2.4.4.1 Parameters

2.4.4.1.1 C: a C-list

2.4.4.1.2 D: the index of the target in the C-list of AP1

2.4.4.1.3 C: the capability to be moved (see 2.3.2)

2.4.4.2 If AP2 is negative or greater than the length of AP1, an error is generated. The capability in the full C-list of the calling subprocess which is specified by AP3 is copied to the location specified by AP1 and AP2.

2.4.5 Create a C-list

2.4.5.1 Parameters

2.4.5.1.1 C: allocation block

2.4.5.1.2 D: length of new C-list

2.4.5.1.3 D: index in full C-listto return the capability

2.4.5.2 Error conditions

2.4.5.2.1 Insufficient resources in AP1 (see alloc block 11.2.1)

2.4.5.2.2 The location specified by AP3 does not exist or does not contain a "null" capability.

2.4.5.2.3 AP2 is negative.

2.4.5.3 A new C-list of length AP2 is created by the authority of AP1. The new C-list is initialized to "null" capabilities and a complete (all options allowed) capability for the new C-list is placed in the full C-list of the calling subprocess at the location specified by AP3.

Operations

3.1 An operation is the vehicle by which a procedure may be executed with modified limits of protection. When invoked, an operation directs the passing of parameters and specifies the action which is to be taken (the procedure to be executed).

3.2 An operation is either

3.2.1 A simple operation (operation of order 1)

3.2.1.1 An action, which consists of one of the following:

3.2.1.1.1 (1) Call a named subprocess (i.e., subprocess specified by a class code and class code index)

3.2.1.1.2 (2) Jump to a named subprocess.

3.2.1.1.3 (3) Invoke a specified system procedure.

3.2.1.2 A list of parameter specifications (PS). Each of these consists of 2 things;

3.2.1.2.1 Case specification, one of

3.2.1.2.1.1 (1) User supplied

3.2.1.2.1.2 (2) Fixed

includes an actual parameter satisfying the kind-specification for this parameter.

3.2.1.2.2 Kind-specification, one of

3.2.1.2.2.1 (1) "any" (2 or 3)

3.2.1.2.2.2 (2) Capability (C:)

This kind-specification includes

1) A type, which may be "any"

2) A set of required option bits, which may be empty

3.2.1.2.2.3 (3) Data word (D:)

This is simply a 60-bit word.

3.2.2 An operation of order n+1 consists of an operation of order n, plus

1) an action, and

2) zero or more additional parameter specifications.

3.3 An operation directs the formation of an actual parameter (AP) list from a list of parameter specifications (PS) and an input parameter (IP) list $(IP_0, IP_1, IP_2, .., I_n)$

3.3.1 Set i ← 1, j ← 1

3.3.2 If i is greater than n, the number of parameter specifications under consideration, then, all done.

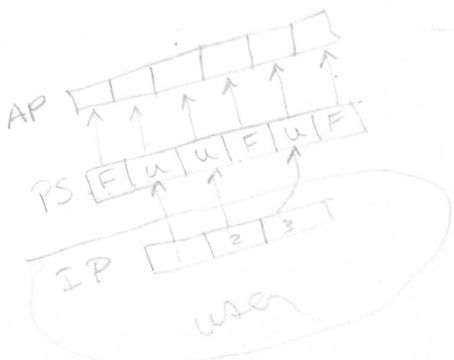
3.3.3 Examine PS_i

1. If the case of the PS_i is user-supplied and the kind of the PS_i is capability

   then fetch the capability indexed by IP_j in the full C-list (see SPEC 2). If the type of the capability does not agree with the type in the PS_i, an error is generated. If the capability does not have all the option bits on required by the PS_i, an error is generated. If no error, then the fetched capability is taken as the actual parameter.

2. If the case of the PS_i is user-supplied and the kind of PS_i is data word

   then IP_j is taken as the actual parameter.



3. If the case of the PS_i is fixed,

then the actual parameter in the PS_i is taken as the actual parameter.

4. If the kind of the PS_i is "any",

then an error is generated.

3.3.4 If the case of PS_i is not fixed, set $j \leftarrow j+1$.

Set $i \leftarrow i+1$.

Go to 3.3.2.

3.4 An operation may be invoked to handle parameter passing and transfer of control in two ways

3.4.1 Due to invocation by a subprocess. This proceeds as follows:

3.4.1.1 The subprocess executes "CEJ K" (note: CEJ must be in the high order part of the word; see subprocess return in SPEC 10)

3.4.1.1.1 If $K \geq 0$ then K is taken as the address of an input parameter (IP) list which is a sequence of data words $(IP_0, IP_1, IP_2, \ldots, IP_m)$

3.4.1.1.2 If $-7 \leq K < 0$ then -k specifies a B register which contains the address of an IP list.

3.4.1.1.3 IF $-15 \leq K \leq -8$ then -(k+8) specifies an X register whose lower 18 bits contain the address of an IP list.

3.4.1.2 The ECS system then does the following:

3.4.1.2.1 Fetch the capability indexed by IP_0 (see SPEC 2)
if the capability is not for an operation, an error is generated.

3.4.1.2.2 The unique name for the operation is placed in the stack entry of the subprocess invoking the operation (top of call stack) and the "operation counter" is initialized to 1.

3.4.1.2.3 An actual parameter list is formed using the input parameter list specified in 3.4.1.1 (starting at IP_1) and the list of parameter specifications of the simple operation contained in the operation.

3.4.1.2.4 The action specified by the simple operation is now performed with the list of actual parameters just constructed.

3.4.2 An operation may also be invoked when a subprocess makes a "failure return" (see Process).

3.4.2.1 If the "operation counter" ("op count") is less than the order of the operation name in the call stack entry, then the operation is invoked.

3.4.2.2 The "op count" is incremented; N ← "op count"

3.4.2.3 A parameter specification list is formed by concatenating the parameter specifications of the first N orders of the operation named in the call stack. An actual parameter list is formed (see 3.3) with the IP list of the original call (starting with IP_1).

3.4.2.4 The action specificed by the N-th order of the operation is now performed with the list of actual parameters just constructed.

3.5 Actions on operations

3.5.1 Change PS case from user-supplied to fixed when the kind of old PS is capability

3.5.1.1 Parameters

3.5.1.1.1 C: an operation

3.5.1.1.2 D: an index, j

3.5.1.1.3 C: a capability

3.5.1.2 If the case of PSj is not user-supplied or the kind of PSj is not capability, then an error is generated. If the type and option bits of AP3 do not satisfy the specifications of PSj an error is generated. Otherwise the case of PSj is changed to fixed and AP3 is taken to be the AP specified by PSj.

3.5.2. Change PS case from user-supplied to fixed when the kind of the old PS is a data-word.

3.5.2.1 Parameters

3.5.2.1.1 C: an operation

3.5.2.1.3 D: an index j

3.5.2.1.3 D: a data word

3.5.2.2 If the case of PSj is not user-supplied or the kind of PSj is not data word, then an error is generated. Otherwise the case of PSj is changed to fixed and AP3 is taken to be the AP specified by PSj.

3.5.3 Increase option mask requirements

3.5.3.1 Parameters

3.5.3.1.1 C: an operation

3.5.3.1.2 D: an index, j

3.5.3.1.3 D: an option mask

3.5.3.2 If the kind of PSj is not capability an error is generated. If the case of PSj is not user-supplied, an error is generated. Otherwise the option mask specified for PSj is replaced by its original value "ored" with AP3.

3.5.4 Add a type

3.5.4.1 Parameters

3.5.4.1.1 C: an operation

3.5.4.1.2 D: an index j

3.5.4.1.3 D: a type number

3.5.4.2 If the kind of PSj is not capability, an error is generated. If the case of PSj is not user-supplied, an error is generated. If the type specified by PSj is not "any", an error is generated. Otherwise the type of PSj is set to that indicated by AP3.

3.5.5 Set kind to data word

3.5.5.1 Parameters

3.5.5.1.1 C: an operation

3.5.5.1.2 D: an index j

3.5.5.2 If the kind of APj is not "any", an error is generated. Otherwise the kind of APj is set to data word.

3.5.6 Set kind to capability

3.5.6.1 Parameters

3.5.6.1.1 C: an operation

3.5.6.1.2 D: an index j

3.5.6.1.3 D: a type number

3.5.6.1.4 D: an option mask

3.5.6.2 If the kind of APj is not "any", an error is generated. Otherwise the kind is set to capability and the type and option bits required are taken from AP3 and AP4.

3.5.7 Display an operation

3.5.7.1 Parameters

3.5.7.1.1 C: an operation

3.5.7.1.2 D: location within the address space of the calling subprocess

3.5.7.1.3 D: maximum number of words to be displayed

3.5.7.2 Error conditions

3.5.7.2.1 AP2 or (AP2 + AP3) does not lie within the address space of the calling subprocess.

3.5.7.3 The number of orders in the operation is placed in the location specified by AP2 followed by the contents (action and parameter specifications) of each order beginning with the first order. As many complete orders as possible will be copied to subproces memory. If all orders are not included, the remainder of the buffer is zeroed. Each order will contain its length, expressed as the number of parameter specifications to help the user make sense of the displayed operation.

3.5.8 Create a new operation

3.5.8.1 Parameters

3.5.8.1.1 C: an allocation block

3.5.8.1.2 D: an index in the full C-list to return the capability for the new operation.

3.5.8.1.3 C: an operation

3.5.8.1.4 D: type of action to be added

3.5.8.1.5 C: class code

3.5.8.1.6 D: class code index

3.5.8.1.7 D: number of new parameter specifications

3.5.8.2 Error conditions

3.5.8.2.1 Insufficient resources in AP1.

3.5.8.2.2 The full C-list entry specified by AP2 does not exist or does not contain a "null" capability.

3.5.8.2.3 AP4 does not specify a jump or a call (0 = call; 1 = jump).

3.5.8.2.4 AP7 is negative.

3.5.8.3 A new operation is created by the authority of AP1 and the new complete (all options allowed) capability is copied to the C-list entry specified by AP2. The new operation is created from the operation AP3, the action specified by AP4 (either a subprocess call, or a subprocess jump) and the subprocess of AP5 and AP6, and the number of parameter specifications of AP7. The new parameter specifications will be initialized as user-supplied of kind "any".

3.5.9 Make a copy of an operation

3.5.9.1 Parameters

3.5.9.1.1 C: an allocation block

3.5.9.1.2 D: an index in the full C-list to return the capability for the new operation

3.5.9.1.3 C: am operation to be duplicated

3.5.9.2 Error conditions

3.5.9.2.1 Insufficient resources in AP1.

3.5.9.2.2 The full C-list entry specified by AP2 does not exist or does not already contain a "null" capability

3.5.9.3 The operation of AP3 is duplicated and a complete (all options allowed) capability for the new operation is placed at the location specified by AP2 in the full C-list.

Subprocess

6.1 A subprocess is the active element of a process. Only a subprocess may perform actions or execute machine instructions.

6.2 Structure

6.2.1 Subprocess operating environment

6.2.1.1 Capability for a C-list

6.2.1.2 A map

6.2.1.3 backpointer to "father" in the subprocess tree

6.2.2 Subprocess identification

6.2.2.1 class code

6.2.2.2 class code index

6.2.3 Entry point address of first instruction to be executed when the subprocess is called.

6.2.4 Error selection mask (ESM)

6.2.5 Interrupt datum and "interrupt waiting" flag

6.3 Actions on a subprocess

6.3.1 Display a subprocess

6.3.1.1 Parameters

6.3.1.1.1 C: a process

6.3.1.1.2 D: a class code (may be obtained by displaying a capability for a class code or as a backpointer in an already displayed subprocess).

6.3.1.1.3 D: a class code index

6.3.1.1.4 D: a location to return the data on the subprocess

6.3.1.2 Error conditions

6.3.1.2.1 The subprocess specified by AP2 and AP3 doesn't exist in AP1.

6.3.1.2.2 AP4 doesn't allow room within the address space of the calling subprocess to put all the subprocess data

6.3.1.2.3 A copy of the C-list capability (not a capability), the class code and index of the father of AP2, AP3 in AP1 (i.e., the backpointer), entry point, the current ESM, and the interrupt flag and datum are copied in a suitable format, beginning at the locations specified by AP4.

6.3.2 Downgrade ESM

6.3.2.1 Parameters

6.3.2.1.1 C: class code

6.3.2.1.2 D: class code index

6.3.2.1.3 D: location of suggested new ESM

6.3.2.2 Error conditions

6.3.2.2.1 The subprocess specified by AP1 and AP2 does not exist in the local process.

6.3.2.2.2 The location specified by AP3 does not lie within the address space of the calling subprocess or allow room to specify the entire ESM.

6.3.2.2.3 The suggested ESM is "ANDED" with the current ESM. The format of the suggested ESM is 32 error classes per word beginning at the high order part of the word (i.e., error class 0 is in bit 59 of the first word of the suggested ESM).

Subprocess Map

8.1 The subprocess map (map) is used to direct the swapping of a subprocess between CM and ECS. The full map, constructed from a number of maps, provides relocation of CM addresses and directs the swapping of a process. The order of swapping of maps in the full map is significant in that the contents of file which appear in more than one map may depend on the order in which the maps are swapped out. (i.e. if read only flag not set on re-entrant routines) However, no promises are made as to the order of swapping.

8.2 Structure

8.2.1 Each map is a fixed length list of entries each of which is one of the following:

8.2.1.1 An "empty" entry

8.2.1.2 A swapping directive

8.2.1.2.1 specification of a file

8.2.1.2.2 an address in the file

8.2.1.2.3 a word count

8.2.1.2.4 a CM address relative to the address space of the map

8.2.1.2.5 a "read-only" flag

8.2.2 Address space size of the map (FL)

8.3 The full map (of a process) is defined by the subprocess tree and the current state of the call stack.

8.3.1 The full map is a sequence of maps consisting of all the maps on the path, in the subprocess tree, between the current "top of path" (defined by the call stack) and the current subprocess (see "process").

8.3.1.1 Let N ← "top of path". Initialize sequence S ← empty.

8.3.1.2 S ← N,S

current subprocess

8.3.1.3 If N = top of subprocess call stack (i.e., current subprocess) S is complete. (must eventually find top of stack - see 9.2.8.2.2)

8.3.1.4 N ← father of N; go to step 8.3.1.2.

8.3.2 Computing the relocation constant of the CM addresses for each map in the full map.

8.3.2.1 Set F ← 0; set N ← index of the subprocess for whose map the relocation constant is being computed.

8.3.2.2 F ← F + (FL of the map belonging to the father of N).

8.3.2.3 If the father of N is the root of the full path, (i.e current top-of-stack) F is the relocation constant for the desired map.

8.3.2.4 Set N ← father of N.

8.3.2.5 Go to step 8.3.2.2.

actions on fullmap:

8.4 ① Display any map entry (action)

8.4.1 Parameters

8.4.1.1 C: a class code

8.4.1.2 D: a class code index

8.4.1.3 D: index of entry in map

8.4.1.4 D: location to return data

8.4.2 Error conditions

8.4.2.1 AP2 is negative or AP1 and AP2 specify a non-existant subprocess in the process.

8.4.2.2 AP3 is negative of greater than the number of entries in the map of AP1, AP2.

8.4.2.3 AP4 does not lie within the address space of the calling subprocess or does not allow space for the data.

8.4.3 The map entry specified by AP1, AP2, and AP3 is copied to the location specified by AP4.

8.5 (2) Replace map entry by "empty"

8.5.1 Parameters

8.5.1.1 C: class code

8.5.1.2 D: class code index

8.5.1.3 D: index in the map of AP1, AP2

8.5.2 Error conditions

8.5.2.1 AP2 is negative or AP1 and AP2 specify a non-existant subprocess in the current process.

8.5.2.2 AP3 is negative or greater than the length of the map of AP1, Ap2.

8.5.3 If the map specified by AP1 and AP2 is in the current full map, and the entry (AP3) is not "read only", the entry is swapped out. Then, the specified entry is changed to "empty". Otherwise, the specified entry is simply changed to "empty".

8.6 (3) Replace map entry with a swapping directive

8.6.1 Parameters

8.6.1.1 C: class code

8.6.1.2 D: class code index

8.6.1.3 D: index in the map of AP1, AP2

8.6.1.4 C: a file

8.6.1.5 D: address in the file

8.6.1.6 D: word count

8.6.1.7 D: CM address relative to the address space of the map

8.6.1.8 D: "read only" flag

8.6.2 Error conditions

8.6.2.1 AP2 is negative or AP1 and AP2 specify a non-existant subprocess.

8.6.2.2 AP3 is negative or greater than the length of the map of AP1, AP2.

8.6.2.3 AP4 is a "read only" file and AP8 is not "set"

8.6.2.4 AP6 is negative

8.6.2.5 All addresses specified by AP5 and AP6 are not in the file of AP4.

8.6.2.6 The entry specified by AP1, AP2, and AP3 is not already "empty".

8.6.2.7 The count + CM address (AP6 + AP7) is greater than FL (address space size) of the specified map (AP1, AP2, and AP3).

8.6.3 The map swapping directive specified by AP4, AP5, AP6, AP7, and AP8 is entered at the location specified by AP3 in the map belonging to AP1, AP2. If the map (AP1, AP2) is in the full map of the calling subprocess, the modified map is swapped in. If swapping directives, they will be executed in the order in which they appear in the map.

8.7 (4) Display full map entry

8.7.1 Parameters

8.7.1.1 D: Index of entry in the full map

8.7.1.2 D: Location to return data

8.7.2 Action and error conditions are equivalent to 8.4 except that the map entry is specified relative to the current full map.

8.8 (5) Replace any entry, in full map, by empty

8.8.1 Parameter

8.8.1.1 Index of entry in the full map

8.8.2 Action and error conditions are equivalent to 8.5 except that the map entry is specified relative to the current full map.

8.9 (6) Replace map entry, in the full map, with a swapping directive

8.9.1 Parameters are the same as 8.6 except that AP1, AP2, and AP3 are replaced by an index relative to the full map.

8.9.2 The action and error conditions are equivalent to 8.6 except that the map entry is specified relative to the current full map.

Process

9.1 A process is the object which is scheduled, by the system, to ~~execute machine instructions.~~ run subprocesses.

9.2 A process consists of

9.2.1 a set of subprocesses (w/class codes)

9.2.2 an exchange jump package

9.2.3 a fixed number of waiting queue chaining words (= max no of event channels on which the process will wait at any one time)

9.2.4 waiting interrupt counter

9.2.5 a call stack; each entry consisting of

call stack entry

1) a subprocess name
2) a P-counter
3) subprocess tree "top of path"
4) operation name
5) operation count (op count)
6) F return flag

9.2.6 RA+1 interrupt subprocess class code

9.2.7 RA+1 interrupt subprocess class code index

9.2.8 Process hierarchies

9.2.8.1 Subprocess tree: The backpointers of ~~each~~ the subprocesses in a process define a tree structure. The "family" of a subprocess A is the set of subprocesses belonging to the subtree of which A is the root (NOTE: The root is part of the tree.) i.e. $A \in Fam(A)$

The "path" of a subprocess is the sequence of subprocesses from the subprocess to the root of the subprocess tree. The "path" is defined by the subprocess backpointers and includes the subprocess which begins the path. i.e. $A \in Path(A)$

9.2.8.2 Call stack: The call stack records changes in control among subprocesses.

9.2.8.2.1 No entry may be made in the call stack if the number of places in the stack before the addition of the proposed subprocess is less than the number of subprocesses in the path of the proposed entry. This is to prevent the call stack from filling up to a point such that high level routines may not be called to sort out the trouble.

9.2.8.2.2 The "top of path", used in computing the full map and full C-list, is computed recursively in the call stack. To compute the new top of path: If the subprocess of the stack entry is in the path of the old top of path, then the new top of path ← old top of path, otherwise the new top of path ← subprocess of the stack entry. The top of path is carried in the call stack to allow insertion of entries in the middle of the call stack.

def. of "top of path"

? when do you want to do this?

9.2.9 Swapping

From time to time a ready process goes through the following sequence of actions:

1. a CPU and CM are selected.
2. The entries in the process full map are scanned in sequence and the indicated words are copied from the indicated files to the indicated region in CM.
3. The exchange jump package of the process is loaded into the central registers of the CPU.

4. The CPU is allowed to compute for a while or until the process goes out of "ready".

5. The central registers of the CPU are copied to the exchange jump package of the process.

6. The entries in the process full map are scanned in sequence and, for each entry without the "read only" flag set, the indicated words are copied from CM to the indicated words in the indicated files.

NOTE: If during step 4 the full map is changed, the entries of any map removed from the full map are scanned immediately or later, as in 6; and the entries of any map added to the full map are scanned as in 2.

9.3 Actions on a process within a process

9.3.1 Error subprocess call (i.e., errors caused or generated by the current subprocess)

9.3.1.1 Relevant process elements (see subprocess) and data

9.3.1.1.1 call stack

9.3.1.1.2 error selection mask (ESM)

9.3.1.1.3 error type data (parameters)

9.3.1.1.3.1 D: error class

9.3.1.1.3.2 D: error number

9.3.1.2 Error algorithm

9.3.1.2.1 Let N ← subprocess generating the error

9.3.1.2.2 If the ESM of N contains the error class (AP1), then go to step 9.3.1.2.4.

9.3.1.2.3 If N = root of the subprocess tree, then generate an error. Otherwise N ← father of N, and go to 9.3.1.2.2.

9.3.1.2.4 Activate subprocess N as if it had been called by another subprocess (see 9.3.3) which caused or generated the error except that the only parameters passed are the error class and error number.

9.3.1.2.5 Lower core (starting at RA+2) in the address space of the subprocess is set to reflect an error entry and the origins in the full map, full C-list, and address space of the previous subprocess (see 9.3.3.3.5).

9.3.2 Process interrupt (an "interrupt" is initiated by one process to "trap" another).

9.3.2.1 Parameters to initiate an interrupt

9.3.2.1.1 C: process to which the interrupt is directed

9.3.2.1.2 C: class code

9.3.2.1.3 D: class code index

9.3.2.1.4 D: a datum (18 bits)

9.3.2.2 Error conditions

9.3.2.2.1 The subprocess specified by AP2 and AP3 does not exist in AP1.

9.3.2.3 Action of an interrupt

9.3.2.3.1 If the specified subprocess in AP1 is already awaiting an interrupt call, the interrupt fails.

9.3.2.3.2 If the process (AP1) is currently executing on another processor or is in the midst of hanging itself on a series of event channels, the interrupt fails.

9.3.2.3.3 If the specified subprocess is in the "path" of the currently active subprocess (in AP1) and is not the same as the currently active subprocess, then a call is made to the specified subprocess with AP4 as the

only actual parameter. An information heading (see 9.3.3.3.5) is constructed as usual with the type of entry set to "interrupt". If AP1 is "blocked" at this time, "wake-up waiting" is set for the process (AP1) and the subprocess is unchained from all event channels before making the subprocess call. (NOTE: the P-counter of the "blocked" subprocess is not incremented so that when the return is made from the interrupt subprocess, it will re-hang on the event channel(s).)

9.3.2.3.4 If the specified subprocess (AP2, AP3) is not in the path of the active subprocess of AP1, then it is marked as "awaiting interrupt" and AP4 is stored with the specified subprocess. The "interrupt waiting" counter of AP1 is also incremented.

9.3.2.3.5 A datum will be returned, in X7, to the subprocess calling for the interrupt to indicate that 1) the interrupt failed, 2) the interrupt subprocess has been called, 3) the interrupt subprocess has been marked as such and will be called when one of its descendents (in the subprocess tree) becomes the current active subprocess (may be never).

9.3.3 Subprocess call - within a process

9.3.3.1 Parameters

9.3.3.1.1 C: class code

9.3.3.1.2 D: class code index

9.3.3.2 Error conditions

9.3.3.2.1 (1) The call stack does not have enough room to allow the subprocess (AP1,AP2) to be called (see 9.2.5.2.1).

9.3.3.2.2 (2) The subprocess specified by AP1 and AP2 does not exist.

9.3.3.2.3 (3) The number of actual parameters to be passed (see 9.3.3.3.4) exceeds the length of the C-list of the subprocess.

9.3.3.3 Action of a subprocess call

9.3.3.3.1 The P-counter of the entry on the top of the call stack (i.e., the entry for the subprocess making the subprocess call) is set to the next instruction to be executed by the calling subprocess.

9.3.3.3.2 A new entry is made on the call stack for the subprocess being called (AP2). This includes the subprocess name, the "top of path" as computed for the new subprocess, and the entry point (P-counter) of the new subprocess. The operation name and op count of the new stack entry are initialized to "null".

9.3.3.3.3 Changes in the full map and full C-list are made as needed and, as necessary, old maps are swapped out and new ones swapped in.

9.3.3.3.4 The actual parameter list from the operation (a subprocess may only be called as the action of an operation) is passed to the subprocess.

9.3.3.3.4.1 Actual parameters which are capabilities are copied, in the order they appear as actual parameters, to the beginning of the full C-list of the called subprocess.

9.3.3.3.4.2 Actual parameters are copied, preserving their order, to the address space of the called subprocess following the 4 word information heading. Actual parameters which are data are copied unchanged. Actual parameters which are capabilities are replaced with their indices in the full C-list (see 9.3.3.3.4.1).

9.3.3.3.5 The information heading consists of 1) an indication of the type of subprocess entry ("normal call" in this case); 2) the origin of the previous subprocess in the full map, the full C-list, and the address space. If the previous subprocess is not in new "full path", these origins are set to zero.

9.3.3.3.6 If the "interrupt waiting" counter of the process is non-zero, the "path" of the called subprocess is scanned for subprocesses which are marked as "awaiting interrupt". If the "interrupt waiting" counter is zero or, there are no such subprocesses in the path of the called subprocess, the called subprocess is allowed to run. If a subprocess(es) are found, the interrupt subprocess closest to the root of the tree is called (i.e., stack entry, map changes, information heading with entry type = "interrupt", etc.) with the datum which was saved (when the subprocess became an interrupt subprocess (see 9.3.2.3.4). The "interrupt waiting" counter is decremented and the interrupt subprocess is allowed to run.

9.3.4 Subprocess jump

9.3.4.1 Parameters

9.3.4.1.1 C: class code

9.3.4.1.2 D: class code index

9.3.4.2 Error conditions

9.3.4.2.1 The call stack, after the removal of the top of the stack, does not have room to allow the specified subprocess (AP1,AP2) to be called.

9.3.4.2.2 The subprocess specified by AP1 and AP2 does not exist.

9.3.4.3 Action of a subprocess jump

9.3.4.3.1 The top of the call stack is removed (i.e., the entry for the subprocess calling for the subprocess jump) and the active counter of its (the top of the stack's) subprocess is decremented.

9.3.4.3.2 A new entry is made on the call stack as in 9.3.3.3.2.

9.3.4.3.3 Full map and full C-list changes are made as needed (see 9.3.3.3.3).

9.3.4.3.4 The actual parameter list is passed to the subprocess (see 9.3.3.3.4).

9.3.4.3.5 An information heading reflecting a jump call and giving origins for the subprocess next in the call stack (if that subprocess is in the full path) is constructed.

9.3.4.3.6 If the "interrupt waiting" counter of the process is non-zero, action is taken as in 9.3.3.3.6. Otherwise, the called subprocess is allowed to run.

9.3.5 Subprocess return (normal)

9.3.5.1 The current subprocess (subprocess calling for the return) is removed from the top of the call stack.

9.3.5.2 If the "interrupt waiting" counter of the process is non-zero, the "path" from the new top of stack is scanned for subprocesses which are marked as "awaiting interrupt". If none are found, proceed to 9.3.5.3. Otherwise, the subprocess with "awaiting interrupt" which is closest to the root of the tree and on the path of the top of stack is called (i.e., call stack entry, full map and C-list changes, information heading with type of entry = interrupt, etc.) with the parameter which was saved when the subprocess became an interrupt subprocess (see 9.3.2.3.4) Also, the "interrupt waiting" counter is decremented.

9.3.5.3 A check is made for forced F return (see 9.3.6.2) and if such is indicated, such is done. Changes in the full map and full C-list are made as needed and, as necessary, old maps are swapped out and new ones are swapped in to conform to the new top of stack.

9.3.5.4 The subprocess which is the new top of call stack now runs. Execution commences at the location specified by its P-counter in the call stack plus the value (may be negative) of the lower 18 bits of the last word executed (the CEJ) by the subprocess. If the operation in the stack entry for this subprocess is "null", the P-counter is not modified. An error will be generated if the modified P-counter is outside the address space of the subprocess.

9.3.6 Subprocess failure return (F return)

9.3.6.1 The current subprocess (subprocess calling for the F return) is removed from the top of the call stack.

9.3.6.2 If the "interrupt waiting" counter is non-zero, then action is taken as in 9.3.5.2. In addition, the top of the stack is marked to cause a forced F return next time a return is made to it.

9.3.6.3 If the op count in the new top of stack is less than the order of the named operation, the operation is invoked to make another subprocess call and to pass parameters.

9.3.6.4 If the op count is equal to the order of the named operation, the operation name is set to "null". Then, execution of the subprocess on the top of the stack commences at the location specified by its P-counter in the call stack.

9.3.7 Jump return – stack cleanup

9.3.7.1 Parameters

9.3.7.1.1 C: class code

9.3.7.1.2 D: class code index

9.3.7.1.3 D: count

9.3.7.2 An error is generated if

9.3.7.2.1 The subprocess specified by AP1 and AP2 does not have AP3 incarnations in the call stack.

9.3.7.2.2 AP3 is not greater than zero.

9.3.7.3 The call stack is scanned from the top for the AP3-th occurrence of the subprocess specified by AP1 and AP2. If, during this scan, any subprocess is encountered which is in the "path" (see 9.2.7.1) of the calling subprocess (top of stack), an F return is made. Otherwise, the call

stack is "poped" until the AP3-rd occurrence of the subprocess specified by AP1 and AP2 and a RETURN is made to this subprocess. The RETURN includes checking for possible interrupts and forced F return.

9.3.8 Display a process

9.3.8.1 Parameters

9.3.8.1.1 C: a class code for a process

9.3.8.1.2 D: an address in subprocess memory

9.3.8.1.3 D: word count

9.3.8.2 Error conditions

9.3.8.2.1 AP1 + AP3 lies outside the address space of the calling subprocess.

9.3.8.3 As much of the following is copied to the subprocess address space as is allowed by AP3. ~~Further, it is copied~~ in the order indicated;

1) exchange package
2) Waiting interrupt counter
3) Non-empty waiting queue chaining words (along with a count of how many of these there are)
4) RA+1 interrupt subprocess class code and class code index
5) Number of entries in the call stack followed by the contents of the stack starting from the top.
6) The subprocess tree in the form: [subprocess name (class code + index): father's name].

File

10.1 A file is a collection of addressable data words

10.2 The basic element of a file is block. Each block has an address. A block of size N with address A is a seq of data words addressed: A, A+1, ..., A+(N-1).

10.3 A file is also composed of one or more nodes. Each node has an address and shape

10.3.1 A node of address A and shape (N) is either empty or a block of address A and size N.

10.3.2 A node of address A and shape (M,N) is either empty or a seq of M nodes, each of shape (N), and addresses A, A+N, A+2*N, ..., A + (M-1)*N.

10.3.3 A node of address A and shape (L,M,N) is either empty or a seq of L nodes, each of shape (M,N) and address A, A+(M*N), ..., A + (L-1)*(M*N).

10.4 A file is a node of address 0 and of shape (N), (M,N), or (L,M,N). Files of these different shape nodes are called respectively "1-level", "2-level", and "3-level" files.

10.5 Actions on files

10.5.1 read a seq of words

10.5.1.1 Parameters

1. C: a file
2. D; an address in the file
3. D; an address in subprocess memory
4. D; a count

10.5.1.2 AP4 words are copied sequentially from the file AP1 at address AP2 to relative CM address AP3. If an empty block is reached, an F return is made. An error is generated if the subprocess field length is exceeded, or the size of the file is exceeded.

10.5.2 Write a seq of words

Same as 10.5.1 except words are copied from CM to the file.

10.5.3 Copy a block

10.5.3.1 Parameters

1. C: a file
2. D: an address within a block in the file
3. C: a file
4. D: an address within a block in the file

10.5.3.2 Action

10.5.3.2.1 An error is generated if the 2 files have different block sizes, or if the addressed block in file AP1 doesn't exist (would be in an empty node) or if the addressed block in file AP2 does exist. An error may also be generated if the allocation block to which file AP3 is attached contains insufficient resources.

10.5.3.2.2 The addressed block (block containing the address AP2) in file AP1 is removed and moved to the addressed block in file AP3. Expected changes in empty-nonempty status of nodes in both files are made.

10.5.4 Create a new block

10.5.4.1 Parameters

1. C: a file
2. D: an address within a block in the file

10.5.4.2 Action

An error is generated if the addressed block already exists, or is beyond the limit of the file; otherwise block of zeros is placed in the file with address AP2. An error will be generated if the allocation block to which the file belongs contains insufficient resources.

10.5.5 Read shape

10.5.5.1 Parameters

1. C: a file
2. D: an address in subprocess memory

10.5.5.2 Action

The 3 interegers which describe the shape of AP1 are placed in the subprocess memory starting at AP2.

10.5.6 Check block

10.5.6.1 Parameters

1. C: a file
2. D: an address within a block

10.5.6.2 Action

X6 is set to 0 if the block does not exist; 1 if it exists and is empty; and 2 if it exists and is nonempty.

10.5.7 Delete block

10.5.7.1 Parameters

1. C: a file
2. D: an address within a block

10.5.7.2 Action

An error is generated if the addressed block does not exist, or is beyond the limit of the file; otherwise the node at

the block is set to empty, and if this causes all of the subnodes of some node to be empty, it also becomes empty, etc.

10.5.8 Create a file

10.5.8.1 Parameters

10.5.8.1.1 C: an allocation block

10.5.8.1.2 D: an index in the full C-list to return the capability for the new file

10.5.8.1.3 D: number of levels in the file (1,2, or 3)

10.5.8.1.4 D: sizes of the level(s)

for a 1 level file: the size N of the single block

for a 2 level file: ((the number of blocks) * 2^{18}) + (the size of the blocks 'L' such that block size will be 2^{L})

for a 3 level file: (((the number of M,N nodes) * 2^{18}) + (the number of blocks 'K' in each node such that there will be 2^{K}=M blocks per M,N node)) * 2^{18} + (the size of the file blocks 'J' such that block size will be 2^{J}=N)

10.5.8.2 Error conditions

10.5.8.2.1 Allocation block AP1 contains insufficient resources to fund the proposed file

10.5.8.2.2 AP2 specified a non-existent location in the full C-list, or a location not containing a "null" capability.

10.5.8.2.3 AP3 is not equal to either 1, 2, Or 3.

10.5.8.2.4 AP4 is not in the proper format to correspond with AP3 or contains overly large numbers (maximums will be published).

10.5.8.3 A file with the structure specified by AP3 is created with nodes of the sizes specified by AP4. All blocks are non-existent. A complete capability (all options allowed) is placed at the location specified by AP2.

Allocation Block

11.1 An allocation block consists of the following items:

I1. number of words of "allocated" ECS space

I2. number of words of "in use" ECS space

I3. a list of objects tied to this allocation block.

(The total ECS space occupied by these objects = I2.)

I4. "Allocated" money

I5. Money used for ECS space/time

I6. Money committed for CPU time

I7. Last time at which I5 was updated

11.2 Actions on an allocation block

11.2.1 Construct an object using ECS space S

I5 := I5 + (current time - I7) * I1 * space time conv

If (I4 .LT. I5 + I6) then ran out of money

I7 := current time

If (I1 .LT. I2 + S) then error

I2 := I2 + S

Place new object on list I3

Point the new object at this allocation block

11.2.2 Destroy an object using ECS space S that points at this allocation block

Remove the object from list I3

I5 := I5 + (current time - I7) * I1 + space time conv

If (I4 .LT. I5 + I6) then ran out of money

I7 := current time

I2 := I2 - S

NOTE: If the object destroyed is an allocation block, the following is also done (its parameters represented by E_i):

Destroy all objects on list E3

I1 := I1 + E1

I4 := I4 + E4

I5 := I5 + E5 + (current time - E7) * E1 + space time conv

I6 := I6 + E6

If (I4 .LT. I5 + I6) then ran out of money

11.2.3 Move allocated items

11.2.3.1 Parameters

1. C: 1st allocation block
2. D: allocated ECS space to move
3. D: allocated money to move
4. D: money used for ECS space time to move
5. C: 2nd allocation block

11.2.3.2 Action

(Use D_i to represent items in 1st allocation block (AP1)

E_i to represent items in 2nd allocation block (AP5))

If (D1 - AP2 .LT. D2) then error

If (D4-(D5+D6+((current time-D7)*D1*space time conv)).LT.AP3)

If (D5 .LT. AP4) then error

D1 := D1 - AP2

E1 := E1 + AP2

D4 := D4 - AP3

E4 := E4 + AP3

D5 := D5 - AP4

E5 := E5 + AP4

IF (D4 .LT. D5 + D6) then ran out of money

IF (E4 .LT. E5 + E6) then ran out of money

11.2.4 **Create an allocation block**

11.2.4.1 Parameters

11.2.4.1.1 C: an allocation block

11.2.4.1.2 D: a location in the full C-list to return a capability for the new allocation block

11.2.4.2 Error conditions

11.2.4.2.1 AP1 has insufficient resources to fund an allocation block

11.2.4.2.2 AP2 does not contain a "null" capability

11.2.4.3 A new allocation block is created by the authority of AP1. A capability (with all options allowed) is placed at the location specified by AP2. The parameters of the new allocation block are initialized as follows:

I1 := 0 I4 := 0 I7 := current time

I2 := 0 I5 := 0

I3 empty I6 := 0