Mission Description Runoff: 12/08/78

Functional Description of CC

Functional Description

Concurrency control

Concurrency control exists as a function of integrity control to control the sharing of resources between tenants.

Control will be exerted at two logical levels. The coarse level, or resource level, is for the control of individual resources, or groups of resources to be controlled as an individual resource. An example of the first would be the control of the sharing of a file. An example of the second would be the control of a group of files under one database name. The fine level, or subresource level, is for the control of subsets of the coarse level resources. A standard use of this level is the treating of file control intervals as subresources of the file.

Whenever users wish to cooperatively share resources, they must express their intent to concurrency control prior to accessing the resource. This expression of intent is via the "enqueue" command which identifies the particular resource or subresource desired and the type of reservation desired. The type of reservation is a statement of the degree to which this user is willing to share this (sub)resource. The user may require EXCLUSIVE use of the resource, in which case he is unwilling to share the resource with anyone else, or the user may require SHARED use, which means that he is willing to share the resource with any other user requesting SHARED use. For the coarse level of resources, the user may also specify SUBRESOURCE, which means that he wishes the right to request individual subresources of this resource and is willing to share the resource only with other users making requests in the SUBRESOURCE mode.

Enqueue commands establish resource "ownership". If a user requests EXCLUSIVE and there is any current owner, a "conflict" is detected. If a user requests SHARED and the current owner(s) have specified SHARED, or there are no current owners, then the requestor may be placed on the ownership list. Otherwise, a conflict is detected again. If a user requests SUBRESOURCE, conflict will result unless there are no current owners, or all current owners have SUBRESOURCE type reservations. In all cases of conflict, the requestor is marked on a waiting list for that resource, and his process is suspended via the WAIT command. Resource requests are handled FIFO, so prior waiters will also cause new requestors to be delayed. The process will wait until the resource becomes free, a deadlock is detected (see next

paragraph), or until the wait time exceeds a value designated by the tenant in the requesting command.

Some subset of the waiting processes can possibly be in a deadly embrace situation, which could result in none of the involved processes ever being awakened. This can occur in a simple case if process "A" holds resource "x" while requesting resource "y" and simultaneously, process "B" holds resource "y" while requesting resource "x". It can be seen that neither resource "x" nor "y" will become available until either process "A" or "B" is forced to relinquish the resources that they currently hold. This particular situation is commonly called a "deadlock". Deadlocks do not occur frequently in systems observed to date, and they are rather expensive to detect. For these reasons, concurrency control will perform deadlock detection in the background. That is to say, it will guarantee that deadlocks will be eventually detected, but possibly after some set increment of time has elapsed.

The deadlock detection mechanism will name the list of processes which are involved in the deadlock. A distinct procedure will pick the process(es) which will receive the deadlock status. When the deadlock is broken, the other processes will proceed. The process(es) receiving the deadlock status will be required to either abort or rollback to a commitment point and retry.

At each enqueue, the user supplies a phase number from his PCB. This number becomes the phase identification of the reservation for its duration. The phase number in the PCB will be initialized to zero at the start of each commitment_unit and will be incremented by integrity_control as the user establishes intermediate checkpoints to which he wishes to recover. Once a new phase is initiated, no user dequeue command will be able to alter a reservation from a prior phase, and all reservation updates will be to more restrictive reservation types (eg. SHARED to EXCLUSIVE) with the original phase identification of the reservation remaining unaltered. These policies will insure the integrity of the user's intermediate checkpoints.

The first time a user updates a shared resource, he must set the update lock on the associated EXECLUSIVE reservation. This may be done at the same time the EXCLUSIVE reservation is made by setting the update lock bit in the command block. Once this lock is set, only integrity_control is allowed to dequeue the reservation. Integrity_control will use the Dequeue_all command available through a restricted entry point. All user accessible dequeue commands will ignore requests to dequeue update_locked resources.

Ownership of a resource by a tenant will be maintained until the resource is explicitly released by means of a dequeue command. Dequeue commands available to the tenant will allow him to dequeue named resources subject to the phase number and update_lock restrictions described in the preceeding paragraphs. The tenant may also declare a set of current files (resources) and a set of current sub-resources over those files so that all non-update locked sub-resources of the current phase, which belong to the current files but which are not on the current subresource list, are dequeued. This command, Dequeue_non-current, will allow a database manager to declare to concurrency_control a set of "current" subresources. The implicit assumption is that all reservations which are not current and not update_locked are not necessary in order to maintain the consistency of the database. For example, when walking an ordered data set, the control intervals searched which do not contain the prior or current data elements usually do not have to remain reserved once the target data element is found.

Mission Description Runoff: 12/08/78

All changes and references to concurrency control tables will be made by concurrency control procedure through the use of the defined command interface.

The basic data structure is illustrated in FIGURE 1. Each data entry in the structure is described below.

- Process_entry - PE

  one entry for each process holding resources or waiting on a resource on behalf of a tenant; points to lists of reservation and waiting entries; located in a linear table indexed by process number.

- Resource_control_block - RB

  one entry per resource known to concurrency_control; points to lists of owners and waiters on this resource; points to lists of subresources; located by the RB_ID token held by the user.

- Subresource_control_block - SB

  one entry per named subresource in use; points to lists of owners and waiters on this subresource; located as uniquely named subresource of designated resource or from subresource_reservation_entry.

- Resource_reservation_entry - RR

  one entry per ownership of resource; identifies owner and RB; identifies ownership type; located from the RR_ID token held by the user.

- Subresource_reservation_entry - SR

  one entry per ownership of a subresource; identifies the owner and the RB; identifies the ownership type; holds the update lock; located by the SR_ID token held by the user.

o Resource_waiting_entry - RW

one entry per process waiting for a resource; identifies the waiting process and the reservation type; the entry is in a FIFO queue pointed to by the RB.

o Subresource_waiting_entry - SW

one entry per process waiting for a subresource; identifies the waiting process and the reservation type; the entry is in a FIFO queue pointed to by the SB.

These data entries will be manipulated from the command interface through two entry points. The first entry point is available to the buffer manager, the access method, the database manager, and the batch interface:

o Allocate_RB - EP1

declares a resource to concurrency control; physically allocates an RB; returns an RB identifying token to be supplied by the user when referencing this resource in subsequent calls; this command never results in a wait.

o Release_RB - EP1

physically deallocates the RB identified by the RB_ID token; the RB_ID token becomes invalid; this call never results in a wait.

o Enq_resource - EP1

enqueues a process on an RB specified by the RB_ID token returned from the Allocate_RB command, or updates an existing reservation identified by the RB_ID and, optionally, the RR_ID from the prior enqueue; the process also supplies the reservation type, the phase number, and a maximum wait time; concurrency control will immediately return if the resource is available, otherwise, the process will be forced to wait; waits can result in a time-out, a deadlock, or a reservation

completed status; an RR_ID token is returned for further efficient reference to this reservation; when updating a current reservation, a transition from SHARED to EXCLUSIVE or from SUBRESOURCE to EXCLUSIVE will cause a dequeue and an enqueue at the head of any existing queue.

o Enq_subresource - EP1

enqueues a process on an SB which is specified by a unique subresource name (SR_name), the RR_ID token returned by the Enq_resource command, and optionally, the SR_ID returned from a prior enqueue of this sub-resource (for updates); the process should also supply the reservation type, the phase number, the update lock setting, and a maximum wait time; concurrency control will immediately return if the resource is available, otherwise, the process will be forced to wait; a wait can result in a time-out, a deadlock, or a reservation completed status; an SR_ID token is returned for further efficient reference to this reservation; multiple calls for the same resource to update the reservation type and the update lock setting are allowed; when updating an existing reservation, a transition from SHARED to EXCLUSIVE will cause a dequeue and an enqueue at the head of any existing queue.

o Deq_resource - EP1

for all SB's which are children of the RB identified by the supplied RR_ID token, delete all ownership reservations of this process; then delete the ownership reservation of this process on this RB; the RR_ID token becomes invalid as well as all SR_ID tokens of all SB reservations; this command never results in a wait.

o Deq_subresource - EP1

dequeues a process from the SB which is specified by a unique subresource name (SR_name), the RR_ID token returned by the Enq_resource command, and optionally, the SR_ID returned from the prior enqueue of this sub-resource; the SR_ID token becomes invalid; this command never results in a wait.

Mission Description Runoff: 12/08/78

o Update_lock - EP1

the update lock for the specified SR will be set; the command will never result in a wait.

o Deq_non-current - EP1

the user will supply a list of of RB_ID's and a list of SR_ID's; for the given list of RB_ID's, all SR_ID's not on the given SR_ID list and whose update locks are not set, will be deleted; this command will only affect entries with the current phase number; the command will not result in a wait.

The second of these entry points will be available only to integrity_control and the batch interface. It will only need to be used at rollbacks, commits, and process aborts.

o Deq_all - EP2

the process will identify itself and a phase number; all reservations for the process with a phase greater to or equal to the supplied phase number will be deleted; update locks will be overridden; this command will never result in a wait.

Usage Information of CC

Usage Information

Concurrency control

A. General Description

The use of concurrency control entails first making the shared resource known to the function. Each resource must be associated with a resource_control_block.

Each resources's RB will be explicitly created and deleted by command from one process on behalf of all other sharing processes. A token will be returned from concurrency control by the Allocate_RB command which will uniquely identify the RB allocated. All subsequent users referring to that RB will be required to produce that RB_ID token. All sharers of the

resource(s) represented by that RB must therefore have addressability to that token. After a command to release an RB has been received, it will be assumed that no subsequent user will produce that RB_ID token, i.e. it must be destroyed by the user.

Resource reservation commands can be of three mutually exclusive types:

EXCLUSIVE: request for exclusive use of all subresources;

SHARED: request for shared use of all subresources;

SUBRESOURCE: request for right to make subresource reservations.

An EXCLUSIVE enqueue will prevent all other enqueue requests for this resource from being serviced until this process dequeues it. The process will share the resource with no one. A data manager would probably want EXCLUSIVE use of a file extent which was to be written.

A SHARED enqueue will prevent all other EXCLUSIVE and SUBRESOURCE requests from being honored for this resource until this process dequeues it. A process requesting this mode is willing to share the resource, but only with other users who specify SHARED. An example of this mode would be the batch allocation of a file with READ disposition.

A SUBRESOURCE enqueue will prevent all other EXCLUSIVE and SHARED requests for this resource from being honored until this process dequeues it. A process requesting this mode is willing to share this resource with any other process who has specified SUBRESOURCE. This enqueue command gives the process the ability to make EXCLUSIVE and SHARED enqueue requests on subsets of the resource. These subsets must be agreed upon by all users, the members must have unique names, and the members must be disjoint. For a database manager, this would mean the ability to control access at the control interval level. One user may hold control interval 5 for update by himself while a group of users are simultaneously sharing control interval 7 for reading.

A reservation entry token is returned to the tenant by each reservation command. This token should be used by the tenant in all subsequent explicit references to this reservation. By convention, a zero token will be invalid. By using this convention both the user and concurrency control will be able to

tell whether a given entry is a valid token.

Once a tenant has secured permission to make subresource requests, it enqueues on subresources of the parent RB by identifying the parent via the reservation entry token and specifying the subresource by a unique name agreed upon by all sharing tenants. Allowable subresource reservation requests are EXCLUSIVE and SHARED, where these requests have the same meanings as requests for RB's.

Subresource_control_blocks will be physically allocated and released as needed by concurrency control. SR_ID tokens, returned to the tenant at enqueue time, can be used only for subsequent dequeues or update lock handling by that tenant.

A reservation request will be honored immediately if the resource is available. If the requested resource is busy and cannot be shared due to incompatibility between the current reservation and the request, then the requestor will be delayed. Tenants will be delayed by a WAIT on a run-time defined RB_Free condition. The corresponding condition will be met when concurrency control dequeues the last owner of an RB and signals the waiting tenant. The tenant will wait until a deadlock is detected, the RB becomes free, or the wait time exceeds a value designated by the tenant in the requesting command.

Deadlock detection will be done at the discretion of concurrency control to promote system-wide efficiency. When a deadlock is detected, the PCU_sequence numbers of the processes involved will be inspected. The process with the largest PCU_sequence number, i.e. the youngest PCU, will be selected to recieve the deadlock status. This process will have one reservation that will be detected as being involved in the deadlock. The phase number of that reservation will be passed to integrity control. This will inform integrity control how far the process must be rolled back to effectively break the deadlock.

All reservation requests will be serviced FIFO. A pending request for EXCLUSIVE usage will prevent a subsequent request for SHARED usage from being immediately granted even though all current owners have SHARED usage specified. This policy prevents a tenant from being indefinitely blocked while waiting as long as the current owners eventually terminate.

Any reservation may be altered by issuing another enqueue command for the resource. If the reservation type is to be changed, the user must be prepared to wait and must supply a wait time. The phase number obtained from the user's PCB during the original enqueue will not be updated. It is expected that most use of

this function will be to effect a transition from SHARED to EXCLUSIVE on resources and subresources. One exception to the FIFO policy mentioned in the preceding paragraph will be when an owner who has specified SHARED, requests EXCLUSIVE. The first such request which cannot be immediately granted due to the existence of concurrent owners, will be placed waiting at the head of the reservation list with an EXCLUSIVE request. Subsequent EXCLUSIVE requests from concurrent owners will be recognized as a deadlock condition. This exception to the FIFO policy facilitates the standard practice of reading a database entity, then writing it later. Any enqueue with the same reservation type and update lock setting as the current reservation will result in no change to the reservation and an immediate return.

Periodically, for efficiency, a user may wish to release all subresources which he has reserved which have no currencies. A command is provided which allows a user to specify a list of resources and a subset of the subresources of those resources. All subresources of the named resources not listed and whose update locks are not set (see next paragraph) will be released. This will apply only to reservations initially made in the current phase. The last established currencies in prior phases will not be affected in order that a rollback can be supported to a prior phase boundary.

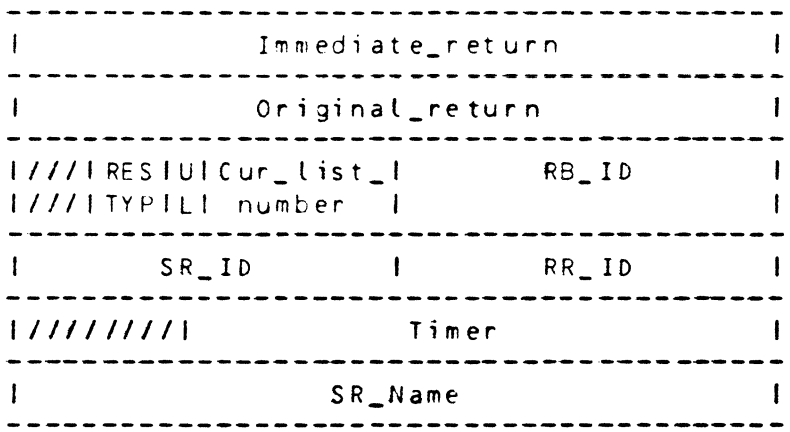
The subresource reservation commands will also support a mechanism referred to as an update lock. Its primary purpose is to give the user the ability to specify a subresource which is needed in order to recover the PCU. The initial setting of the lock is specified in the reservation request word. The lock may be set later by using the Update_lock command or by issuing another enqueue command with the lock bit set. Note that an update lock may never be reset. An update locked resource will not released by the Release_non-current command. Updated file extents should be marked as update locked by the data manager so that they will be available for possible rollback even though they may become non-current.

A command is provided to dequeue all resources of a process. Resources and subresources can also be dequeued individually by identifying them explicitly in the dequeue commands. These commands override update locks, but are honored only if the phase number in the PCB is less than or equal to the phase when the reservation was originally established. Once a phase number is written in a reservation block, subsequent references to that reservation will never cause the phase number to be rewritten.

B. Parameter Stack & Command Block Descriptions

The first descriptor on the parameter stack will define the six word command block described below. The first two words of the block are the return words as defined by system conventions. The lower 12 bits of the first word will contain the status from each command. The parameters in the block will always have the same positions in the block for all commands, but only the variables called out in the command descriptions will be used for a particular command.

The last descriptor on the parameter stack should only be supplied for the Deq_non-current command. It will define a subordinate descriptor segment. This segment contains a variable-size array of descriptors pointing to currency and keep lists.



Command_block

o Cur_list_number

an 18-bit parameter which is an unsigned integer describing the number of currency lists supplied by the user; there is one descriptor for each list, so this identifies the number of descriptors to be used from the Currency_list_d subordinate descriptor segment.

o RB_ID

an 18-bit token identifying a particular RB; the user obtains it on the return from the Allocate_RB command; the user supplies it for the subsequent Enq_resource and Deq_non-current commands referencing this resource; it is up to the user to maintain a mapping of RB's to particular resources.

o RR_ID

an 18-bit token identifying a particular resource_reservation_entry (RR); the user obtains it on the return from the Enq_resource command; the user supplies it for the subsequent Enq_resource_update, Deq_resource, and Enq_subresource commands.

o SR_ID

an 18-bit token identifying a particular subresource_reservation_entry (SR); the user obtains it on the return from the Enq_subresource command; the user supplies it for subsequent Enq_subresource_update, Deq_non-current, Deq_subresource_token, and Update_lock commands; a zero token is invalid.

o Reservation_type

a 2-bit parameter designating the reservation type requested; reservation types are explained above; allowable values are:

0 - Invalid

1 - EXCLUSIVE

2 - SHARED

3 - SUBRESOURCE.

o Update_lock

a 1-bit parameter which is a lock on the reservation; once set, the lock cannot be reset and the reservation will not be dequeued by a Deq_non-current command; the

recognized settings are:

0 - reset

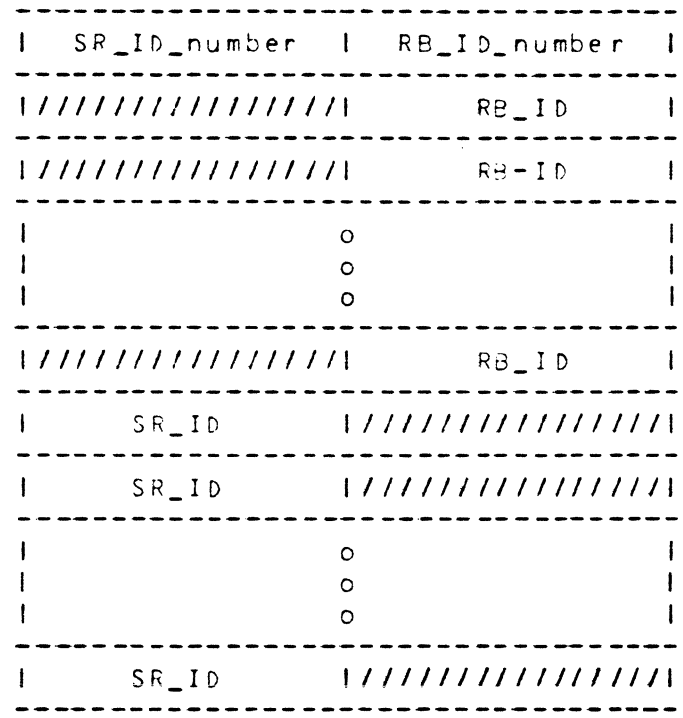
1 - set.

o Timer

a 30-bit parameter specifying the maximum time to wait while performing this request; the units are in milliseconds; a wait time exhausted status will be returned if this amount of time elapses while waiting on an enqueue request; specification of 0 time will result in an immediate return with no wait performed, but a possible status of wait time exhausted.

o SR_name

a 36-bit Subresource_name which uniquely identifies a subresource of a named resource; the user supplies this name on Enq_subresource and Deq_subresource_name commands; a name of zero is valid.



Currency_list

- SR_ID_number

  an 18-bit integer specifying the number of SR_ID's on the end of the list.

- RB_ID_number

  an 18-bit integer specifying the number of RB_ID's on the beginning of the list.

- RB_ID

  an 18-bit RB_ID previously obtained from an Allocate_RB command; this list defines the resources whose reservations will be dequeued.

- SR_ID

an 18-bit SR_ID returned by the Enq_subresource command; this list, along with the update locks, will define which reservations will be kept.

Mission Description Runoff: 12/08/78

C. Command Descriptions

1. Allocate_RB
2. Release_RB
3. Enq_resource
4. Enq_subresource
5. Deq_resource
6. Deq_subresource
7. Update_lock
8. Deq_non-current
9. Deq_all

Mission Description Runoff: 12/08/78

1. Allocate_RB

use:

declares a resource to concurrency control; physically allocates an RB; returns an RB identifying token to be supplied by the user when referencing this resource in subsequent calls; this command never results in a wait.

output:

Status
- 0 - normal return
- 1 - request denied; space is temporarily exhausted

RB_ID

input arguments:

none

2. Release_RB

use:

physically deallocates the RB identified by the RB_ID token; the RB_ID token becomes invalid; this call never results in a wait.

output:

Status
- 0 - normal return
- 4 - request denied; invalid token supplied by user
- 7 - request denied; tenants are enqueued on this RB

input arguments:

RB_ID

3. Enq_resource

use:

enqueues a process on an RB specified by the RR_ID token returned from the Allocate_RB command, or updates an existing reservation identified by the RB_ID and, optionally, the RR_ID from the prior enqueue; the process also supplies the reservation type, the phase number, and a maximum wait time; concurrency control will immediately return if the resource is available, otherwise, the process will be forced to wait; waits can result in a time-out, a deadlock, or a reservation completed status; an RR_ID token is returned for further efficient reference to this reservation; when updating a current reservation, a transition from SHARED to EXCLUSIVE or from SUBRESOURCE to EXCLUSIVE will cause a dequeue and an enqueue at the head of any existing queue.

output:

Status
- 0 - normal return
- 1 - request denied; space is temporarily exhausted
- 2 - request denied; allowing the tenant to wait would result in deadlock
- 3 - request denied; Timer elapsed
- 4 - request denied; invalid token supplied by user
- 5 - request denied; invalid Reservation_type

RR_ID

input arguments:

RB_ID

Reservation_type

Timer

RR_ID (optional)

4. Enq_subresource

use:

enqueues a process on an SB which is specified by a unique subresource name (SR_name), the RR_ID token returned by the Enq_resource command, and optionally, the SR_ID returned from a prior enqueue of this sub-resource (for updates); the process should also supply the reservation type, the phase number, the update lock setting, and a maximum wait time; concurrency control will immediately return if the resource is available, otherwise, the process will be forced to wait; a wait can result in a time-out, a deadlock, or a reservation completed status; an SR_ID token is returned for further efficient reference to this reservation; multiple calls for the same resource to update the reservation type and the update lock setting are allowed; when updating an existing reservation, a transition from SHARED to EXCLUSIVE will cause a dequeue and an enqueue at the head of any existing queue.

output:

Status
- 0 - normal return
- 1 - request denied; space temporarily exhausted
- 2 - request denied; deadlock detected
- 3 - request denied; time-out
- 4 - request denied; invalid token supplied by user
- 5 - request denied; invalid reservation_type

SR_ID

input arguments:

RR_ID

SR_name

Reservation_type

Update_lock

Timer

CLASS: CC

Mission Description

Runoff: 12/08/78

SR_ID (optional)

6. Deq_resource

use:

for all SB's which are children of the RB identified by the supplied RR_ID token, delete all ownership reservations of this process; then delete the ownership reservation of this process on this RB; the RR_ID token becomes invalid as well as all SR_ID tokens of all SB reservations; this command never results in a wait.

output:

Status
- 0 - normal return
- 6 - request denied; the specified resource was not reserved by this tenant

input arguments:

RB_ID

RR_ID (optional)

8. Deq_subresource

use:

dequeues a process from the SB which is specified by a unique subresource name (SR_name), the RR_ID token returned by the Enq_resource command, and optionally, the SR_ID returned from the prior enqueue of this sub-resource; the SR_ID token becomes invalid; this command never results in a wait.

output:

Status
- 0 - normal return
- 4 - request denied; invalid token supplied by user
- 6 - request denied; the specified resource was not reserved by the tenant

input arguments:

RR_ID

SR_name

SR_ID (optional)

7. Update_lock

use:

the update lock for the specified SR will be set; the command will never result in a wait.

output:

Status
- 0 - normal return
- 4 - request denied; invalid token supplied by user

input arguments:

SR_ID

8. Deq_non-current

use:

the user will supply a list of of RB_ID's and a list of SR_ID's; for the given list of RB_ID's, all SR_ID's not on the given SR_ID list and whose update locks are not set, will be deleted; this command will only affect entries with the current phase number; the command will not result in a wait.

output:

Status
- 0 - normal return
- 4 - request denied; invalid token supplied by user
- 8 - request denied; invalid descriptor structure in command blocks

input arguments:

Cur_list_number

Currency_list
- SR_ID_number
- RB_ID_number
- RB_ID (RB_ID_number)
- SR_ID (SR_ID_number)

9. Deq_all

use:

the process will identify itself and a phase number; all reservations for the process with a phase greater to or equal to the supplied phase number will be deleted; update locks will be overridden; this command will never result in a wait.

output:

Status
0 - normal return

input arguments:

none