.TH "PROJECT 3: FILE SYSTEMS AND WAD-BASED FUSE DAEMON" 1 "April 2025" "Version 1.0"

.SH NAME
Project 3: File Systems and WAD-based FUSE Daemon

.SH SYNOPSIS
This project implements a C++ FUSE-based daemon and library to mount and manipulate WAD files as a virtual file system in Linux.

.SH DESCRIPTION
Library filepaths and filenames:
.nf
    /home/reptilian/wad/
        libWad/Wad.h
        libWad/Wad.cpp
    /home/reptilian/wadfs/
        wadfs.cpp
.fi

The WAD file structure is represented using a tree of `Node` objects since this would allow for efficient traversal and insertion. Each `Node` represents a file or directory and is wrapped by a `DescriptorObject` which holds metadata including name, offset, and size.
A `fileMap` (map<string, DescriptorObject*>) provides constant-time lookups from full paths to their descriptor data. Marker descriptors such as "E#M#" (map markers) and "_START/_END" pairs (namespace markers) are used to define directory boundaries.

.TP
.B Wad()
Opens the WAD file, reads the header and descriptor list, and builds the in-memory directory tree and fileMap. Also sets up the structure used by all library functions.

.TP
.B loadWad()
Initializes a Wad object from a file path. Reads the header and descriptor list, constructs an in-memory directory tree, and populates the `fileMap` for fast access.

.TP
.B getMagic()
Returns the 4-character magic string read from the WAD header, which identifies the file format and validates the file as a WAD archive.

.TP
.B isContent(), isDirectory()
Checks whether a path refers to a valid content file or a directory, based on its descriptor length and naming.

.TP
.B getSize(), getContents()
Returns the file size or reads lump content starting from a specified offset. Supports partial reads up to buffer length or file boundary.

.TP
.B getDirectory()
Lists immediate children of a directory by scanning the associated Node's children

vector and formatting names.

.TP
.B createDirectory()
Inserts new directory markers ("_START", "_END") into the descriptor list, updates the WAD file's header, and appends a new directory node to the tree.

.TP
.B createFile()
Inserts a placeholder file descriptor with offset 0 and length -1, reserving it for later writing. Prevents writing to map markers or creating files with invalid names.

.TP
.B writeToFile()
Populates a newly created file's lump region with actual content, updates its length/offset, and rewrites the descriptor section. Ensures content is written only once to each new file.

.PP
The `wadfs` daemon uses the library functions mentioned above via six FUSE callbacks:
.nf
    - `getattr`: Maps file metadata from `DescriptorObject`.
    - `mknod`: Uses `createFile()` to define a new file.
    - `mkdir`: Calls `createDirectory()` to add a namespace.
    - `read`: Uses `getContents()` to retrieve file contents.
    - `write`: Writes content using `writeToFile()`.
    - `readdir`: Lists directory contents with `getDirectory()`.
.fi

.TP
.B wadfs main()
Initializes the Wad object using the file path passed as a command-line argument. It then modifies `argv` to drop the WAD path and launches `fuse_main()` with the parsed arguments and Wad instance. This setup separates WAD parsing from FUSE logic, allowing the virtual filesystem to expose WAD contents using standard commands like ls, cat, and mkdir, making game data easily accessible and modifiable through a familiar POSIX interface.

.SH TESTING
Library functions were tested using `libtest.cpp`, which leverages GoogleTest to evaluate all major functionalities.
The FUSE daemon was tested by mounting WAD files using ./wadfs/wadfs -s sample1-test.wad ./mountdir. File and directory operations were validated by navigating (cd, ls -al), reading (cat mp.txt), and copying files (cp). Write functionality was tested by creating files (vi file.txt), writing content, and creating directories (mkdir ex). After each operation, the filesystem was unmounted (fusermount -u) and remounted to confirm persistence and verify correct behavior of FUSE callbacks (getattr, read, write, mkdir, mknod, readdir).

.SH BUGS
The library functions fail when I try to add a file under a nested directory.

.SH LINK
https://youtu.be/l7IA4P6AXc8

.SH REFERENCES/CITATIONS
"Project 3: File Systems." University of Florida, 2023.

.SH AUTHOR
Aarithi Rajendren