Wanda Dumb File System

Shaw Chuang

9th March 1994

1 Introduction

This document describes the Wanda Dumb File System (WDFS), a file system that provides Unix file access on Wanda. The file system has been designed to be very simple and lightweight. WDFS is built using MSRPC and in the current form consists of a set of Wanda library and two Unix daemons. The document also briefly documents the internal structures of WDFS.

2 User Interface

The WDFS provides two sets of interface for file access. The first is a standard C interface to the file system:

- fopen
- fclose
- fdopen
- fseek
- fread
- fwrite
- fscanf
- ftell
- ferror
- feof
- frewind

The syntax of these functions has been kept to be the same as in the C library where possible. The few exceptions were:

- fopen only "r" "w" "r+" "w+" types are supported. The "b" option can also be used, mainly for compatibility reason, since WDFS treats them the same.
- fdopen null function

Manual pages for all of these functions exist and should be referenced in the first instance.

The second set of user interfaces is Unix system call like and this provides a more direct and efficient interface to the underlying RPC mechanism.

- open
- close
- lseek
- read
- write

It maps almost directly onto the MSRPC calls and therefore has the same size limit on the data unit that MSRPC has. The maximum data length should be less than (1500 - 2 - PUID_size - QOS_param - 8). Note that this interface has been integrated with the Wanda Unix Compatible library to allow uniform access to the file system and Wanda socket library. Furthermore, this interface has been carefully coded to have no dependency on the stdio code which exists only in the user space. This makes Wanda kernel WDFS integration is a trivial task.

3 Name Space

The WDFS uses a file name space that is the equivalent to the NFS file name space on the Unix file server. That means Unix file names could be used in a Wanda file access. This is particularly convenient in the Computer Laboratory environment, where the unified Unix file name space is maintained across the Unix machines. This together with the Unix-like library interface enables easy usage and transfer of programmes between the two operating systems.

4 Unix Daemons and Tools

Two MSRPC interfaces were defined for the WDFS service.

- File Interface
- Directory Interface

4.1 File Interface and File Server

The File Interface defines the interface to the file server. This file interface has been defined to be very general and in effect is the interface to a object server.

The present file server interface offers six basic functions, `read, write, create, delete, getlen` and `setlen`. Only four operations, `read, write, getlen` and `setlen` is directly accessible from the WDFS client. `create` and `delete` are reserved operations that is only allowed directly from the directory server `wdds`. An open operation is considered unnecessary. All access to the file server is done through using a object identifier (PUID). The PUID is regarded to be a opaque field which uniquely identifies a file within a file system and it is not interpreted by the client of the file system. Since the current file server is serving Unix files, the PUID is actually a Unix file name. The current implementation of the file server performs no further access control check on any operation, though future implementations of the file system could also use the PUID as a capability for access control. It will be the responsibility of the directory server (see below), which performs all initial access control checks, to issue the correct PUID.

The file server, wdfs, is implemented as a stateless server and that greatly simplified crash recovery. For performance optimisation, the file server maintains state between operations. Even though the open call is not necessary as a client operation, the file server itself does need to open the Unix file to perform the file operations requested in the remote operation. The Unix file descriptor is cached as an optimisation to avoid reopening of files for each operation. When the number of opened file reaches the maximum number allowed in a Unix process, a `Least Recently Used` replacement policy is used to flushed out entries from the cache.

When a file is opened and cached, there is always a possibility of the file becoming stale. A timeout mechanism is therefore implemented to close the file after 60 sec of inactivity. A Unix signal SIGHUP can also be sent to the file server process to cause all open files to be closed.

The file MSRPC interface defines a set of error exceptions that might result from the MSRPC file operation. This list of exception (defined in `master/iface/msdr/files.x`) has been kept to the minimum. They represent only file operation errors and should be usable across all platforms. In this Unix implementation, a mapping is made to convert from the Unix errno onto this set of exception.

4.2 Directory Interface and Directory Server

We have provided a skeleton directory service via the Directory Interface. The purpose of the directory server is to allow access control check and name management. The directory interface has been defined in a fairly generic way which allows the directory server to be modified in future to implement any enhancements.

The directory server carries out the access control check on the file assuming the user is `wanda`. Only if all checks succeed will the directory server return the corresponding PUID, together with information about the particular file, to the calling client.

Just as the file MSRPC interface, the directory MSRPC interface also defines a set of error exceptions that might result from the MSRPC directory operation (defined in `master/iface/msdr/files.x`. A mapping is made to convert from the Unix errno onto this set of exception.

Two of the `files` interface procedures `create` and `delete`, which performs the creation and deletion of file objects, are reserved for wdds. This will ensure the wdds maintaining a consistent view of the file system.

The wdds interface permits a list of file servers to be returned with each of the PUID returned to the calling client, thus allowing replication of the WDFS. The selection of which actual wdfs to use will be a matter of client function.

5 Wanda Library

When the open call (fopen or open) is made, a MSRPC call to the directory server is first made, with the appropriate access mode, file types etc parameters. It locates the WDFS directory server by obtaining its address from the environment variable WDDS Any number of alternative directory servers can be listed in this variable. The format is:

The format is:

$$\texttt{WDDS} \;=\; "h_{10}.h_{11}.h_{12}.h_{13}, h_{20}.h_{21}.h_{22}.h_{23}$$

Where h_{ij} is the j^{th} octet in internet dotted notation of the MSNL address of host h_i. No assumption should be made on the selection of the current directory server from the WDDS environment variable.

5.1 Tools

Monitoring tool `wdfs_ping` and `wdds_ping` are for checking the state of the the `wdfs` and `wdds` daemon respectively.
The usage of the corresponding Unix tool is:

```
wdfs_ping [-c] server_name
wdds_ping [-c] server_name
```

If `-c` option is used, the tool does continuous pinging of the daemon via the NULL procedure of the MSRPC interface.

6 Logging

The file server and directory server output their activities to the standard output. You should check the `unixdaemons` script to find out the logging location.