Software Design Specification – DMS ClientNode

Sub-components:

User application:

mclient.c

ipcsocket.c

User level attach

attach

Query Vol info

Query result

Return Ret

NOK

(Ret = att fail)

ioctl attach

OK

ret

Send detach

Driver not working (att fail)

Ret =Vol max volumes

Ret =Attach fail

Ret = Vol attaching

Ret = Vol exist

Ret =Vol exist detaching

Wait dev file

OK

Wait timeout

OK

OK

(Ret = att OK)

Ioctl attach

finish

ret

Do force detach

Driver not working (att fail)

Timeout

Ioctl attach

finish

ret

OK

(Ret = att Fail)

Driver not working (att fail)

Send Detach

Detach ok

Driver not working (det fail)

Detach Vol Not exist

Detach vol exist attaching

Detach vol exist detaching

Detach vol exist busy

Detach fail

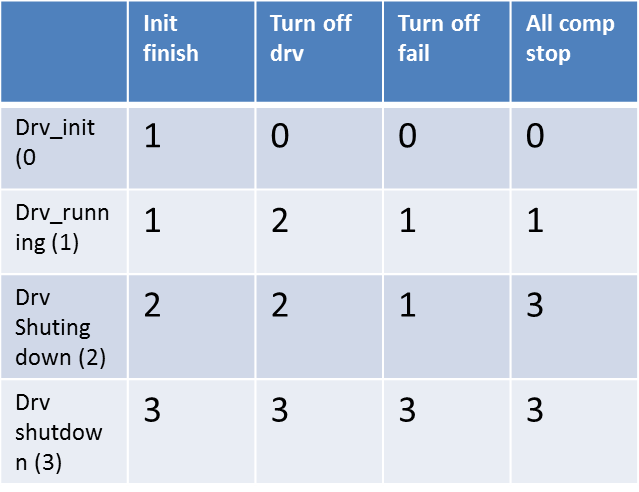
Return Ret

ret

ioctl detach

detach

User level detach

ccmablkmain.c

Drv off fail

Drv\_init

All component  
stop

Turn off drv

Init finish

init module

Drv shutdown

Drv shuting  
down

Drv\_running

volume\_manager.c

volume state

UnBlock IO

detach volume

detach volume

detach volume

Block IO

UnBlock IO

Timeout

Or

no ongoing attch

Other

Vol attach

Vol\_

block io detach

Vol IO  
block yield

Vol\_

attaching

Wait timeout

Or no ongoing attch

Other

Vol attach

vol\_IO  
yield

All IOs done?

detach volume

UnBlock IO

Block IO

attach finish

attach volume

Vol

detacted

Vol\_

detaching

vol\_IO  
blocking

Vol\_ready  
for user

Volume state transition

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Attach finish** | **Block IO** | **Unblock IO** | **Yield IO** | **No Yield IO** | **Yield IO timeout** | **detach** | **No ligner IO** |
| Vol attaching (0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ready for user (1) | 1 | 2 | 1 | 3 | 1 | 1 | 6 | 1 |
| IO blocking (2) | 2 | 2 | 1 | 4 | 2 | 2 | 5 | 2 |
| IO yield (3) | 3 | 4 | 3 | 3 | 1 | 1 | 6 | 3 |
| IO block Yield (4) | 4 | 4 | 3 | 4 | 2 | 2 | 5 | 4 |
| IO block detach (5) | 5 | 5 | 6 | 5 | 5 | 5 | 5 | 5 |
| Vol detaching (6) | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 |
| Vol detached (7) | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |

dms\_client\_mm.c

Descriptions:

DMS ClientNode memory management component try to manage memory resource of DMS ClientNode. Normally, a linux kernel driver use kmalloc or other similar memory malloc fuction to get memory buffer from system. DMS ClientNode module also uses kmalloc to get memory. But for our own consideration, all sub-components of DMS ClientNode don’t call kmalloc directly. We design a DMS ClientNode memory management component which provides a general interface for other sub-components. Other components call the API to get memory.

The reasons for design DMS ClientNode memory management module as following:

1. DMS ClientNode kernel module using lots of data structures when IO is on-going. Normally, these data structure is small and call kmalloc frequently will degrade the IO performance. Pre-allocate necessary memory for these data structures will prevent call kmalloc frequently.
2. Monitor the memory resources used by DMS ClientNode module. With a generic API, DMS ClientNode knows how many API be called and traces it. The information will show at procfs components and DMS ClientNode will know whether memory leak appears when remove kernel module.
3. With a generic memory allocation API, DMS ClientNode can perform resource control when memory resource is nervous by checking the counting information of each kind of memory resource types. DMS ClientNode module provides a wait queue for user thread sleep on this wait queue. When the memory resource is not nervous, memory management module will wake up threads which sleep on it. Currently, we will focus on io request only.

Exported Interface:

ccma\_malloc: Other module can use this API to allocate memory which not in memory buffer. It is a wapper of kmalloc. The major function of this API is using kmalloc to give a memory to caller and increase the counting of API called.

ccma\_free: When caller using ccma\_malloc to get a memory, it should call ccma\_free to return this memory.

ccma\_malloc\_pool: Other module can use this API to allocate memory which already pre-allocate in memory buffer. Currently, there are 9 types of data structures. (io requests, copy payloads, nn requests, fdn requests, located information, dn requests, dn rx parameter list, dn rx packets, dn rx read buffer).

ccma\_free\_pool: When other module uses ccma\_malloc\_pool to get a memory, it should call ccma\_free\_pool to return memory.

ccma\_malloc\_dofc: This is a specific function which will check whether memory resource is nervous. We define nervous through a congestion control variable, when number of memory allocated out large than congestion control variable, DMS ClientNode will return NULL to caller. Caller using this API when it expected to perform a resource control. Currently, this API only works when the resource type is io request.

client\_mm\_init: This API will initialize 9 memory pools and pre-allocate memory chunk for these 9 memory pools. Current, memory pools contain io requests, copy payloads, nn requests, fdn requests, located information, dn requests, dn rx parameter list, dn rx packets, dn rx read buffer.

client\_mm\_release: This API will release every memory chunk in 9 memory pools.

sleep\_to\_wait\_mem: When caller using ccma\_malloc\_dofc and get a NULL memory, it can choose to sleep on wait queue.

wake\_up\_mem\_resource\_wq: When caller using ccma\_free\_pool to release an io requests, DMS ClientNode memory management will checking whether memory is nervous. If no, wake up threads that call sleep\_to\_wait\_mem.

set\_mem\_cong\_control: Other components can set the memory resource congestion counting. An initial value will be 7/8 of io request pool sizes.

get\_mem\_usage\_cnt: How many items of memory pool used by other components.

get\_mem\_item\_size: Return memory chunk size of memory pool.

get\_mem\_pool\_size: Return the number of memory chunks of a memory pool.

Data Structure

A memory pool contains:

uint32\_t pool\_size: How many memory chunks in this pool

atomic\_t num\_items\_using: Counting information about how many chunks used by other components.

struct list\_head free\_list: a link list which link all memory chunks that can be used by other components.

struct list\_head using\_list: a link list which link all memory chunks that already used by other components.

size\_t item\_size: the size of memory chunk in list.

bool is\_mutex\_lock: a bool to indicate the type of lock is mutex or spin\_lock

pool\_lock: a lock to protect link list of free\_list and using\_list. It is a union of mutex and spin\_lock.

A memory chunk contains:

A memory chunk is a memory with continuous space. It can be divided into 3 pieces:

First piece: an unsigned char indicated whether this memory comes from system or comes from memory buffer.

Second piece: a struct list\_head which provide pointer for free\_list or using\_list in memory buffer.

Third piece: User put their data in this space.

Design algorithm:

Pre-allocated Memory: allocates a fix number of memory chunk through kmalloc, and adds these memory chunks to free\_list.

Allocate memory:

Step 1: If there are still have memory chunks in free list, remove it from free\_list and add to using\_list. If no, goes step 2.

Step 2: Using kmalloc to get a memory from system, mark the first piece of memory chunk as as 0x00 which indicate this memory come from system. Return it to caller.

Interaction with other components:

ccmablkmain.c: call function

client\_mm\_init

client\_mm\_release

set\_mm\_cong\_control

procfs: call function

get\_mem\_usage\_cnt

get\_mem\_item\_size

get\_mem\_pool\_size

ccma\_malloc\_dofc

wake\_up\_mem\_resource\_wq

sleep\_to\_wait\_mem

All components:

ccma\_malloc

ccma\_malloc\_pool

ccma\_free

ccma\_free\_pool

MISC:

Currently, we only provide interface for user setting io request pool size. This is because we will control the total amounts of io requests. The max size of other memory pool can be derived from the size of io request pool size. On current design, a io request will contains 17 LB (MAX\_NUM\_OF\_LB\_PER\_REQUEST) at most, so the size of other memory pool be as following:

COPYPAYLOAD\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*2; (For safety, we give twice of io request pool size)

NNREQ\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST;

FDNREQ\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST;

LOCATEDREQ\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST;

WPARAM\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST;

PACKET\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST;

READ\_BUFFER\_POOL\_SIZE = IORREQ\_POOL\_SIZE\*2; (For safety, we give twice of io request pool size)

DNREQ\_POOL\_SIZE =

(IORREQ\_POOL\_SIZE\*MAX\_NUM\_OF\_LB\_PER\_REQUEST)\*

MAX\_DNLOCATION\_PER\_HBID;

(MAX\_DNLOCATION\_PER\_HBID is 3, means that each LB has 3 data copies)