Detailed Level Design for Lustre Parallel CIFS Driver

Matt Wu

2005/02/16

1 Context per cluster (global context)

1.1 Functional Specification

1.1.1 Overview

```
VOLUME_CONTEXT contains the key configuration informaton of a cluster, such as:
1, Lustre server configuration
  a) mds/ost servers alias names (ip addr, host / samba netbios names)
  b) content of ost map (ost servers' information: ost index id ...)
  c) Samba share point name of all the servers
2, Runtime management definitions
  a) serialization lock to protect possible races on structure access
  b) mds root directory change monitor thread to detect the ost changes

The LanmanRedirector is different to local file systems. All the remote servers are tre
We could not get the cluster configuration at the time of mounting. But we could create
VOLUME_CONTEXT can be directly allocated from NonPaged Pool, and initialized in DoMount
The CLUSTER_CONTEXT are to be allocated from a LookAsideList, in PostDoCreate callback
When the last reference of opened file is released, the CLUSTER_CONTEXT will be destroy
DoDismount is called until system is to be shutdown. Then all the  CLUSTER_CONTEXT chai
```

1.1.2 Routines

```
1, VOLUME_CONTEXT allocation/initialization
   PVOLUME_CONTEXT SfCreateVcb(VOID);
2, VOLUME_CONTEXT destruction: cleanup all the resource
   VOID SfDestroyVcb(IN PVOLUME_CONTEXT);
3, CLUSTER_CONTEXT allocateion/initialization
   PCLUSTER_CONTEXT SfCreateCcb(IN PVOLUME_CONTEXT, IN UNICODE_STRING);
4, CLUSTER_CONTEXT destruction
   VOID SfDesctroyCcb(IN PVOLUME_CONTEXT, IN PCLUSTER_CONTEXT);
5, Insert/Remove the CLUSTER_CONTEXT to/from VOLUME_CONTEXT list
```

```
   BOOLEAN SfInsertCcb(PVOLUME_CONTEXT, PCLUSTER_CONTEXT);
   VOID SfRemoveCcb(PVOLUME_CONTEXT, PCLUSTER_CONTEXT);
   PCLUSTER_CONTEXT SfSearchCcb(PVOLUME_CONTEXT, UNICODE_STRING);
6, Refer/derefer the CLUSTER_CONTEXT
   VOID SfReferCcb(PCLUSTER_CONTEXT);
   VOID SfDerefCcb(PCLUSTER_CONTEXT);
```

1.2 Use Case

Volume context creation:

```
BOOLEAN DoMount(PDEVICE_OBJECT DeviceObject
#ifdef NEW_FDDK_INTERFACE
                      ,PPRIVATE_CONTEXT PPrivateContext
                      ,PVOID* PVolumeContext
#endif //NEW_FDDK_INTERFACE
                      )
{
    PVOLUME_CONTEXT Vcb = NULL;
    ......
#ifdef NEW_FDDK_INTERFACE
    Vcb = SfCreateVcb();
    if(!Vcb) {
        ExFreePool(devName);
        return FALSE;
    }
    *PVolumeContext = Vcb;
    KdPrint(("GUIMON mount callback for FS %x, DrvLtr %C\n",PPrivateContext->FsType,PPr
#endif // NEW_FDDK_INTERFACE
    ......
}
```

Volume context destruction:

```
BOOLEAN DoDismount(PDEVICE_OBJECT PDeviceObject
#ifdef NEW_FDDK_INTERFACE
                          ,PPRIVATE_CONTEXT PPrivateContext
#endif NEW_FDDK_INTERFACE
                         )
{
#ifdef NEW_FDDK_INTERFACE
    PVOLUME_CONTEXT Vcb = NULL;
#endif NEW_FDDK_INTERFACE
    DbgPrint("DoDismount callback\n");
#ifdef NEW_FDDK_INTERFACE
    if (PPrivateContext->VolumeContext) {
        Vcb = (PVOLUME_CONTEXT)(PPrivateContext->VolumeContext);
```

```
        if (Vcb->Magic != VOLUME_CONTEXT_MAGIC) {
            DbgBreak();
        }
        SfDestroyVcb(Vcb);
    } else {
        DbgBreak();
    }
#endif NEW_FDDK_INTERFACE
    return TRUE;
}
```

Cluster context creation:

```
{
    PCLUSTER_CONTEXT Ccb = NULL;
    /* allocate new ClusterContext and do initialization */
    Ccb = SfCreateCcb(Vcb, MdsRoot);
    if (!Ccb) {
        DbgPrintf(DBG_ERROR, ("Failed to allocate cluster context ...\n");
        return status;
    }
    /* Attach the ClusterContext to VolumeContext list */
    if (!SfInsertCcb(Vcb, Ccb)) {
       DbgPrintf(DBG_ERROR, ("Failed to attach the cluster context.\n"));
    }
    /* Add a reference count of the ClusterContext */
    SfReferCcb(Ccb);
}
```

Cluster context destruction:

```
SfDeferCcb will directly call SfDestroyCcb to free the cluster context when the referen
```

1.3 Logic Specification

1.3.1 Sturcture definition

```
/*
 *  Volume Context
 *
 *  This structure manages the gobal data for a lustre cluster.
 */
#define VOLUME_CONTEXT_MAGIC 'CLOV'
typedef struct _VOLUME_CONTEXT {
    /* Magic Number */
    ULONG       Magic;
```

```
    /* Flags */
    ULONG       Flags;
    /* Serialize ERESOURCE Lock */
    ERESOURCE               Lock;
    /* Cluster Context Lookaside List */
    NPAGED_LOOKASIDE_LIST   CcLAList;
    /* Stream Context Lookaside List */
    NPAGED_LOOKASIDE_LIST   ScLAList;
    /* Number of Cluster Contextes, i.e. clusters*/
    ULONG                   NumOfCc;

    /* List of Cluster Contextes */
    LIST_ENTRY              CcList;
} VOLUME_CONTEXT, *PVOLUME_CONTEXT;
/*
 *  Cluster Context
 *
 *  This structure manages the configuration data of a cluster.
 */
#define CLUSTER_CONTEXT_MAGIC 'CULC'
typedef struct _CLUSTER_CONTEXT {
    /* Magic Number */
    ULONG       Magic;
    /* Flags */
    ULONG       Flags;
   //
    // parallel I/O cluster contextes
    //
    /* Serialize ERESOURCE Lock */
    ERESOURCE               Lock;
    /* Reference count to control the lifecycle */
    ULONG                   Refer;
    /* full name of mds share point, which is to identify the cluster */
    UNICODE_STRING          MdsRoot;
   /* total servers number */
    ULONG                   NumOfSrvs;

    /* server alias names array, including mds+ost servers. The item
       item could be samba netbios share name, ip address, host name */
    UNICODE_STRING          SrvNames[][];
    /* full name of ost share points */
    UNICODE_STRING          OstRoot[];
    /* .SRV_MAP management structure */
    struct {
       /* name for .SRV_MAP */
       UNICODE_STRING       Name;
```

```
        /* Ost Configuration Info */
        struct ost_map *     Map;
    } SrvMap;
    /* Mds root directory change monitor thread */
    struct {
        /* Handle of opened mds root */
        HANDLE               Handle;
        /* Flag to control the change notify monitor thread */
        BOOLEAN              bShut;
    } Notify;
    /* backtrace pointer to the VolumeContext */
    PVOLUME_CONTEXT         Vcb;
    /* List Entry Link to be attached into VolumeContext */
    LIST_ENTRY              Link;
} CLUSTER_CONTEXT, *PCLUSTER_CONTEXT;
/*
 *  Example of the MDS share point name and Srvmap file path
 */
/* root directory of lustre share point*/
#define SF_LUSTRE_SHARE_POINT L"\\Device\\LanmanRedirector\\mds\\lustre"

/* ost configuration file, under root directory */
#define SF_OST_MAP_NAME L"\\Device\\LanmanRedirector\\mds\\lustre\\.SRV_MAP"
#define SF_OST_MAP_LENGTH  (sizeof(SF_OST_MAP_NAME) - 2)
/*
 *  Data Format of Configuration File: .SRV_MAP
 */
/* the data format of the SRV_MAP content */
struct ost_map {
    /* magic and flags */
    __u16  magic;
    __u16  flags;
    /* number of ost devices */
    __u32  osts_count;
    /* array of ost information */
    struct ost_info osts_info[1];
};
#define OST_MAP_MAGIC  'OM'
/* information per ost*/
struct ost_info {
    /* Magic and flags */
    __u16  magic;
    __u16  flags;
    /* ost index number */
    __u32  ost_index;
    /* ost uuid string, trailing with NUL */
```

```
    const  char uuid[40];
    /* ip address of the ost server */
    __u32  ipaddress;
};
#define OST_INFO_MAGIC 'OI'
```

1.3.2 VolumeContext Routines

```
/*
 * SfCreateVcb
 *     The routine is to allocate VOLUME_CONTEXT and initialize it's
 *     content. The volume context is to be allocated from system pool.
 *
 * Arguments:
 *     N/A
 *
 * Return Value:
 *     PVOLUME_CONTEXT: the pointer to Vcb or NULL in case of failure
 *
 * Notes:
 *     N/A
 */
PVOLUME_CONTEXT
SfCreateVcb(VOID)
{
    1, allocate volume context from NonPaged system pool
    2, initialize the necessary members, such as Magic, Lock
    3, initialize LookAsideList pools
}
/*
 * SfDestroyVcb
 *     The routine is to release VOLUME_CONTEXT structure
 *
 * Arguments:
 *     Vcb: The volume context to be freed.
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfDestroyVcb(PVOLUME_CONTEXT Vcb)
{
    1, call SfDestroyCcb to free the ClusterContext structures attached to the list
```

```
    2, release the resources of the Lock, LookAsideList, etc
    3, free the memory of the VolumeContext to system
}
```

1.3.3 ClusterContext Routines

```
/*
 * SfCreateCcb
 *     The routine is to allocate CLUSTER_CONTEXT and initialize it's
 *     content. It's to be allocated from Vcb->CcLAList.
 *
 * Arguments:
 *     Vcb: the volume context
 *     MdsRoot: the path name of the mds share point
 *
 * Return Value:
 *     PCLUSTER_CONTEXT: the pointer to Ccb or NULL in case of failure
 *
 * Notes:
 *     N/A
 */
PCLUSTER_CONTEXT
SfCreateCcb(
        IN PVOLUME_CONTEXT Vcb,
        IN UNICODE_STRING MdsRoot
        )
{
    1, call SfSearchCcb to get check the cluster context is in the list or not. If atta
    2, allocate cluster context from LookAsideList: Vcb->CcLaList
    3, initialize the necessary members, such as Magic, Lock
    4, start the MdsRoot change notification monitor thread
    5, query and construct the cluster server configuration
}
/*
 * SfDestroyCcb
 *     The routine is to release a CLUSTER_CONTEXT.
 *
 * Arguments:
 *     Vcb: the volume context
 *     Ccb: the cluster context to be released
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
```

```
 */
VOID
SfDesctroyCcb(
      IN PVOLUME_CONTEXT Vcb,
      IN PCLUSTER_CONTEXT Ccb
)
{
    1, stop the directory change monitor thread
    2, release and free all the resources
}
/*
 * SfInsertCcb
 *     The routine is to attach a Ccb to the Vcb global list.
 *
 * Arguments:
 *     Vcb: the volume context
 *     Ccb: the cluster context to be attached
 *
 * Return Value:
 *     TRUE: successful; FALSE: failed to insert
 *
 * Notes:
 *     N/A
 */
BOOLEAN
SfInsertCcb(
      IN PVOLUME_CONTEXT Vcb,
      IN PCLUSTER_CONTEXT Ccb
      )
{
    1, acquire the lock protect of VolumeContext
    2, attach the cluster context to Vcb->CcList
    3, increase the total number of cluster context attached
    4, release the serialization lock
}
/*
 * SfSearchCcb
 *     The routine is to search a special Ccb with it's name
 *     in the Vcb global list.
 *
 * Arguments:
 *     Vcb: the volume context
 *     MdsRoot: the path name of Mds share point
 *
 * Return Value:
 *     PCLUSTER_CONTEXT: return the result of the cluster
```

```
 *     context point or NULL if there's no.
 *
 * Notes:
 *     N/A
 */
PCLUSTER_CONTEXT
SfSearchCcb(
        IN PVOLUME_CONTEXT Vcb,
        IN PUNICODE_STRING MdsRoot
        )
{
    1, search the Vcb->CcList to check whether the cluster context keyed with the MdsRo
    2, return the cluster context if found, or NULL in case not in the list.
}
/*
 * SfRemoveCcb
 *     The routine is to remove the given Ccb from the Vcb global list.
 *
 * Arguments:
 *     Vcb: the volume context
 *     Ccb: the cluster context to be removed
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfRemoveCcb(
      IN PVOLUME_CONTEXT Vcb,
      IN PCLUSTER_CONTEXT Ccb
      )
{
    1, acquire the lock protect of volume context (Vcb)
    2, remove the cluster context from Vcb->CcList
    3, decrease the total number of cluster context attached
    4, release the serialization lock
}
/*
 * SfReferCcb
 *     The routine is to grab a reference of the cluster context.
 *     Normally called by SfCreateScb when creating new streams.
 *
 * Arguments:
 *     Ccb: the cluster context being referred
```

```
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfReferCcb(IN PCLUSTER_CONTEXT Ccb)
{
    call InterlockedIncrement to increase the refer count by 1
}
/*
 * SfDerefCcb
 *     The routine is to release the reference of the cluster.
 *     context. Called by SfDestroyScb when releasing streams.
 *
 * Arguments:
 *     Ccb: the cluster context being referred
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfDerefCcb(IN PCLUSTER_CONTEXT Ccb)
{
    1, call InerlockedDecrement to decrease by 1
    2, in case of the refercount becomes ZERO, call SfDestroyCcb(Ccb->Vcb, Ccb) to free
}
```

1.4 State Management

VOLUME_CONTEXT lifecycle:

```
1, creation in DoMount
2, possible parallel operations on the structure
    2.1 refence by SfCreateCcb and SfDestroyCcb
    2.2 update via SfInsertCcb and SfRemoveCcb
3, destruction in DoDismount
```

CLUSTER_CONTEXT lifecycle:

```
1, creation in PostDoCreate, when the MdsRoot is matched
2, possible parallel operations:
```

2.1 another PostDoCreate trys to update the context
2.2 reference in IRP_MJ_READ / IRP_MJ_WRITE
2.3 update via the directory change monitor thread

3, destroyed when the last reference was dropped when STREAM_CONTEXT being released.

2 Context per stream

2.1 Functional Specification

2.1.1 Overview

The STREAM_CONTEXT structure stands for an opened file. It contains:
1, the user's security context information
2, full path name of the file
3, the lov stripe distribution layout
This structure will be allocated and released very frequently, for we need do the proce
The destruction of the STREAM_CONTEXT is controlled by Osr filter. When the last refere
The osr fddk provies us two routines to register and query the STREAM_CONTEXT.
OsrFilterSetPerStreamContext:
this routine allows the user to create a NonPaged Pool Stream Context that is unique on
OsrFilterGetPerStreamContext:
the routine allows the user to retrieve any Stream Context that has previously been set
Osr fddk also provides the support of context per file object. That means every open in
On the query of the long path name of an opened file, Osr fddk already provides some ro
But the long file names that fddk provides are in different formort of our usage, and t
Some of job is already done in the previous parallel CIFS demo project. This part is to

2.1.2 Routines

```
/* create a new stream context */
PSTREAM_CONTEXT
SfCreateScb(
      IN PDEVICE_OBJECT DeviceObject,
      IN PIRP,
      IN PVOLUME_CONTEXT
      );
/* destroy the stream context */
VOID SfDestroyScb(IN PVOID pScb);
```

2.2 Use Case

NTSTATUS PostDoCreate(PDEVICE_OBJECT DeviceObject, PIRP Irp
#ifdef NEW_FDDK_INTERFACE

```
                              ,PPRIVATE_CONTEXT PPrivateContext
#endif // NEW_FDDK_INTERFACE
                              )
{
    PSTREAM_CONTEXT    Scb = NULL;
    ......
    /* only execute our operations when the original IRP_MJ_CREATE was successfully com
    if (!NT_SUCCESS(Irp->IoStatus.Stauts)) {
        DbgPrintf(DBG_INFO, "the original Irp was completed but with failure return cod
        return (STATUS_SUCCESS);
    }
     /* Query STREAM_CONTEXT and get whether we already set it before */
#ifdef NEW_FDDK_INTERFACE
        Scb = (PSTREAM_CONTEXT) PPrivateContext->StreamContext;
#else // NEW_FDDK_INTERFACE
#ifdef FDDK_STREAM_CONTEXT
        Scb = (PSTREAM_CONTEXT) OsrFilterGetPerStreamContext(currentIrpStackLocation->F
#else //FDDK_STREAM_CONTEXT
        if(PPrivateContext->StreamContext) {
            Scb = OsrFilterLookupPerStreamContext(
                      currentIrpStackLocation->FileObject,
                      VolumeContext,
                      (PVOID) VolumeContext->InstanceNumber.QuadPart);
        }
#endif //FDDK_STREAM_CONTEXT
#endif // NEW_FDDK_INTERFACE
     /* We need allocate new STREAM_CONTEXT in case there's none. */
     if (!Scb) {
        Scb = SfCreateScb(DeviceObject, Irp, Vcb);
        if (!Scb) {
             /* failure case .... */
        }

        /* register the STREAM_CONTEXT to Osr Filter */
#ifdef FDDK_STREAM_CONTEXT
        status = OsrFilterSetPerStreamContext(
                             currentIrpStackLocation->FileObject,
                             Scb,SfDestroyScb);
#else //FDDK_STREAM_CONTEXT
         OsrFilterInitPerStreamContext(
                             (POSR_PER_STREAM_CONTEXT) Scb,
                             VolumeContext,
                             SfDestroyScbb );
          status = OsrFilterInsertPerStreamContext(
                             currentIrpStackLocation->FileObject,
                            (POSR_PER_STREAM_CONTEXT) Scb );
```

```
#endif //FDDK_STREAM_CONTEXT
     }
    ......
}
```

2.3 Logic Specification

2.3.1 structure definition

```
/*
 *  Stream Context
 *
 *  The structure contains the stream information for every file.
 */
typedef struct _STREAM_CONTEXT {
    /* Magic Number */
    ULONG       Magic;
    /* Flags */
    ULONG       Flags;
    /* Serialize ERESOURCE Lock */
    ERESOURCE                   Lock;
    /* Backtrace to it's Cluster Context */
    PCLUSTER_CONTEXT            Ccb;
    /* the real Fcb of the stream */
    PFSRTL_ADVANCED_FCB_HEADER  Fcb;
    /* Security context of the operator, to be impersonated later
       in IRP_MJ_READ and IRP_MJ_WRITE dispatch routines */
    PSECURITY_CLIENT_CONTEXT    SCC;
    /* Full name in unicode, such as \\Device\\LanmanRedirector\\mds\\.SRV_MAP */
    UNICODE_STRING              Name;
    /* lov_mds_md: stripe distribution information */
    struct lov_mds_md *         lmm;
} STREAM_CONTEXT, *PSTREAM_CONTEXT;
/*
 * EA Name for File Stripe Distribution Information
 */
/* name of EA lov_dist, could also be "trusted.lov" */
#define XATTR_NAME_LOV_DIST "LOV_DIST"
/* length of lov_dist, including the trilling "\\0" */
#define XATTR_LEN_LOV_DIST sizeof(XATTR_NAME_LOV_DIST)
```

2.3.2 StreamContext routines

```
/*
 * SfCreateScb
 *     The routine is to allocate STREAM_CONTEXT and initialize it's
```

```
 *     content, including lmm update, security context
 *
 * Arguments:
 *     DeviceObject: the volume device object
 *     Irp: the Irp structure
 *     Vcb: the volume context
 *
 * Return Value:
 *     PSTREAM_CONTEXT: the pointer to Scb or NULL in case of failure
 *
 * Notes:
 *     N/A
 */
PSTREAM_CONTEXT
SfCreateScb(
    IN PDEVICE_OBJECT  DeviceObject,
    IN PIRP Irp,
    IN PVOLUME_CONTEXT Vcb
    )
{
    1, check if the stream context is already created for the file
    2, if created already, then get it, goto step 7; otherwise step 3
    3, allocate new STREAM_CONTEXT from Vcb->ScLaList
    4, initialize the members, create the user security context
    5, get the CLUSTER_CONTEXT and insert the Scb to the Ccb
    6, qeruy the long file name of the opened file via QueryFileName
    7, query the lov_mds_md and update the attributes
}
/*
 * SfDestroyScb
 *     The routine is the callback of osr stream context management.
 *     It's called by the osr filter when the Scb is no longer used.
 *
 * Arguments:
 *     pScb: the address of the STREAM_CONTEXT to be freed
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfDestroyScb(IN PVOID pScb)
{
    1, free the long file name, lmm, SCC ...
```

```
    2, drop the reference of it's CLUSTER_CONTEXT
    3, destroy all the members and free the memory...
}
```

2.4 State Machine

STRAM_CONTEXT lifecycle:

```
1, creation in PostDoCreat
2, possible parallel operations on the sturcture

    2.1 lov_mds_md update in PostDoCreate, completion of IRP_MJ_CREATE
    2.2 name update in IRP_MJ_SETINFORMATION
    2.3 reference in IRP_MJ_READ/IRP_MJ_WRITE

3, destruction by SfDestroyScb callback, called by OsrFiltr, when the last instance of
```

3 Names Management

3.1 Functional Specification

3.1.1 Overview

```
Any file shared on the remote server is identified with an UNC name, which includes the
We need implement our own file name query routine for our purpose. Though osr filter pr
There's another possiblility that a server could have different names, such as the host
For mapped driver letters, we must handle this case by querying the symbol link of the
Main tasks for this part:

    1, Query the full path name for a specified file
    2, Convert different server names (mapped driver, ip address) to a unique and commo
```

3.1.2 Naming support routines

```
Query the long path name for a given FileObject
NTSTATUS
SfQueryFileName(
      IN PFILE_OBJECT,
      IN OUT PUNICODE_STRING
      );

Construct the file name on ost servers
NTSTATUS
SfConstructOstName(
      IN PCLUSTER_CONTEXT,
```

```
    IN UNICODE_STRING,
    IN ULONG OstIndex,
    OUT PUNICODE_STRING
    );
```

3.2 Use Case

In the osr filter, \Filter\fdispatch.cpp, FilterCompletionWorkRoutine calls GetFileLon
The ConstructOstName is to build the file long names on a spceial ost server. It's to b

3.3 Logic Specification

```
/*
 * QueryFileName
 *   This routine will try and get the name of the given file object.
 *   This is guaranteed to always return a printable string (though it
 *   may be NULL). This will allocate a buffer if it needs to.
 *
 * Arguments:
 *   FileObject: the file object we want the name for
 *   Name: unicode name string, to contain the newly allocated buffer
 *
 * Return Value:
 *   NT status code
 */
NTSTATUS
SfQueryFileName(
    IN PFILE_OBJECT FileObject,
    IN OUT PUNICODE_STRING Name
    )
{
    Construct irp to query the FileNameInformation on the specified FileObject. Then re
}
/*
 * ConstructOstName
 *     Construct the shared file path name on the specified OST server
 *
 * Arguments:
 *     Ccb: the CLUSTER_CONTEXT, represents the whole cluster
 *     NameonMds: the full file name on MDS server in UNICODE
 *     OstIndex:  the OST server index number
 *     NameonOst: to contain the full path name on Ost share
 *
 * Return Value:
 *     NT status code
```

```
 *
 * Notes:
 *     N/A
 */
NTSTATUS
ConstructOstName(
    IN PCLUSTER_CONTEXT  Ccb,
    IN UNICODE_STRING    NameonMds,
    IN ULONG             OstIndex,
    IN PUNICODE_STRING   NameonOst
    )
{
    just replace the MDS share point with the corresponding OST share point
}
```

3.4 State Management

N/A

4 File Manipulation Support

4.1 Functional Specification

The file operations here cover opening, closing, reading, EA querying.

```
SfOpenFile:
    Open a specified file
SfCloseFile:
    Close the opened file handle
SfQueryFileEA:
    Query EA values
SfReadFile:
    Read data from file
SfWriteFile:
    Write data to file
```

4.2 Use Case

```
Example: reading the ost configuraiton information, i.e. the content
         of file ".OST_MAP" under root directory:
    /* 1, open the ost map file under root: ".OST_MAP" */
    status = SfOpenFile(SF_OST_MAP_NAME, &handle);
    if (!NT_SUCCESS(status)) {
        return status;
    }
```

```
    /* 2, get the total length of ost map file */
    status = ZwQueryInformationFile(
                    handle,
                    &Iosb,
                    &FSI,
                    sizeof(FSI),
                    FileStandardInformation
                    );
    if (!NT_SUCCESS(status)) {
        goto errorout;
    }
    /* 3, allocate memory buffer for ostmap */
    ostmap = ExAllocatePoolWithTag(...)

    /* 4, now read the ost map information */
    status = SfReadFile(
                    handle,
                    0,
                    FSI.AllocationSize,
                    ostmap,
                    &Iosb.Information
                    );
```

4.3 Logic Specification

```
/*
 * SfOpenFile
 *     Open a file
 *
 * Arguments:
 *     FileName: file name to be opened
 *     Handle: to store the file handle opened
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfOpenFile(
    IN PUNICODE_STRING FileName,
    IN BOOLEAN bRead,
    IN BOOLEAN bWrite,
    OUT HANDLE * Handle,
    OUT PFILE_OBJECT * FileObject
```

```
    )
{
    NTSTATUS            status;
    IO_STATUS_BLOCK     ioSb;
    OBJECT_ATTRIBUTES   objAttr;
    InitializeObjectAttributes(
        &objAttr,
        FileName,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
        NULL,
        NULL
        );
    status = ZwCreateFile(
                  Handle,
                  // SYNCHRONIZE |
                  (bRead ? GENERIC_READ : 0) |
                  (bWrite ? GENERIC_WRITE : 0),
                  &objAttr,
                  &ioSb,
                  NULL,
                  FILE_ATTRIBUTE_NORMAL,
                  FILE_SHARE_READ | FILE_SHARE_WRITE,
                  FILE_OPEN,
                  0,
                  NULL,
                  0);
    if (!NT_SUCCESS(status)) {
        SF_LOG_PRINT( SFDEBUG_DISPLAY_ERROR_MESSAGE,
                      ("SfOpenFile %wZ failed: 0x%08lx\n",
                      FileName, status));
        goto errorout;
    } else {
        SF_LOG_PRINT(SFDEBUG_DISPLAY_FILE_OPENCLOSE,
            ("SfOpenFile: handle = %xh for %wZ\n", *Handle, FileName));
    }
    if (FileObject) {
        /* get the FileObject with the opened handle */
        status = ObReferenceObjectByHandle(
                            *Handle,
                            MAXIMUM_ALLOWED,
                            *IoFileObjectType,
                            KernelMode,
                            FileObject,
                            NULL );
        if (!NT_SUCCESS(status)) {
            SF_LOG_PRINT(SFDEBUG_DISPLAY_ERROR_MESSAGE,
```

```
                ("SfOpenFile:          failed to get the fileobject of file name = %wZ\
                        FileName));
            SfCloseFile(*Handle);
            goto errorout;
        } else {
            SF_LOG_PRINT(SFDEBUG_DISPLAY_FILE_OPENCLOSE,
                ("SfOpenFile:          FileObject = %xh for %wZ\n", *FileObject, FileNa
        }
    }
errorout:
    return status;
}
/*
 * SfReadFile
 *     Read data from file
 *
 * Arguments:
 *     Nandle: file handle to be read
 *     Offset: the start address to be read from
 *     Length: length in bytes to be read
 *     Buffer: buffer to contain the data
 *     BytesRead: bytes successfully read from file
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfReadFile(
    IN  HANDLE          Handle,
    IN  PLARGE_INTEGER  Offset,
    IN  ULONG           Length,
    OUT PUCHAR          Buffer,
    OUT PULONG          BytesRead
    )
{
    NTSTATUS            status;
    IO_STATUS_BLOCK     ioSb;
    ioSb.Status = 0;
    ioSb.Information = 0;
    status = ZwReadFile(
                    Handle,
                    0,
                    NULL,
```

```
                    NULL,
                    &ioSb,
                    Buffer,
                    Length,
                    Offset,
                    NULL);
    if (NT_SUCCESS(status)) {
        *BytesRead = ioSb.Information;
    } else {
        *BytesRead = 0;
    }
    return status;
}
/*
 * SfWriteFile
 *     Write data to file
 *
 * Arguments:
 *     Nandle: file handle to be written
 *     Offset: the start address to be written from
 *     Length: length in bytes to be written
 *     Buffer: buffer containing the data
 *     BytesWirtten: bytes successfully written to file
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfWriteFile(
    IN  HANDLE          Handle,
    IN  PLARGE_INTEGER  Offset,
    IN  ULONG           Length,
    OUT PUCHAR          Buffer,
    OUT PULONG          BytesWritten
    )
{
    NTSTATUS            status;
    IO_STATUS_BLOCK     ioSb;
    ioSb.Status = 0;
    ioSb.Information = 0;
    status = ZwWriteFile(
                    Handle,
                    0,
```

```
                    NULL,
                    NULL,
                    &ioSb,
                    Buffer,
                    Length,
                    Offset,
                    NULL);
    if (NT_SUCCESS(status)) {
        *BytesWritten = ioSb.Information;
    } else {
        *BytesWritten = 0;
    }
    return status;
}
/*
 * SfQueryFileEA
 *     This routine is to query content of the specified
 *     EA of the given file.
 *
 * Arguments:
 *     device: device name of the underlying fs
 *     file: file object of the file to be queried
 *     EaName: the EA to be queried
 *     Length: length in bytes of the input buffer
 *     Buffer: buffer to contain the EA value
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfQueryFileEA(
    IN  PDEVICE_OBJECT  device,
    IN  PFILE_OBJECT    file,
    IN  PUCHAR          EaName,
    IN  ULONG           Length,
    OUT PUCHAR          Buffer
    )
{
    NTSTATUS            status;
    IO_STATUS_BLOCK     iosb;
    KEVENT              event;
    PIRP                irp = NULL;
    PIO_STACK_LOCATION  irpSp = NULL;
```

```
PFILE_FULL_EA_INFORMATION EaBuffer = NULL;
PFILE_GET_EA_INFORMATION  EaList = NULL;
ULONG                     EaLength = 0;
ULONG           EaListLength = 0;
UCHAR           EaListBuffer[sizeof(FILE_GET_EA_INFORMATION ) + 31];
/* initialize the status value */
status = STATUS_SUCCESS;
/* grab a reference of the file object */
ObReferenceObject(file);
/* initialize the EaList */
EaList = (PFILE_GET_EA_INFORMATION)EaListBuffer;
EaList->NextEntryOffset = 0;
EaList->EaNameLength = (UCHAR)strlen(EaName);
RtlCopyMemory(
    &(EaList->EaName[0]),
    EaName,
    EaList->EaNameLength
    );
EaList->EaName[EaList->EaNameLength + 1] = 0;
EaListLength =      sizeof(FILE_GET_EA_INFORMATION) - 1 +
        EaList->EaNameLength + 1;
/* allocate EaBuffer */
EaLength = sizeof(FILE_FULL_EA_INFORMATION) + Length +
           EaList->EaNameLength;
EaBuffer = (PFILE_FULL_EA_INFORMATION)
        ExAllocatePoolWithTag(NonPagedPool, EaLength, SFLT_POOL_TAG);
if (!EaBuffer) {
    SF_LOG_PRINT(SFDEBUG_DISPLAY_ERROR_MESSAGE,
        ("SfQueryFileEA: failed to allocate memory for %wZ\n", &file->FileName));
    status = STATUS_INSUFFICIENT_RESOURCES;
    goto errorout;
}
RtlZeroMemory(EaBuffer, EaLength);
/* allocate new irp */
irp = IoAllocateIrp(device->StackSize, FALSE);
if (NULL == irp) {
    SF_LOG_PRINT(SFDEBUG_DISPLAY_ERROR_MESSAGE,
        ("SfQueryFileEA: failed to allocate Irp for %wZ\n", &file->FileName));
    status = STATUS_INSUFFICIENT_RESOURCES;
    goto errorout;
}
irp->Tail.Overlay.OriginalFileObject = file;
irp->Tail.Overlay.Thread = PsGetCurrentThread();
irp->RequestorMode = KernelMode;
irp->UserIosb = &iosb;
irp->Flags = IRP_SYNCHRONOUS_API;
```

```
/* set up the next I/O stack location. */
irpSp = IoGetNextIrpStackLocation( irp );
irpSp->MajorFunction = IRP_MJ_QUERY_EA;
irpSp->FileObject = file;
irpSp->DeviceObject = device;
irp->Tail.Overlay.AuxiliaryBuffer = EaListBuffer;
irpSp->Parameters.QueryEa.EaList = EaListBuffer;
irpSp->Parameters.QueryEa.EaListLength = EaListLength;
if (device->Flags & DO_BUFFERED_IO) {
    irp->AssociatedIrp.SystemBuffer = EaBuffer;
    irp->UserBuffer = EaBuffer;
    irp->Flags |= (ULONG) (IRP_BUFFERED_IO |
                           IRP_DEALLOCATE_BUFFER |
                           IRP_INPUT_OPERATION);
} else if (device->Flags & DO_DIRECT_IO) {
    PMDL mdl = NULL;
    mdl = IoAllocateMdl( EaBuffer, Length, FALSE, TRUE, irp );
    if (mdl == NULL) {
        status = STATUS_INSUFFICIENT_RESOURCES;
        goto errorout;
    }
    __try {
        MmProbeAndLockPages( mdl, KernelMode, IoWriteAccess );
    } __except(EXCEPTION_EXECUTE_HANDLER) {
        SF_LOG_PRINT(SFDEBUG_DISPLAY_ERROR_MESSAGE,
                           ("SfQueryFileEa: failed to allocate memory when queryin
        IoFreeMdl(irp->MdlAddress);
        irp->MdlAddress = NULL;
        status = STATUS_INSUFFICIENT_RESOURCES;
    }
} else {
    irp->UserBuffer = EaBuffer;
}
irpSp->Parameters.QueryEa.Length = EaLength;
irpSp->Flags = SL_RESTART_SCAN | SL_RETURN_SINGLE_ENTRY;
/* initialize the completion context */
KeInitializeEvent(&event, SynchronizationEvent, FALSE);
/* set the Irp completion routine */
IoSetCompletionRoutine( irp, SfCommonIrpComplete,
                        &event, TRUE, TRUE, TRUE);
/* issue the irp to the lower layer device */
status = IoCallDriver(device, irp);
/* wait for the completion routine to be called */
if (STATUS_PENDING == status) {
    NTSTATUS localStatus = KeWaitForSingleObject(
                                        &event,
```

```
                                                Executive,
                                                KernelMode,
                                                FALSE, NULL );
            SF_ASSERT(STATUS_SUCCESS == localStatus);
        }
        if (NT_SUCCESS(status)) {
            SF_ASSERT(EaBuffer->EaValueLength <= Length);
            RtlCopyMemory(Buffer, &EaBuffer->EaName[EaBuffer->EaNameLength + 1],
                          EaBuffer->EaValueLength );
        }
    errorout:
        /* release the refer of the fileobject */
        ObDereferenceObject(file);
        if (irp) {
            IoFreeIrp(irp);
        }
        if (EaBuffer) {
            ExFreePool(EaBuffer);
        }
        return status;
    }
    #define SfCloseFile ZwClose
```

4.4 State Management

N/A

5 LOV Config Query Support

5.1 Functional Specification

We place here the routines to query LOV OST/File EA configuraiton. There are several tasks:

1. SfReadOstMap: Reading the ost configuration file and store the data in CLUSTER_CONTEXT
2. SfMonitorOstChanges: monitor the root directory changes and filter the update on OstMap
3. SfQueryLovDist: Query "LOV_DIST" EA for the file being read or wirtten and store it in STREAM_CONTEXT

5.2 Use Case

```
SfReadOstMap:
It's to be called when creating CLUSTER_CONTEXT in PostDoCreate, the first time to acce
```

Also will be called in the MDS root directory change monitor thread to update the confi
SfMonitorOstChange:
It's being called in the root directory change monitor thread.
SfQueryLovDist:
called when constructing STREAM_CONTEXT in PostDoCreate, only do the query for files, t

5.3 Logic Specification

```
/*
 * SfReadOstMap
 *     This routine is to read the ost map informaiton from
 *     file: ".SRV_MAP" under root directory
 *
 * Arguments:
 *     Ccb: the cluster context representing a lustre cluster
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     1, The access of SRV_MAP file will be bypassed by the
 *        filter driver itself.
 *     2, It's to be called in IRP_MJ_CREATE, so no need to
 *        impersonate the user's security context.
 */
NTSTATUS
SfReadOstMap(IN PCLUSTER_CONTEXT Ccb)
{
    NTSTATUS    status;
    HANDLE      handle;
    struct ost_map * ostmap;
    IO_STATUS_BLOCK     iosb;
    FILE_STANDARD_INFORMATION   FSI;
    /* open the ost map file under root: ".SRV_MAP" */
    status = SfOpenFile(Ccb->SrvMap.Name, &handle);
    if (!NT_SUCCESS(status)) {
        return status;
    }
    /* get the total length of ost map file */
    status = ZwQueryInformationFile(
                    handle,
                    &iosb,
                    &FSI,
                    sizeof(FSI),
                    FileStandardInformation
                    );
```

```
    if (!NT_SUCCESS(status)) {
        goto errorout;
    }
    /* allocate memory buffer for ostmap data */
    ostmap = ExAllocatePoolWithTag(
                    NonPagedPool,
                    FSI.AllocationSize.LowPart,
                    SFLT_POOL_TAG
                    );
    if (NULL == ostmap) {
        status = STATUS_INSUFFICIENT_RESOURCES;
        goto errorout;
    }
    /* now read the ost map information */
    status = SfReadFile(
                    handle,
                    0,
                    FSI.AllocationSize.LowPart,
                    ostmap,
                    &iosb.Information
                    );
    if (!NT_SUCCESS(status)) {
        ExFreePool(ostmap);
        goto errorout;
    }
    /* update the ostmap to global CLUSTER_CONTEXT with lock acquired */
    ......
    Ccb->SrvMap.map = ostmap;
    ......
errorout:
    SfCloseFile(handle);
    return status;
}
/*
 * SfMonitorOstChange
 *     This routine is to query directory change notification
 *     record of the MDS root share point to monitor any changes
 *     on the .OST_MAP to trace the OST deletion or insertion.
 *
 * Arguments:
 *     Ccb: The cluster context, which has MdsRoot name setted.
 *
 * Return Value:
 *     BOOLEAN: TRUE if the file was just changed / updated, or FALSE
 *
 * Notes:
```

```
 *     N/A
 */
BOOLEAN
SfMonitorOstChange(IN PCLUSTER_CONTEXT Ccb)
{
    1, call SfOpenFile to open the Mds root directory: Ccb->MdsRoot
    2, call ZwNotifyChangeDirectoryFile to query the change notificaiton
    3, filter the FILE_NOTIFY_INFORMATON
    4, if we get changes on file .SRV_MAP, we need read the content of .SRV_MAP and upd
    5, in case get the exiting indiction, such as system shutdown, deletion of the clus
}
/*
 * SfQueryLovDist
 *     This routine is to query the value of "LOV_DIST"
 *     EA of a given file on MDS.
 *
 * Arguments:
 *     Name: full path name of the file to be queried
 *     lmm:  buffer to store the content of the EA
 *     size: length in bytes of the lmm buffer
 *
 * Return Value:
 *     Nt status code
 *
 * Notes:
 *     Later we could use irp method with the existing
 *     FileObjectto skip another IRP_MJ_CREATE/CLEANUP/
 *     CLOSE re-entry
 */
NTSTATUS
SfQueryLovDist(
    IN PWCHAR name,
    IN ULONG  size
    IN OUT struct lov_mds_md *  lmm,
    )
{
    NTSTATUS    status;
    HANDLE      handle;
    /* open the file on MDS */
    status = SfOpenFile(name, &handle);
    if (!NT_SUCCESS(status)) {
        return status;
    }
    /* query the value for EA: XATTR_NAME_LOV_DIST */
    status = SfQueryFileEA( handle,
                            XATTR_NAME_LOV_DIST,
```

```
                                size,
                                (PUCHAR) lmm
                            );
errorout:
    /* done! do close the file */
    SfClose(handle);
    return status;
}
```

5.4 State Management

```
N/A
```

6 Security Support

6.1 Functional Specification

Two functions needed to create and destroy the user's security context:

```
SfCreateSCC:
    To capture the user's security context during IRP_MJ_CREATE.
    The security client context will be stored in STREAM_CONTEXT.
SfDestroySCC:
    To destroy the security context, called in SfStreamContextDestroy.
```

6.2 Use Case

SCC Creation/Destruction:

1. SfStreamContextCreate calls SfCreateSCC to capture the user's security context, and store it in NameEntry.
2. SfStreamContextDestroy calls SfDestorySCC to destroy the security context.

SCC impersonation:

1. call SeImpersonateClientEx to impersonate the stored security context
2. do file open operations as if we are in the user's context with the user's privilege.
3. call PsRevertToSelf to restore to the original context, terminate the impersonation.

6.3 Logic Specification

```
/*
 * SfCreateSCC
 *     Capture the user's security client context for later usage
 *     to impersonate the user
 *
 * Arguments:
 *     N/A
 *
 * Return Value:
 *     PSECURITY_CLIENT_CONTEXT: The user's security context
 *
 * Notes:
 *     This routine should be called in the user's context, normally
 *     in IRP_MJ_CREATE or IRP_MJ_DEVICE_CONTROL ...
 */
PSECURITY_CLIENT_CONTEXT
SfCreateSCC()
{
    SECURITY_QUALITY_OF_SERVICE SQS;
    PSECURITY_CLIENT_CONTEXT    SCC;
    /* allocate SCC in NonPaged Pool */
    SCC = (PSECURITY_CLIENT_CONTEXT)
           ExAllocatePool(NonPagedPool,
                          sizeof(SECURITY_CLIENT_CONTEXT));
    if (!SCC) {
        return NULL;
    }
    /* Initialize SQS to capture SCC */
    RtlZeroMemory(&SQS, sizeof(SECURITY_QUALITY_OF_SERVICE));
    SQS.Length = sizeof(SECURITY_QUALITY_OF_SERVICE);
    SQS.ImpersonationLevel = SecurityImpersonation;
    SQS.ContextTrackingMode = SECURITY_STATIC_TRACKING;
    SQS.EffectiveOnly = FALSE;
    SeCreateClientSecurity(
                PsGetCurrentThread(),
                &SQS,
                TRUE,
                SCC );
    return SCC;
}
/*
 * SfDestroySCC
 *     Destory the user's security client context
 *
```

```
 * Arguments:
 *     SCC: the security context to be freed
 *
 * Return Value:
 *     N/A
 *
 * Notes:
 *     N/A
 */
VOID
SfDestroySCC(PSECURITY_CLIENT_CONTEXT  SCC)
{
    SeDeleteClientSecurity(SCC);
    ExFreePool(SCC);
}
```

6.4 State Management

We will create a single SECURITY_CLIENT_CONTEXT for every file and it's to be stored in STREAM_CONTEXT, So the creation and destruction are to be done according to the lifecycle of STREAM_CONTEXT.

7 I/O Dispatch

7.1 Functional Specification

7.1.1 Overview

```
This part is the core part of the parallel I/O. Here we will take over the whole contro
```

7.1.2 I/O dispatch routines

```
SfReadWrite:
    The handler routine for IRP_MJ_READ and IRP_MJ_WRITE. For lustre
    file I/O, it will call SfRedirectIO to handle it specially.
SfRedirectIO:
    Do spliting the big request to smaller ones, basing on the file
    stripe distribution infromaiton.
SfRedirectIOComplete:
    The completion routine of the reidrected Irps. It does status
    tracing and final cleaning up.
```

7.2 Use Case

The typical procedure of the redirect I/O likes the followings:

- **Reading Process:**

1. User issues a reading request to the files on remote MDS server
2. A cached I/O IRP_MJ_READ is to be called ultimately to our filter driver.
3. SfReadWrite will take over the request, and return STATUS_PENDING to the osr filter to indict that osr filter is to be bypassed to response the request.
4. SfReadWrite will execute a workitem, then SfRedirectIO will be called in the workitem callback routine.
5. So it meets the conditions in SfReadWrite to call SfRedirectIO to split and dispatch to several OST servers, ex: ost1, ost2.
6. The cached sub irps will trigger paging I/Oes on ost1 and ost2, but these requests are out of our concern. The paging reading request will be fed via the data from remote ost1 and ost2 servers.
7. Then system should call sub irp's completion routines: SfRedirectIOComplete. The completion will trace the result of the request and do cleaning up. We'll complete the original Irp with corresponding status code after all the sub irps are completed.
8. Now the original request to MDS server is processed and the whole reading circle goes to it's end.

- **Writing Process:**

1. User issues a write request on a file on remote MDS server.
2. Osr filter driver will catch the routine IRP_MJ_WRITE (cached I/O normally). If it's written to files on MDS server, we need return STATUS_PENDING to tell osr filter that we'll take over the handling of the writing request.
3. SfReadWrite will call SfRedirectIO in a workitem to handle the I/O spliting and redirecting.
4. Normally for local file sysem, when a cached writing arrives, file system driver will call CcCopyWrite to prepare the pages and write data into the cache pages, then return to user and the user is to be notified that it succeeds to write data. (But actually the data is still in the cache. The cache manager will later issue a request of IRP_MJ_WRITE (paging I/O, non-cached) to write all the cache into the underlying devices.) But for a network redirector, the writing procedure is different. It will be directly written to the remote server even the I/O is marked as cached, so there are no paging writing at all.

5. Then SfReadWrite will call SfRedirectIO to do to complete the request. Same to the procedure of reading on steps: 5, 7 and 8.

6. After SfReadWrite returns, the data has been written to the remote OST servers. Then system gets the information that the writing request to MDS server is finished.

7.3 Logic Specification

```
Structures for Irp completion routine:
/* I/O request record per stipe/ost */
typedef struct _SF_RW_SUB_REQ {
    UNICODE_STRING  Name;       /* full path name on the ost */
    PFILE_OBJECT    FileObject; /* FileObject of this file */
    PDEVICE_OBJECT  FsDevice;   /* the file system device object */
    PIRP            Irp;        /* the irp for this sub request */
    NTSTATUS        Status;     /* result of this sub irp */
    ULONG           Length;     /* bytes length to be read or wirtten */
} SF_RW_SUB_REQ, *PSF_RW_SUB_REQ;
/* Irp completion context to trace sub irps */
#define SW_RW_CONTEXT_MAGIC 'RWCM'
typedef struct _SF_RW_CONTEXT {
    ULONG   Magic;              /* Magic & Flags*/
    ULONG   Flags;
    ULONG   RefCount;           /* Reference count*/
    ULONG   NumOfIrps;          /* Total count of sub reqests */
    PIRP    OriginalIrp;        /* The original request packet */
    WORK_QUEUE_ITEM WorkItem;   /* Workitem to do cleaning up */
    SF_RW_SUB_REQ SubReqs[0];   /* the children request records */
} SF_RW_CONTEXT, *PSF_RW_CONTEXT;
Parallel I/O handler routines:
/*
 * SfReadWrite
 *     The driver callback routine for IRP_MJ_READ and IRP_MJ_WRITE
 *
 * Arguments:
 *     DeviceObject: the device object of the unmaed fs device
 *     Irp: the Irp packet
 *
 * Return Value:
 *     nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
```

```
SfReadWrite
(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
#ifdef NEW_FDDK_INTERFACE
    IN ,PPRIVATE_CONTEXT PrivateContext
#endif NEW_FDDK_INTERFACE
    )
{
    1, check the STREAM_CONTEXT of the file
    2, queue a WorkItem to decrease the IRQL
    3, system will execute SfRedirectIO in the WorkItem dispatch routine
}
/*
 * SfRedirectIO
 *     This routine is to split the original I/O on MDS to
 *     several and issue them to different OSTs.
 *
 * Arguments:
 *     DeviceObject: the device object of the unmaed fs device
 *     Irp: the Irp packet
 *     Scb: the STREAM_CONTEXT of the file being accessed
 *
 * Return Value:
 *     nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfRedirectIO(
    IN PDEVICE_OBJECT   DeviceObject,
    IN PIRP             Irp,
    IN PSTREAM_CONTEXT  Scb
    )
{
    1, split the original irp basing the file's stripe distribution
       the sub request are stored in the SF_RW_CONTEXT structure
    2, build the file name path on OST share points
    3, issue the sub requests to corresponding OST devices
    4, the cleanup job is to be done by the irp completion routine
}
/*
 * SfRedirctIOComplete
 *     The irp complete routine where we get the last control on the
 *     issued Irp. We need store the Irp process result and queue a
```

```
 *     workitem to do cleanup if this is the last Irp.
 *
 * Arguments:
 *     DeviceObject: the device object of the unmaed fs device
 *     Irp: the Irp packet
 *     context: the SF_RW_CONTEXT record
 *
 * Return Value:
 *     nt status code
 *
 * Notes:
 *     N/A
 */
NTSTATUS
SfRedirectIOComplete(
    IN PDEVICE_OBJECT  DeviceObject,
    IN PIRP            SubIrp,
    IN PVOID           Context
    )
{
    PSF_RW_CONTEXT rwContext = Context;
    ...
    1, check the sub request result and update the original irp
    2, do cleaning up of this sub request
    3, complete the original irp if it's the last completed sub reqest
}
```

7.4 State Management

N/A

8 Osr FDDK Issues

8.1 Functional Specification

8.1.1 OsrFilter Callback routines

We need provide 3 types of callback routines when registering OSR_FILTER_CALLBACK. HLD document already describes this part in detail.

1. Mount and DisMount
2. Create: Complete
3. Read/Write:Dispatch

8.1.2 File system drivers to be filtered

In default osr recognizer will attach itself to the stacks of all the file system drivers. First it maintains a table named: "Table of Watched NT File System Names", in which all known windows nt file systems are listed. The recognizer will start a timer to check the file systems in the table after a prieod of time. Second, it registers the file system change callback routine to monitor all the dynamical loadings of file system drivers.

For us only the network file system driver is under our concern. Then we need modify the Table of Watchted NT File System Names in file Recognizer/recognizer.cpp and the RecognizerFsNotification in Recognizer/fsdetect.cpp to bypass other file systems.

We only only keep the LanmanRedirector in the NT File System Table. And modify the RecognizerFsNotification to hook the known file system drivers in the table, i.e. only one file system LanmanRedirector could be hooked. The behavours on file systems are becoming invisible to us.

8.1.3 Driver module names

Our product will contain 3 driver modules. The drivers contains automatical loading mechanisym inside. To prevent cofliction between other filter productions using osr fddk we need rename the modules for our purpose:

The module names are defined in "inc\config.h".

Osr Module Names	Lustre Parallel I/O Product Names	Comment
OsrFiltr.sys	CfsFM.sys	Fiter Management Module
OsrRec.sys	CfsVM.sys	Volume Management Module
callback.sys	CfsPIO.sys	The Parallel I/O Handler Module

8.2 Use Case

N/A

8.3 Logic Specification

```
Table of Watched NT File System Drivers:
extern OSR_FDDK_RECOGNIZER_FS_NAMES FileSystemNameData[OSR_FDDK_FILTER_FS_TYPE_MAX+1] =
{
    //
    // Table of Watched NT File System Names
    //
    //  Any file system with an entry in this table will result in
    // this driver attempting to catch MOUNT requests for that file
    // system.  If, for example, you didn't care about filtering
    // requests for any HPFS volumes, you would remove HPFS from
    // this list.
```

```
    //
    //  ***NOTE: If you change the number of entries in this table
    //  you must also CHANGE THE VALUE FOR OSR_FDDK_FILTER_FS_TYPE_MAX
    //  in "filter.h"
    //
    //
    // TYPE,   FS Device Name,   Recognizer Name (if there is one)
    //
    {OSR_FDDK_FILTER_FS_TYPE_RDR,  L"\\Device\\LanmanRedirector",       NULL, NULL},
    {OSR_FDDK_FILTER_FS_TYPE_NULL, L"",                                 NULL,  NULL},
};
RecognizerFsNotification:
////////////////////////////////////////////////////////////////////////
//
// RecognizerFsNotification
//
//  This routine is called by NT whenever a new file system registers.
//
// Inputs:
//   DeviceObject - the new file system device object
//   Active - indicates if the file system is loading or unloading
//
// Outputs:
//   None.
//
// Returns:
//   Void function.
//
// Notes:
//  This is valid for both NT 3.51 and NT 4.0 although this mechanism
//  was MS confidential until very recently.
//
////////////////////////////////////////////////////////////////////////
VOID RecognizerFsNotification(PDEVICE_OBJECT DeviceObject,
                                    BOOLEAN Active)
{
    ......
    //
    // if we don't get the file system driver in the watched
    // table, we just return to system without any hooking,
    // since we only care the Lanman network redirector file
    // system driver.
    //
    if(!found) {
        return;
    }
```

```
    ......
}
Driver Entry Point:
//
// DriverEntry
//
// This is the initial entry point for the driver.
//
// Inputs:
//  DriverObject   Pointer to Driver Object created by I/O manager
//  RegistryPath   Pointer to registry path representing this Driver
//
// Returns:
//  Success        To indicate Driver's inituaialization processing
//                 was successful
//  NT ERROR STATUS     Otherwise -- Driver does not remain loaded
//
NTSTATUS
DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
    NTSTATUS code;
    POSR_FILTER_CALLBACK callback;
    //
    // Debug/informational code.
    //
    KdPrint(("pCIFS: DriverEntry called\n"));

    OsrDebugDriverBuildInfo();

    //
    // Initialize common routine(s)
    //
    OsrElogStart(DriverObject);
    //
    // Take a breakpoint if requested to by the appropriate
    // registry parameter.
    //
    OsrRegistryBreakOnEntry(RegistryPath);
    //
    // Initialize the work item free list
    //
    KeInitializeSpinLock(&WorkItemFreeListLock);
    InitializeListHead(&WorkItemFreeList);
    //
    // Build a callback structure.  This will be used to register with
    // the filter driver.
```

```
//
callback = ExAllocatePool(NonPagedPool,sizeof(OSR_FILTER_CALLBACK));
//
// If the allocation fails, exit.
//
if (!callback) {

    KdPrint(("pCIFS: DriverEntry: Allocate failed for callback\n"));
    OsrLogDriverError(L"Alloc failed", STATUS_INSUFFICIENT_RESOURCES);
    return(STATUS_INSUFFICIENT_RESOURCES);

}
//
// Initialize the callback data structure.  The general structure
// allows two entries PER IRP_MJ_* value - one for pre-processing,
// one for post-processing.  In addition, there is also an entry
// just for mount processing.
//

RtlZeroMemory(callback, sizeof(OSR_FILTER_CALLBACK));

//
// Hooking the routines which we are concerning ...
//
callback->Dispatch[IRP_MJ_CREATE] = DoCreate;
callback->Complete[IRP_MJ_CREATE] = PostDoCreate;
callback->Dispatch[IRP_MJ_SET_INFORMATION] = DoSetInformation;
callback->Dispatch[IRP_MJ_QUERY_INFORMATION] =
                                  DoQueryInformation;
callback->Dispatch[IRP_MJ_READ]   = DoReadWrite;
callback->Complete[IRP_MJ_WRITE]  = DoReadWrite;
callback->Mount = DoMount;
callback->Dismount = DoDismount;

//
// Ensure the filter driver is loaded before calling it.
//
code = OsrLoadDriver(FDDK_FILTER_NAME_L);

if (!NT_SUCCESS(code) && code != STATUS_IMAGE_ALREADY_LOADED) {
    //
    // Failure
    //

    KdPrint(("pCIFS: DriverEntry: Filter not loaded 0x%x\n", code));
```

```
            ExFreePool(callback);

            OsrLogDriverError(L"No filter", code);

            return(code);

        }
        //
        // Now register our callback entry points.
        //
        code = OsrFilterRegister(callback, OSR_FILTER_VERSION, &CallbackHandle);
        if (!NT_SUCCESS(code)) {

            ExFreePool(callback);
            KdPrint(("Registration for callback failed\n"));

            OsrLogDriverError(L"CB Reg Fail", code);
            return(code);
        }
        //
        // Driver load has completed successfully. Note that at this
        // point,we may already have calls from the filter driver
        // into our dispatch routines.
        //
        return(STATUS_SUCCESS);
    }
```

8.4 State Management

N/A

9 Focus of Inspection

1. The design is reasonable ? Could be better ?
2. Could there be possible DLM deadlocks ?
3. Possibility of stale cache in client side.
4. Section 4 (file operations) and 6 (security support) are similar to original demo project. No need to pay much time on these two sections.

10 References

1. Lustre book, codes, documents

2. CIFS Parallel I/O Filter HLD document
3. Help documents of Ifskit 2003
4. OSR documents in Ifskit 2003
5. Windows NT File System Internals by Rajeev Nagar
6. Inside windows 2000 the 3th edition
7. Documents on http://www.osronline.com

11 Inspection Summary

N/A