II Map Actions

Associated with each subprocess is a map which directs the swapping of the subprocess address space between Central Memory and ECS files. A map consists of a fixed length sequence of map entries each of which is either zero or contains a swapping directive. The user may zero or change a map entry, and may display an entry from the map associated with any given subprocess or from the full map. A non-zero map entry consists of 1) a file, 2) an ECS file address, 3) a central memory address, 4) a word count, and 5) a read-only flag. Thus the map indicates what portions of which files are copied to/from specified portions of the subprocess space at the beginning/end of processing.

A. Zero a Map Entry

AP1	C: Class code (subprocess name) (OB.CHAMP)
AP2	D: Index in logical map of the subprocess

When zeroing a map entry, the user specifies the name of the subprocess (class code) whose map entry is to be zeroed, and the index of the entry in the subprocess; logical map. The result is that when the subprocess address space is swapped between ECS and Central Memory, the file block formerly referenced by the zeroed entry will not be swapped.

Possible errors while zeroing a map entry:

Class	#	Param #	Description
4	5		Subprocess does not exist
2	2	2	Negative map index
2	3	2	Map index exceeds map length
11	0		Attempt to change or zero DAE

B. Change (create) a map entry (read/write or read only)

AP1	C:	Class code of subprocess whose entry is to be changed (OB.CHMAP
AP2	D:	Index of entry in logical map of subprocess
AP3	C:	Associated file (read only: OB.PLMAP,OB.RDFIL; read/write: { OB.PUMAP, OB.RDFIL, OB.WFILE
AP4	D:	Address in file which is to be changed
AP5	D:	Address in CM of new entry
AP6	D:	Word count of new entry

When a map entry is changed, care must be taken if the map involved is part of the full map. In this case, a non-zero map entry must be swapped out before it is replaced. The new entry is then constructed and swapped in. Note that overlapping map entries will behave oddly since the whole map is not swapped.

Possible errors while changing a map entry:

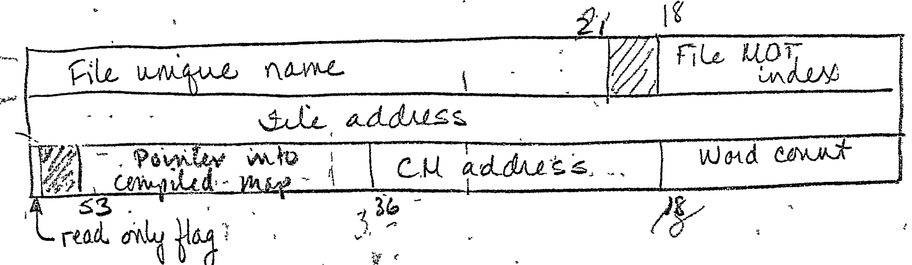
Class	#	Param #	Description
4	5		Subprocess does not exist
2	2	2	Negative map index
2	3	2	Map index exceeds map length
4	3		Buffer full
2	2	5	Negative CM address
2	0	6	Negative word count

C. <u>Display a Map Entry from the Map of a Named Subprocess</u>

AP1	C:	Class code of subprocess whose entry is to be displayed
AP2	D:	Index of entry in Logical map of subprocess
AP3	D:	Address of a 3 word buffer

This action will insert into the 3 word buffer area (AP3) the current contents of the indicated map entry of the subprocess specified. Note that the length of the map (maximum for AP2) can be obtained by using the Display Subprocess action. The three words of the designated map to the specified buffer.

Map Entry



Possible errors while displaying a map entry:

Class	#	Param	Description
4	5		Subprocess does not exist
2	2	3	Negative address for buffer
2	2	3?	Write area exceeds user fl
2	2	2	Negative map index
2	1	2	Map index too large

D. Display Entry in Full Map

AP1	D:	Index of entry in full map
AP2	D:	Address of a 3 word buffer

The maps of the subprocesses in the full path are concatenated to form the full map in much the same way as the full C-list is formed. An entry in the full map can be displayed if the index of the entry in the full map is given along with the address of a buffer where the entry should be "displayed". The format of the displayed entry is the same as for named subprocess version of Display Map Entry.

Class	#	Param	Description
2	3	1	Index of entry too large (exceeds length of full map)
2	2	1	Index of entry negative
2	2	2	Pointer to buffer negative
2	3	2	Pointer to buffer + 3 greater than user's field length

III Event Channel Actions

Event channels are ECS objects which are used to synchronize the behavior of running processes as well as to implement "block" and "wake-up" mechanisms. Each event channel should handle a particular kind of event. The user can create an event channel, send an event, get an event from an event channel, get an event from any one of a list of event channels, and destroy an event channel. If the user attempts to get an event from a channel which has no events, the user's process is either blocked (stops running) until some other process sends an event to the event channel, or F-return action is initiated.

A. Create an Event Channel

AP1	C: Capability for allocation block (OB.CREEC)
AP2	D: C-list index for new event channel capability
AP3	D: Length of event queue

When an event channel is created it consists of a three word header and an event queue (see Event Channels) which is initially empty. The header word is used to maintain the queue of events and a queue of waiting processes, which develops if the queue of events becomes empty. When creating an event channel, the user specifies the name of the Allocation block which funds the ECS space occupied by the event channel, a C-list index where the system can put the capability (with all options allowed) for the event channel when it creates it, and the length (number of events) of the event quene.

Possible errors while creating an event channel

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index exceeds full C-list
9	0	Length of event queue ≤ 0
9	1	Event queue too large

B. Send an Event (with/without duplicate event checking)

AP1	C:	Capability for the event channel	OB.SNDEV without checking OB.SDEVX with checking
AP2	D:	Datum part of event	

To send an event to an event channel, the user specifies the index of the capability for the event channel and specifies a 60 bit datum to be passed with the event. The system responds by indicating the disposition of the event to the user in X6. the following responses are possible:

Condition	Response
Event put in event queue	1
Event passed to a process	2
"YOU LOSE" event put in event queue	3
Event queue full	4
Duplicate event found	5

The first response indicates that all went well, and there was no process awaiting an event in the process queue. The second response indicates that there was a process waiting in the queue and that it was passed the event. The third response indicates that there was only one free slot in the event queue (an event occupies two words); the intended datum has been replaced by a "you lose" datum (-0) so that the process which ultimately gets the datum will be aware that the event queue had been full and that information was lost.

The fourth response indicates that no action has been taken since the queue was full. The fifth response is returned only if a search for duplicate events was desired and a duplicate was found in which case no further action is taken.

Possible errors resulting from sending an event.

Class	#	Description
9	2	Event channel does not exist

C. Get an event or hang

AP1 C: Capability for event channel (OB.GETEV)

A user requests an event from a channel using the C-index of the channel in question. If the event queue is empty, the process must wait ("hang" or "Block") until an event arrives before resuming execution. If more than one process is awaiting, an event sent to that channel is passed to the first process while the other process(es) continues to wait.

x

Possible error while getting an event:

Class	#	Description
9	2	Event channel does not exist

D. Get an Event or F-return

AP1 C: Capability for event channel (OB.GTEVF)

The user requests an event from a channel using the C-list index of the event channel's capability. If the event queue is empty, an F-return will be returned in order to permit the process it take alternative action.

The only possilbe error is the same as for C above.

E. Get an event from one of a list of event channels or hang (F-return)

~~[illegible]~~ AP1 c: capab for event chan

AP2

The procedure for getting an event from one of a list of event channels is similar to that for getting a single event (See C above). The channels are interrogated one at a time and if their respective event queue is empty, the user's process will be queued on the process queue of the event channel. If an event subsequently arrives or is discovered on one of the event channels in the list, the process is removed from all the process queues on which it has already been chained and it is passed the event. If no event arrives or is discovered before the last event channel is interrogated, the process must wait ("hang" or "block") until an event arrives on one of the event channels.

Possible errors while getting an event from a list of channels:

Class	#	Description
9	2	Event channel does not exist
2	0	Number of channels is negative
2	1	Number of channels is too large

F. Destroy an Event Channel

AP1 C: Capability for event channel (OB.DSTRY)

An event channel can be destroyed. The only paramemeter required is the C-list index of the event channel which is to be destroyed. If there are any processes waiting on the event channel's process queue, an F-return is initiated leaving the event channel intact.

Possible errors while destroying an event channel:

Class	#	Description
9	2	Event channel does not exist

IV C-List Actions

User access to all objects within the ECS system is controlled by capabilities. A capability identifies the object it refers to, specified the type of the object, and the set of allowed actions on that object (options). Capabilities are grouped together in capability-lists (C-lists) which are themselves objects within the ECS system. Individual capabilities are referred to by their index within a C-list. Since the capability, residing in a C-list, authorizes access to an object, the user is never allowed to fabricate a capability. The system creates a capability with all options allowed when an object is created. System actions are provided to permit the user to create a C-list, examine a capability, to copy capabilities between C-lists and within a C-list, and to downgrade the option mask. Thus, the user can transfer the right to access an object and can curtail that access, but he may never manufacture that right or increase the set of allowable actions on the object.

A C-list, which initially consists of only empty positions, is assigned to every subprocess within a process. For every process there exists a sequence of subprocesses called the full path. Corresponding to the full path, the full C-list is defined as the concatenation of the C-lists belonging to the subprocesses in the full path. When referring to capabilities in the full C-list, the capability index is interpreted as if the C-lists in the full C-list have been joined to form one long C-list.

A. Create a C-list

AP1	C: Capability for allocation block (OB.CPECL)
AP2	D: Index in full C-list to return new capability
AP3	D: Length of new C-list

A capability list (C-list) is a sequence of capabilities and "empty" positions. Each C-list is filled with "empties" (zero words) upon creation. To create a capability list, the user mstt supply the index of the Allocation block which funds the space occupied by the C-list. In addition to the length of the new C-list, the user must supply an index in the full C-list for the capability for the new C-list.

Possible errors while creaing a C-list.

Class	#	Param	Description
6	0		Allocation block does not exist
6	1		No ECS available
6	2		No money available
2	4	2	C-list index is negative
2	5	2	C-list index exceeds full C-list
2	0	3	Length of new C-list $\leq$ 0
2	1	3	Length of new C-list too large

B. Display a Capability from the Full C-list

AP1 D: Index in full C-list

When referring to capabilities within the full C-list, the capability index used is interpreted as if the C-lists in the full C-list were joined to form one long C-list. Thus, the index of the desired capability is all that is required to display it. The two words of the capability are returned in X6 and X7.

X6 =	option mask	type
X7 =	unique name	MOT index

Possible errors while displaying a capability

Class	#	Description
2	4	Capability index negative
2	5	Capability index exceeds full C-list length
8	0	Capability list does not exist

C. Display a Capability from an arbitrary C-list

AP1 C: Capability for C-list
AP2 D: Index in the C-list

To display a capability from a C-list which is not in the full C-list, the user must specify both the index of the C-list and the index within the C-list of the desired capability.

Possible errors are identical to those given for B above.

D. Copy a Capability within full C-list and Decrease the Options

AP1	D: Index of desired capability
AP2	D: Index of destination C-list entry
AP3	D: Mask of options to preserve

The user can copy a capability from one C-list to another and in doing so may decrease the number of allowed options. Recall that when an object is created, a capability is returned which has all the option bits (the high order 42 bits of the first word) set. The user must indicate the C-list index of the capability he wishes to copy, the C-list index for the altered capability, and a bit-mask which will be logically "anded" with the options of the original capability to produce the option mask for the new copy of the capability.

Possible errors while copying a C-list and decreasing the options. See B above.

E. Copy capability from Full C-list to Arbitrary C-list (and vice-versa)

AP1	C: Index of destination (source) C-list (OP.CPYIN, OB.CPYOT)
AP2	D: Index within destination (source) C-list of capability
AP3	D: Index in the full C-list of source (destination) Capability

In order to simp transfer a capability between the full C-list and an arbitrary C-list two parameters are required to indicate the location of the capability in the aribtrary C-list, and a third to locate the capability in the full C-list.

Possible errors: See B above.

F. Destroy a C-list

AP1	C: Capability for C-list	(OB.DSTRY)

The user may destroy a C-list when he no longer needs it; only the index of a capability for the C-list is required. If the C-list to be destroyed is in the full path of the user's process, an F-return is initiated and the C-list is not destroyed.

Possible errors while destroying a C-list

Class	#	Description
5	0	C-list does not exist

V Operations

Operations are ECS objects which control the initiating of system action and subprocess calls. Each operation is composed of an initial order specifying a desired action, some checking information and, for subprocess call, a class code. The initial order is followed optionally by a sequence of orders (containing similar information) indicating alternative actions should all the preceding orders result in F-returns.

The checking information in each order consists of a parameter specification type for each parameter required in the actual parameter list for the indicated action, a word containing the option bits and type for each capability parameter to be supplied by the user, and all fixed parameters, whether capabilities or data. This checking information is used by the system entry/exit routines when constructing the actual parameter list.

The parameter specification types are:

Type	Description
any	when an operation is created, all parameter specifications are initialized to "any", and must be fixed using the various action supplied before the operation may be used.
"any capability"	a capability is expected from the user, but no type or option checking is to be performed on it.
"user-supplied capability"	the user must supply a capability whose type and option bits match those stored in the operation.
"user-supplied datum"	the user must supply a 60-bit datum but no checking is performed on it
"fixed capability"	both words of a capability are stored in the operation and no corresponding information is taken from the user's input parameter list.
"fixed datum"	a 60-bit datum word is stored in the operation; it is passed to the actual parameter list unchanged.

There are two actions for creating new operations; the first creates an operation with one order to call or jump-call a designated subprocess, and the second creates an operation of order N by adding one order to an already existing operation of N-1 orders. All operations constructed by the user are subprocess calls. Actions are also available for copying an operation and for changing the parameter specifications for an operation.

A. Make a subprocess call or subprocess jump operation

AP1	C: Capability for allocation block (OB.ALORD)
AP2	D: C-list index for new operation
AP3	D: Type (0=call; nonzero=jump)
AP4	C: Class code of subprocess to be called by the new operation (OB.CALOP)
AP5	D: Number of parameters to be used by the subprocess call.

To create a new operation to be used for subprocess call or jump call, the user supplies the index of a capability for the Allocation block which is to fund space in ECS for the new operation.

In addition the user gives the C-list index where the system will place the capability for the new operation, the type action (call or jump call) of the new operation, the name (class code) of the sub-process to be called by the operation, and the number of parameter specifications needed for the subprocess call. Upon creation, all of the parameter specifications of an operation are initialized to "any", and therefore the operation may not yet be invoked (unless it is parameterless).

Possible errors while creating a new operation

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	0	Number of parameter specifications is negative
7	7	Too many parameters
2	4	Negative C-list index
2	5	C-list index too large
7	6	Order too large for scratch area
4	5	Subprocess does not exist

B. Add an Order to an Operation

AP1	C: Capability for allocation block (OP.ALORD)
AP2	D: C-list index for new operation (order)
AP3	C: Capability for existing operation (OB.ADDDR)
AP4	D: Type of order (0=call; nonzero=jump)
AP5	C: Class code of subprocess called by the new order (OB.CALOP)
AP6	D: Number of new parameters being added

This action creates an operation of order N . The first parameter is the C-list index for the Allocation block which is to fund space in ECS for the new operation; the second parameter is the C-list index where the system will put the capability for the new operation. In the third parameter the user specifies an already existing operation of order N-1 , which is copied with the new order appended. The last three parameters describe the new order by indicating whether it is a call or jump call to a subprocess, the name (class code) of the subprocess to be called, and the number of additional parameters. The parameters of the new order will be initialized to type "any" and must be fixed up before the new order of the operation is used.

Possible errors while adding an order

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index is too large
7	1	Operation has been deleted (doesn't exist)
4	5	Subprocess does not exist
2	0	Number of parameter specifications is negative
2	1	Number of parameter specifications is too large
7	6	Order exceeds scratch area

C. Copy an Operation

AP1	C: Capability forallocation block (OB.ALORD)
AP2	D: Full C-list index for new operation
AP3	C: Operation to copy

The user can copy an already existing operation by specifying the C-list index of the funding allocation block, the full C-list index for the desired operation, and the current C-list index of the desired operation. This action is used prior to fixing parameter specifications of an operation to avoid changing the original version of the operation.

Possible errors while copying an operation:

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index exceeds full C-list
7	1	Operation does not exist

D. Change a parameter specification type

In order to specify the parameter specification types in an order of an operation created by either A or B above, a set of actions is provided. Each takes as parameters a C-list index for an operation and a parameter specification index (considering, the parameter specification for the first parameter of the first order as having an index of 0). Some require additional information depending on the type of parameter specification being changed.

1. Change parameter specification from "any" to "user-supplied datum".

AP1 C: Capability for operation (OB.CHTYP)
AP2 D: Index of parameter specification to change

Possible errors:

Class	#	Description
7	1	Operation does not exist
2	2	Index is negative
2	3	Index is too large
7	4	Parameter specification type should be "any"

2. Change parameter specification for "any" to "any capability"

AP1 C: Capability for operation (OB.CHTYP)
AP2 D: Index of parameter specification to change

Possible errors: See 1 above.

3. Change parameter specification type from "any" to "user-supplied capability"

AP1	C:	Capability for operation (OB.CHTYP)
AP2	D:	Index of parameter specification type
AP3	D:	Capability type
AP4	D:	Capability option bit mask

The type of a capability occupies an 18 bit field of which exactly 9 of the 18 bits must be set. Table 1 below gives the types for ECS objects currently available.

Table 1. Capability types

Object	Type
Process	777_8
C-list	1377_8
File	1577_8
Operation	1677_8
Class Code	1737_8
Event Channel	1757_8
Allocation Block	1767_8

The option bit mask stored in a capability occupies a 42-bit field and the meanings of the various option bits is determined by the type of object the capability identifies. See Appendix B for the name, description and relative position of all option bits. The positions of the bits are given reading from right to left; thus bit position 0 is the low order bit of the field.

Possible errors while changin parameter specificaion type from "any" to "user-supplied capability":

Class	#	Param	Description
2	2	2	Index is negative
2	3	2	Index is too large
7	2	3	Capability type exceeds 9 bits
7	1		Operation does not exist
7	2	4	Option bits bad
7	4		Parameter specification should be "any"

4. Change a parameter specification type from "user-supplied datum" to "fixed datum"

AP1	C:	Capability for operation (OB.CHTYP)
AP2	D:	Index of parameter specification type
AP3	D:	60-bit datum word

Possible errors while changing parameter specification from "user-supplied datum" to "fixed datum"

Class	#	Param	Description
7	1		Operation does not exist
2	2	2	Index is negative
2	2	3	Index is too large
7	5		Parameter specification should be user-supplied

5. Change a parameter specification type from "user-supplied capability" to "fixed-capability"

AP1	C:	Capability for operation (OB.CHTYP)
AP2	D:	Index of parameter specification type in o· ration
AP3	C:	A capability

Possible errors while changing a parameter specification type from "user-supplied capability" to "fixed capability"

Class	#	Param	Description
7	1		Operation does not exist
2	2	2	Index is negative
2	2	3	Index is too large
7	5		Parameter specification should be user-supplied

Note in the last two cases (4 and 5) that "fixing" a parameter specification type requires two steps, changing the specification first to a user-supplied type and then to the corresponding fixed type.

Actions 3, 4, and 5 involve reallocating the operation in ECS, since each requires inserting one additional word to the order.

VI Process and Subprocess Actions

Processes are the active elements of the ECS portion of the Time Sharing System. Only within the context of a process may code be executed and system actions initiated. A process consists of a set of central registers call the exchange jump package, a set of subprocesses organized in a tree structure, a call stack recording the flow of control among the subprocesses, and a set of state flags describing the state of the process.

There are system actions to create, examine, destroy and manipulate the elements of a process. There are also actions which control the processing environment of a process by transferring control from one subprocess to another and by controlling the error processing and external interrupt status of the process.

A. Create a Class code (subprocess name) with new permanent part

AP1	C: Capability for class code

A class code is a protected 60-bit datum used to identify a subprocess within a process. The 60 bits are divided into two 30-bit parts; the upper 30 bits constitute the permanent part and the lower 30 bits, the temporary part. This action causes a new class code to be constructed by the system with a permanent part which is different from that of all other class codes. The new class code is returned in the full C-list at the location specified by the parameter of the action.

Possible errors while creating a class code:

Only those detected during System entry/exit

B. Set temporary part of class code

AP1	C: Capability for class code (OB.TEMP)
AP2	D: C-list index for modified class code
AP3	D: New temporary part (30 bits)

The emporary part specified by the user is inserted into the class code (lower 30 bits). It may be used to create "classes" of class codes which have the same permanent part and different temporary parts. The class code with the new temporary part is returned in the full C-list at the specified location.

C. Create a Process

AP1	C: Capability for Allocation block (OB.CREPR)
AP2	D: C-list index for returned process capability
AP3	D: Number of event channel chaining words
AP4	D: Number of stack entries
AP5	C: Class code for initial subprocess (OB.SONSP)
AP6	D: Number of map entries in initial subprocess
AP7	D: Compiled map buffer size for initial subprocess
AP8	D: Subprocess field length
AP9	D: Subprocess entry point
AP10	C: Capability of C-list for subprocess (OB.LOCCL)
AP11	C: Capability of file for 1st map entry (Read/Write: OB.WFILE, OB.RDFIL, OB.PLMAP) for initial subprocess
AP12	D: Address within file
AP13	D: Address in CM
AP14	D: Count of words to be swapped
AP15	D: Capability fo file for 2nd map entry (Read Only: OB.RDFIL, OB.PLMAP) for initial subprocess
AP16	D: Address within file
AP17	D: Address in CM
AP18	D: Count of words to be swapped

There are 18 prameters required for the system action which creates a process. The first cour are used to consutrct the process descriptor while the remaining 14 are necessary to specify the initial subprocess which is created along with the process. As usual when creating any system object, the first two parameters required are the C-list index of the Allocation block which is to fund the area in ECS where the object is to be placed, and the C-list index where the system will return the capability for the object. Parameters 3 and 4 determine the size of the process queuing word buffer (number of event channels the process can hang on at once) and the length of the call stack which records the flow of control among the subprocess belonging to the process. Each entry in the call stack contains the information necessary to reinitiate processing where it was terminated due to a subprocess call.

Among the parameters defining the initial subprocess, the first six (AP5-AP10) are used to fill in the subprocess descriptor and the last eight specify the contents of the two map entries (Read/Write and Read Only) which define the local address space (portions of storage available to it) of the initial subprocess. The data necessary to describe a subprocess are gathered into the process descriptor. The user supplies the class code (identifying name) of the subprocess, the number of entries which will be in the logical map, the length of a buffer area which will contain the compiled map, the length of the subprocess local address space, the entry point in the subprocess where execution begins when it is called and a C-list index for a capability for a C-list (local C-list). The logical map contains an entry for each contiguous portion of information which is to be copied between ECS files and the local address space in CM of a subprocess at the beginning and/or end of the processing of that subprocess. To expedite this procedure, the compiled map is generated from the logical map, using the absolute ECS addresses of the sections of ECS files referenced by the logical map entries. Since one map entry may span several data blocks in a file, the size of the compiled form of the map will increase accordingly. The length of the local address space (AP8) of a subprocess is the upper limit on the information copied in CM under the direction of the subprocesses map. The local C-list (AP10) of a subprocess controls the objects which it can access.

The eight remaining parameters specify the contents of the two map entries, which indicate the initial body of the subprocess. The first map entry (specified by parameters AP11-AP14) defines a portion of an ECS file which is copied into CM before processing under the control of the subprocess is initiated and when this processing stops is copied back into the ECS file from which it came, thereby (possibly) altering the content of the ECS file. The second map entry, however, defines a section of an ECS file which is read into CM only, and will never be copied back into ECS, thus protecting the ECS file from being altered. The parameters include the C-list index of the associated ECS file, the addresses in the file and in CM between which the information is to be transfered (swapped) and the number of words to be swapped.

The new process, after being constructed, is scheduled to run and will begin execution at the entry point of the initial process.

Possible errors while creating a process:

Class	#	Param #	Description
6	0		Allocation block does not exist
6	1		No ECS available
6	2		No money available
2	4		C-list index is negative
2	5		C-list index is too large
2	0	3	Number of chaining words $\leqq$ 0
2	1	3	Number of chaining words too large
2	0	4	Number of stack entries < 1
2	0	6	Number of map entries < 2
2	0	7	Compiled map buffer size is negative
2	0	8	Length of local address space is negative
2	1	8	Length of local address space is too large
2	0	9 or 15	Subprocess entry point < 2
2	1	9 or 15	Subprocess entry point exceeds field length
2	2	12 or 16	File address is negative
2	3	12 or 16	File address too large
2	2	13 or 17	CM address is negative
2	3	13 or 17	CM address exceeds field length
2	0	14 or 18	Word count for map entry < 0
2	1	14 or 18	Word count for map entry too large

D. Creating a Subprocess

AP1	C: New subprocess class code (OB.SONSP)
AP2	C: Class code of the "father" of the subprocess (OB.FATHR)
AP3	D: Number of map entries
AP4	D: Compiled map buffer size
AP5	D: Subprocess FL
AP6	D: Subprocess entry point
AP7	C: Subprocess local C-list index (OB.LOCCL)

The action of creating a subprocess involves constructing the 8 word process descriptor. The parameters are similar to those required to create the initial subprocess except for AP2 and the absence of map entry parameters. The subprocesses in a process are organized in a true structure in which each subprocess references only is predecessor ("father"). For each subprocess, the term "ancestors" refers to the sequence of subprocesses which starts with the subprocess and terminates with the root of the subprocess tree. Note that a subprocess is always an "ancestor" of itself. The term "son" of a subprocess refers to any of the subprocess for which that subprocess is the "father".

Each newly created subprocess is linked with the subprocess tree at the subprocess referenced by AP2. Note that since no map entries are made for the subprocess at the time of its creation, they must be constructed via a system action to provide executable code and data for the subprocess, before the subprocess can be used.

Possible errors while creating a subprocess

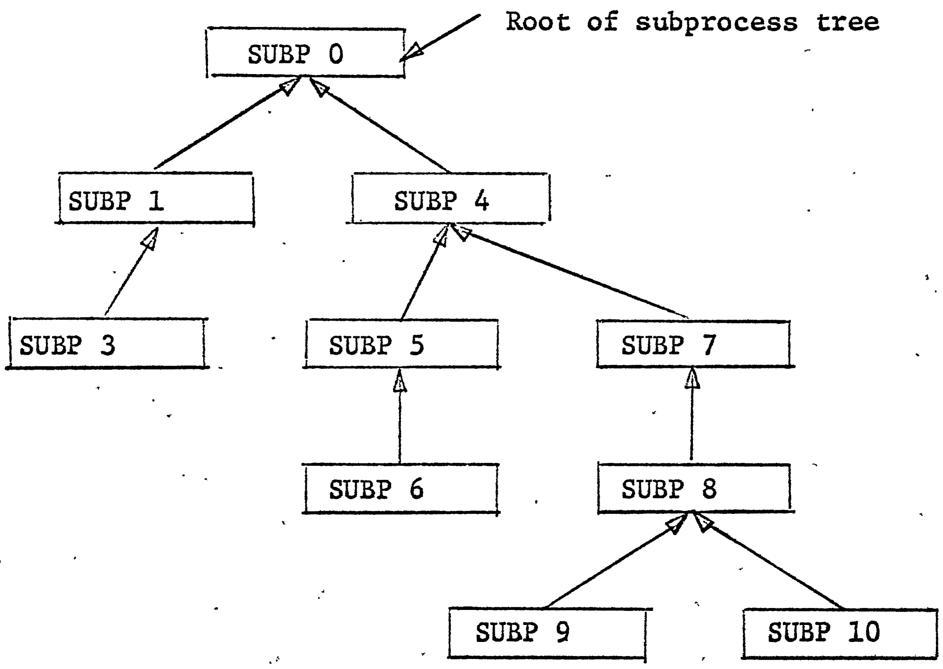
Class	#	Param #	Description
6	0		Allocation block does not exist
6	1		No ECS available
6	2		No money available
4	0		Duplicate subprocess name
4	1		"Father" does not exist
2	0	3	Number of map entries ≦ 0
2	1	3	Number of map entries exceeds field length
2	0	4	Compiled map buffer size is negative
2	0	5	Subprocess field length < 0
2	1	5	Subprocess field length is too large
2	0	6	Entry point < 2
2	1	6	Entry point > FL
4	3		Nospace for compiled map
8	0		C-list does not exist
4	4		Process becomes too big

E. Subprocess Call

A normal subprocess call is initiated by calling on the system in the usual manner, using an operation whose action is "subprocess call". A normal subprocess call may also be initiated as the result of F-return action under the control of a multi-ordered operation (see p. above). A new processing environment is established (described below) as a result of the transfer of control to a different subprocess. At any given time, there are two distinguished subprocesses within a subprocess. They are the current subprocess and the end-of-path subprocess. The current subprocess is always an "ancestor" of the end-of-path subprocess; the sequence of subprocesses from the end-of-path to the current subprocess (inclusive) is called the full path. The end-of-path is difined dynamically by the flow of control among the subprocesses. The current subprocess may be considered to be the subprocess currently in control. The end-of-path and current subprocesses are reassigned whenever a new subprocess is called. The subprocess being called the callee becomes the new current subprocess. If the callee is an "ancestor" of the old end-of-path, the new end-of-path becomes the same as the callee (i.e., the full path consists of a single subprocess--the callee). See Figure .

The full path defines the sphere of protection invoked by the current sub-process. The access into the current subprocess permitted to other objects within the system is controlled by the full C-list, the full map, and the full address space, each of which is defined by the full path. The full map determines the configuration of the address space available to the current subprocess. The configuration of the subprocess tree defines the static relationship between the subprocess (subprocesses closer to the root may be given the privileges of their descendents) while the full path dynamically controls the boundaries of access applied to the current subprocess. This system of controlling the bounds of protection allows the construction of processes which may exercise varying degrees of protection while maintaining synchronization between the subprocesses involved.

SUBPROCESS TREE



FULL PATH EXAMPLE

CALLING SEQUENCE			CURRENT SUBP	END-OF-PATH SUBP	FULL PATH
		SUBPO	SUBPO	SUBPO	SUBPO
SUBPO	calls	SUBP9	SUBP9	SUBP9	SUBP9
SUBP9	calls	SUBP6	SUBP6	SUBP6	SUBP6
SUBP6	calls	SUBP4	SUBP4	SUBP6	SUBP6,5,4
SUBP4	calls	SUBPO	SUBPO	SUBP6	SUBP6,5,4,0
SUBPO	calls	SUBP5	SUBP5	SUBP6	SUBP6,5
SUBP5	calls	SUBP3	SUBP3	SUBP3	SUBP3

A subprocess call also causes a new stack entry to be constructed and placed on the call stack. In addition, the origins (relative to the local environment) of the address space, C-list, and map of the calling subprocess are computed and stored in cells 2, 3, and 4 of the local address space. If the calling subprocess is not a member of the new full-path, then these cells are zeroed. Starting in cell 5 of the local address space are copied the parameters of the subprocess call.

For a normal call the parameters of the call are first formatted in the actual paramente area of the process descriptor by the system entry mechanism. These parameters are drawn from the user's input parameter list (IP list) under the direction of the operation being used for the subprocess call (IPO). In addition, the system entry routine places the name (class code) of the called subprocess at the number of parameters and a bit string denoting the types of the types of the parameters at the end of the actual parameter area. After establishing the correct processing environment for the called subprocess, the parameters are transfered to the local address space and local C-list of the called subprocess. Datum parameters are simply copied to the next parameter cell in the local address space. Capability parameters are copied to successive positions in the local C-list and the index of the parameter in the local C-list is stored in the next parameter cell in the local address space. On the completion of the parameter passing, execution is initiated at the entry point of the called subprocess.

Possible errors during subprocess call:

Class	#	Param #	Description
4	5		Named subprocess does not exist
4	6		No room on stack for subprocess
4	7		No room for parameters
4	8		Too many capability parameters
8	0		C-list does not exist

Subprocess Return

Like the subprocess call, the subprocess return must construct a new processing environment before returning control to the user. The return routines reactivate a subprocess using information left in a stack entry. The full path recorded in the stack entry is sufficient to reconstruct the processing environment. The P-counter from the stack entry control where in the subprocess execution is initiated. The normal return requires the P-counter to be modified by the low order 18 bits of the CEJ instruction which originally caused control to pass to another subprocess (see p. above).

Possible errors during subprocess return

Class	#	Description
4	9	Stack empty
2	2	P-counter < 0
2	3	P-counter exceeds field length

Subprocess F-return

If the execution of the called subprocess has resulted in an F-return, a flag is set in the call stack entry. Under this condition the return is similar to the normal return except that the last used operation (found by looking up the IP list pointer in the previous stack entry) is checked for the presence of further orders. If the F-return count is not equal to the number of orders in the operation which is also found in the stack entry, the next order is processed. Otherwise, control returns to the subprocess which called the operation one word above the subprocesses normal entry point.

Possible errors during a subprocess F-return:

Class	#	Description
4	10	Stack empty
7	0	IPO is not a capability for an operation
7	1	Operation does not exist
2	1	IP list is too big

G. Subprocess Jump Return

AP1	C: Class code of subprocess to return to (OB.SPRET)
AP2	D: Number of stock occurrences of AP1 to skip (0=1, -1 = down to last)

The subprocess jump return provides a method for getting calls off of the process call stack. The user specifies the class code of the subprocess to which the return is to be made. In addition, he indicates the number of occurrences of that subprocess in the call stack which should be skipped in looking for the call which is to become the new top of the stack. Zero indicates the first (most recent) call whereas -1 indicates the last (earliest) call.

call. Upon finding the proper stack entry, the stack is reduced to make that entry the top of stack and normal subprocess return action is initialized.

H. Modify P-counter of subprocess

AP1	C; Class code of subprocess (OB.PCNT)
AP2	D: Number of stack occurrences of AP1 to skip
AP3	D: New P-counter

The user can modify the P-counter in a subprocess which has already been called by identifying the subprocess, the number of stack occurrences of the subprocess to skip (see G above) and the new P-counter. The P-counter is modified in the stack and the new P-counter will be used the next time that entry becomes the top of the stack. If the caller attempts to modify his own P-counter an F-return is made.

Possible errors while modifying the P-counter

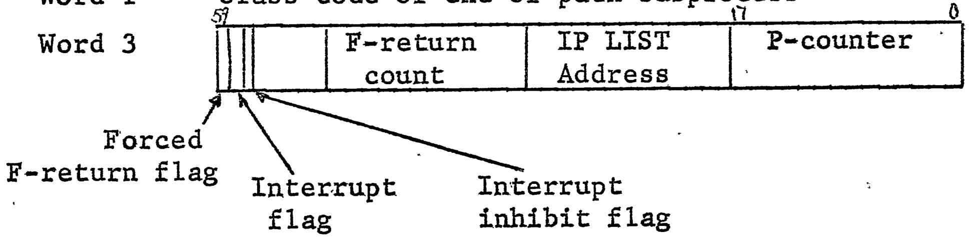
Class	#	Description
2	2	P-counter is negative
2	3	P-counter exceeds user's FL

I. Display Stack

AP1	D: CM address of a buffer area
AP2	D: Size of buffer area (≥ 4)

The user may examine the call stack of a process. He must supply a address of a buffer area and its length so that the system can copy the stack into the specified area. The number of entries in the stack is stored in the first word of the buffer. As many entries as possible starting with the current top of stack are then copied into succeeding 3 word sections of the buffer. The stack entries are reformatted.

Word 0	C. ss code of current subprocess
Word 1	Class code of end of path subprocess
Word 3	



J. Display Stack Entry

AP1	D: CM address of buffer
AP2	D: Desired stack entry

A particular entry in the call stack of a process can be examined if the system is supplied with the CM address of a buffer area (each entry is 3 words long) and the index (relative to the top of the stack) of the desired stack entry. Format same as in I above.

Possible errors while displaying a stack entry:

Class	#	Param	Description
2	2	1	CM address is negative
2	3	1	CM address exceeds user's FL
2	2	2	Stack entry pointer negative
2	3	2	Stack entry pointer exceeds stack

K. Send Process Interrupt

AP1	C: A process (OB.SDINT)
AP2	C: Class code of a subprocess (OB.INTSP)
AP3	D: An 18 bit interrupt datum

The process interrupt is one of the two ways in which a running process may effect the execution of another running process (the other is via an event channel). The process interrupt enables one process to force the calling of a specified subprocess (AP2) (called the interrupt subprocess) within another process (AP1) (called the interrupted process); i.e., the first rpocess forces the interrupted process to call the interrupt subprocess. However, the interrupt is given a "priority" in that the interrupt subprocess will not be called unless (or until) it is an "ancestor" of the "current subprocess"; that is, the subprocess which is actually executing in the interrupted process at the time of the call (or thereafter). Therefore, how soon the interrupt subprocess gets entered depends upon its position in the subprocess tree and the flow of control in the interrupted process. An 18-bit interrupt dauum (AP3) is passed as the parameter, of the call of the interrupt subprocess. Once a subprocess becomes an interrupt subprocess, and until that subprocess is called as an interrupt subprocess, all subsequent interrupts to that subprocess are disabled (have no effect). Since each subprocess is technically its own ancestor, it is necessary when an interrupt subprocess is called to automatically inhibit interrupts for the current (= interrupt) subprocess. When interrupts are inhibited for a subprocess, an interrupt to the subprocess will be remembered but cannot cause the interrupt subprocess call as long as the interrupt inhibit is set and the subprocess in question is the current subprocess.

At every normal subprocess call and return, a check is made for waiting interrupt subprocesses (subprocesses for which a process interrupt has been issued but which have not yet happened to be the ancestor of any current subprocess). If any interrupt subprocesses are waiting, the ancestors of the new current subprocess are checked to see if any of them is an interrupt subprocess. If so, the interrupt subprocess is executed instead; execution begins two words above its entry point.

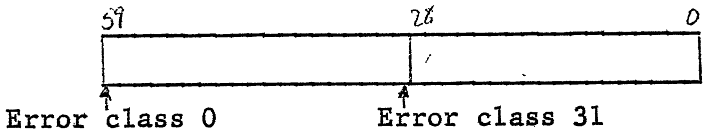
Possible errors while sending a process interrupt:

Class	#	Description
2	1	Interrupt datum exceeds 18 bits

L. Set Local ESM (Error Selection Mask)

AP1 D: Pointer to new ESM

The error selection mask, which determines which classes of errors a subprocess can handle, may be set in the current subprocess by specifying a pointer to the new ESM. The ESM is a bit string (32 bits per word) in which a 1 indicates acceptance of the corresponding error class; i.e.,



Possible errors while setting local ESM;

Class	#	Param	Description
2	2	1	Pointer to ESM < 0 >
2	3	1	Pointer to ESM > FL

M. Set ESM in any subprocess

AP1 D; Pointer to new ESM
AP2 C: Capability for class code subprocess name (OB.STESM)

By specifying the name (class code) of a given subprocess in addition to a pointer to a new ESM, the Error Selection Mask for any given subprocess may be reset.

Possible errors while setting ESM in any subprocess:

Class	#	Param	Description
4	5		Subprocess does not exist
2	2	1	Pointer to ESM < 0
2	3	1	Pointer to ESM > FL

N. Destroy Process

AP1 C: Capability for process to be destroyed (OB.DSTRY)

The system action of destroying a process requires only a parameter giving the C-list index of the process which is to be deleted. The process will be removed from any event channels on which it is waiting and its address space in ECS released.

Possible error while destroying a process

Class	#	Description
5	3	Process does not exist

O. Destroy a Subprocess

AP1 C: Capability for class code of subprocess to be destroyed (OB.DSTRY)

A subprocess can be destroyed if it is currently a leaf of the subprocess tree; otherwise an F-return will be made. If the subprocess is in the call stack, an error is generated.

Possible errors while destroying a subprocess

Class	#	Description
4	5	Subprocess does not exist
4	11	Attempt to delete subprocess in stack

P. Save (Restore) Registers

AP1 D: Pointer to 16 word buffer for registers

The exchange jump package for a process can be saved (restored) in (from) the user's area if a pointer to a 16 word buffer is specified. When the registers are restored, only the programmable registers (A, B and X) are restored.

Possible errors while saving (restoring) registers:

Class	#	Description
2	2	Pointer to buffer is negative
2	3	Pointer to buffer is too large (within 16 words of user's FL)

Appendix A

User supplied parameters for system action

File Actions

Create a File

AP1	C:, Capability for an Allocation block (OB.ALFIL)
AP2	D: C-list index to return capability
AP3	D: Number of levels in file
AP4	D: Pointer to list of shape numbers

Create a Block

AP1	C: Capability for file (OB.CREBL)
AP2	D: Address of block in file

Check for missing blocks

AP1	C: Capability for file
AP2	D: Address of block in file

Read Shape of a File

AP1	C: Capability for file
AP2	D: Address of buffer for shape numbers
AP3	D: Buffer size

READ(Write) a File

AP1	C: Capability for file (OB.RDFIL, OB.WFILE)
AP2	D: Address in file
AP3	D: Address in CM
AP4	D: Word count

Mover a block of a file

AP1	C: Capability of a source file (OB.RDFIL)
AP2	D: Address of source block
AP3	C: Capability for destination file (OB.WFILE)
AP4	D: Address of destination block

Delete a Block from a File

AP1	C: Capability for file (OB.DELBK)
AP2	D: Address of block to be deleted

Delete a File

AP1	C: Capability for file (OB.DSTRY)

II MAP ACTIONS

Zero a map entry

AP1	C: Class code (subprocess name) (OB.CHMAP)
AP2	D: Index in logical map of subprocess

Change (create) a map entry (read/write or read only)

AP1	C: Class code of subprocess (OB.CHMAP)
AP2	D: Index of map entry in AP1
AP3	C: Associated file (read only or read/write) (OB.PLMAP, OB.RDFIL, OB.WFILE)
AP4	D: Address in file to be changed
AP5	D: Address in CM of new entry
AP6	D: Word count of new entry

Display map entry in named subprocess

AP1	C: Class code for subprocess
AP2	D: Index of entry in logical map of AP1
AP3	D: Address of 3 word buffer

Display entry in full map

AP1	D: Index of entry in full map
AP2	D: Address of 3 word buffer

III EVENT CHANNEL ACTIONS

Create an event channel

AP1	C: Capability for allocation block (OB.CREEV)
AP2	D: C-list index for new event channel capability
AP3	D: Length of event queue

Send an event (with or without duplicate checking)

AP1	C: Capability for event channel (OB.SNDEV, OB.SDEVX)
AP2	D: Datum part of event

Get an event or hang

AP1	C: Capability for event channel (OB.GETEV)

Get an event or F-return

AP1	C: Capability for event channel (OB.GTEVF)

Get an event from one of a list of event channels or hang (F-return)

AP1:	D: pointer to list of event channel C-list indices (OB.SETEV... (OB.GTEVF...
AP2	D: Number of channels in list

Destrony an event channel

AP1	C: Capability for event channel (OB.DSTRY)

IV C-LIST ACTIONS

Create a C-list

AP1	C; Capability for allocation block (OB.CRECL)
AP2	D: Index in full C-list to return new capability
AP3	D: Length of new C-list

Display a capability from full C-list

AP1	D: Index in full C-list

Copy a capability within full C-list and decrease options

AP1	D: Index of desired capability
AP2	D: Index of destination C-list entry
AP3	D: Mask if options to preserve

Copy capability from full C-list to arbitrary C-list (vice-versa)

AP1	C: Index of destination (source) C-list (OB.CPYIN,OB.CPYOT)
AP2	D: Index within destinaion (source) C-list of capability
AP3	D: Index in full C-list of source (destination) capability

Destroy a C-list

AP1	C: Capability for C-list (OB.DSTRY)

V OPERATIONS

Make a subprocess call or subprocess jump operation

AP1	C: Capability for allocation block (OB.ALORD)
AP2	D: C-list index to return new operation
AP3	D: Type (0= call, non-zero=jump)
AP4	C: Class code for subprocess called by new operation (OB.CALOP)
AP5	D: Number of parameters used by the subprocess call

Add an order to an operation

AP1	C: Capability for allocation block (OB.ALORD)
AP2	D: C-list index to return new operation
AP3	C: Capability for existing operation (OB.ADDOR)
AP4	D: Type of order (0=call; non-zero=jump)
AP5	C: Class code of subprocess called by new order (OB.CALOP)
AP6	D: Number of new parameters being added

Copy an operation

AP1	C: Capability for allocation block (OB.ALORD)
AP2	D: Full C-list index for new operation
AP3	C: Operation to copy

Change a parameter specification type from "None" to "user-supplied datum"

AP1	C: Capability for operation (OB.CHTYP)
AP2	D: Index of parameter specification

Change a parameter specification type from "none" to "any capability"

AP1	C: Capability for operation (OB.CHTYP)
AP2	D: Index of parameter specification type to change

Change a parameter specification type f om "none" to "user-supplied capability"

AP1	C: Capability for operation (OB.CHTYP)
AP2	D: Index of parameter specification type
AP3	D: Capability type
AP4	D: Capability option bit mask

Change a parameter specification type from "user-supplied datum" to "fixed-datum"

AP1	C: Capabilit y for operation (OB.CHTYP)
AP2	D: Index of parameter specification type
AP3	D: 60-bit datum word

Change a parameter specification type from "user-supplied capability" to "fixed capability"

AP1	C: Capability for operation (OB.CHTYP)
AP2	D: Index of parameter specification type in operation
AP3	C: A capability

VI PROCESS AND SUBPROCESSES

Create a class code

AP1	C: Capability for class code

Set Temporary part of class code

AP1	C: Capability for class code (OB.TEMP)
AP2	D: C-list index for modified class code
AP3	D: New temporary part (30 bits)

Create a Process

AP1	C: Capability for Allocation block (OB.CREPR)
AP2	D: C-list index for returned process capability
AP3	D: Number of event channel chaining words
AP4	D: Number of stack entries
AP5	C: Class code for initial subprocess (OB.SONSP)
AP6	D: Number of map entries in initial subprocess
AP7	D: Compiled map buffer size for initial subprocess
AP8	D: Subprocess field length

AP9	D: Subprocess entry point
AP10	C: Capability of C-list for subprocess (OB.LOCCL)
AP11	C: Capability of file for 1st map entry (Read/Write: OB.WFILE, OB.RDFIL, OB.PLMAP) for initial subprocess
AP12	D: Address within file
AP13	D: Address in CM
AP14	D: Count of words to be swapped
AP15	D: Capability of file for 2nd map entry (Read Only: OB.RDFIL, OC.PLMAP) for initial subprocess
AP16	D: Address within file
AP17	D: Address in CM
AP18	D: Count of words to be swapped

Create a subprocess

AP1	C: Capability for new subprocess class code (OB.SONSP)
AP2	C: Capability for class code of the "father" of subprocess (OB.FATHR)
AP3	D: Number of map entries
AP4	D: Compiled map buffer size
AP5	D: Subprocess field length
AP6	D: Subprocess entry point
AP7	C: Capability for subprocess local C-list index (OB.LOCCL)

Subprocess call

See Operations

Subprocess return

See Operations

Subprocess F-return

See Operations

Subprocess Jump Return

AP1	C: Capability for class code of subprocess to return to (OB.SPRET)
AP 2	D: Number of stack occurrences of AP1 to skip

Modify P-counter of subprocess

AP1	C: Capability for class code of subprocess (OB.PCNT)
AP2	D: Number of stck occurrences of AP1 to skip
AP3	D: New P-counter

Display stack

AP1	D: CM address of a buffer area
AP2	D: Size of buffer area (≥ 4)

Display stack entry

AP1	D: CM address of buffer
AP2	D: Desired stack entry

<u>Send process interrupt</u>

AP1	C: Capability for a process (OB.SDINT)
AP2	C: Capability for a class code of a subprocess (OB.INTSP)
AP3	D: An 18 bit interrupt datum

<u>Set local ESM (Error Selection Mask)</u>

AP1	D: Pointer to new ESM

<u>Set ESM in any subprocess</u>

AP1	D: Pointer to new ESM
AP2	C: Cap ability for class code

<u>Destroy a process</u>

AP1	C: Capability for process to be destroyed (OB.DSTRY)

<u>Destroy a subprocess</u>

AP1	C: Capability for the class code of subprocess to be destroyed (OB.DSTRY)

<u>Save (restore) registers</u>

AP1	D: Pointer to 16 word buffer for registers

Appendix B

Options

Option	Description	Object	Bit #
OB.ALFIL	Allocate a file	Allocation Block	
OB.CREBL	Create a block	File	
OB.WFILE	Write on a file	File	
OB.RDFIL	Read a file	FIle	
OB.DELBK	Delete a block from a file	File	
OB.DSTRY	Destroy the object	Any object	
OB.CHMAP	Change a map entry	Class Code	
OB.CREEC	Create an event channel	Allocation block	
OB.SNDEV	Send an event (wåthout checking)	Event Channel	
OB.SDEVX	Send an event (with check for duplicate)	Event Channel	
OB.GETEV	Get an event (or hang)	Evetn Channel	
OB.GTEVF	Get an event (or F-return)	Event Channel	
OB.CRECL	Create a C-list	Allocation block	
OB.CPYIN	Copy into a C-list	C-list	
OB.CPYOT	Copy out of a C-list	C-list	
OB.LOCCL	Local C-list		
OB.ALORD	Create an order of an operation	Allocation block	
OB.ADDOR	Add an order	Op ration	
OB.CHTYP	Change parameter specification		
CP.CHTYP	type	Operation	
OB.CREPR	Create a process	Allocation bl ck	
OB.PCNT	Modify P-counter	Subprocess	
OB.CALOP	Create call operation	Subprocess	
OB.SDINT	Send process interrupt	Process	
OB.STESM	Set ESM of subprocess	Subprocess	
ON.INTSP	Interrupt subprocess	Subprocess	
OB.SONSP	Name son subprocess	Subprocess	
OB.FATHR	Name father subprocess	Subprocess	
OB.SPRET	Subprocess to return to	Subprocess	