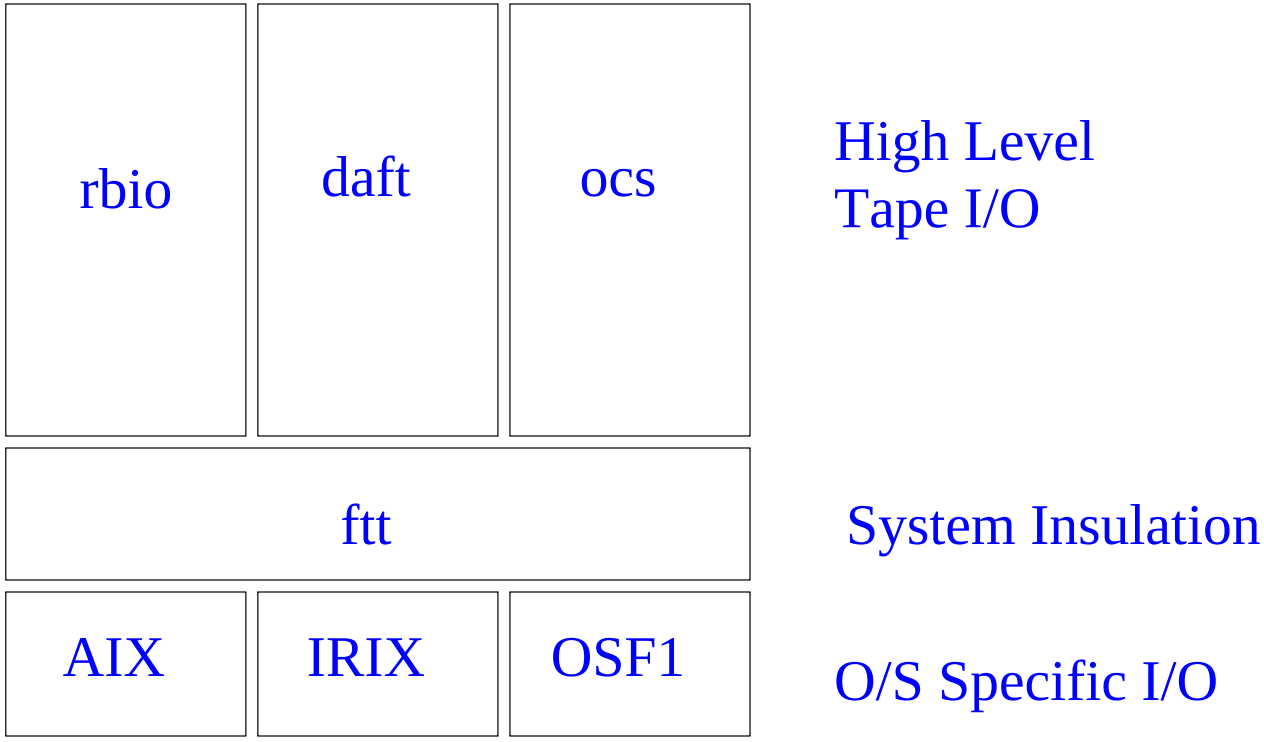
Fermi Tape Tools (ftt)

- **Consolidation of System Dependant Tape I/O Code**
- **Support (so far) for:**

AIX 3.2.x	DLT 4000
IRIX 5.x	Exabyte 8200
OSF1 3.2	Exabyte 8500
SunOS 5.x	Exabyte 8505

ftt Architecture



Design Goals

- Portability
- Device Independence
- O.S. Independence
- Statistics Collection
- Consistent Error Returns

Product Deliverables

- library (libftt.a)
- Setuid executable (ftt_suid)
- Test Program (ftt_test)
- Test Script Suite

```c
#include <stdio.h>
#include <ftt.h>
#include <errno.h>
#include <stdlib.h>

/*
** routine to rewind both drives in parallel
*/
rewind_both(ftt_descriptor dfrom, ftt_descriptor dto) {

    /* rewind "to" drive asynchronously */

    switch(ftt_fork(dto)) {

    case 0:     /* fork succeeded, rewind & report status */
        ftt_rewind(dto);
        ftt_report(dto);
        break;

    case -1:    /* fork failed, rewind in foreground anyway */
        ftt_rewind(dto);
        break;
    }

    /* rewind from drive */
    ftt_rewind(dfrom);

    /* wait for to drive asynchronous rewind */
    ftt_wait(dto);
}
```

```
/*
** tape copy subroutine
** open both drives, set mode on output to match input, copy, and prin
+t
** statistics.
*/
int
tapecopy(char *from , char *to) {
    ftt_descriptor dfrom, dto;
    ftt_stat_buf *bpfrom, *bpto;
    char *errorstring;
    char *buffer;
    int max, length;
    int density, mode;

    dfrom = ftt_open(from, FTT_RDONLY);
    if( 0 == dfrom ) {
        fprintf(stderr, "%s\n", ftt_get_error(0));
        return 0;
    }

    dto = ftt_open(to, FTT_RDWR);
    if( 0 == dto ) {
        fprintf(stderr, "%s\n", ftt_get_error(0));
        return 0;
    }
```



```
    rewind_both(dfrom, dto);
```

```
    /* get initial statistics */

    bpfrom = ftt_init_stats(dfrom);
    bpto = ftt_init_stats(dto);

    /* set output drive to same density/compression as input drive */

    density = atoi(ftt_extract_stats(bpfrom[1], FTT_TRANS_DENSITY));
    mode = atoi(ftt_extract_stats(bpfrom[1], FTT_TRANS_COMPRESS));
    ftt_set_mode(bpto,density,mode,0);

    /* allocate a buffer big enough to read biggest block */

    max = ftt_get_max_blocksize(dfrom);

    buffer = malloc(max);
    if (0 == buffer) {
        fprintf(stderr, "unable to allocate buffer for copy, errno %d"
+, errno);
        return 0;
    }

    length = ftt_read(dfrom, buffer, max);
    while (length != 0 ) { /* not end of tape */
        while (length != 0) { /* not end of file */

            if (length < 0) {
                fprintf(stderr, "%s\n", ftt_get_error(0));
                ftt_close(dfrom);
```

```
                ftt_close(dto);
                return 0;
            }

            ftt_write(dto, buffer,length);
            length = ftt_read(dfrom, buffer, max);
        }
        /* saw an EOF, write one */
        ftt_writefm(dto);

        /* read first block from next file, or another EOF */
        length = ftt_read(dfrom, buffer, max);
    }
    ftt_update_stats(dfrom,bpfrom);
    ftt_update_stats(dto,bpto);

    fprintf(stdout, "read drive statistics:\n");
    ftt_dump_stats(stdout,bpfrom[0]);
    fprintf(stdout, "write drive statistics:\n");
    ftt_dump_stats(stdout,bpto[0]);

    rewind_both(dfrom, dto);

    ftt_close(dfrom);
    ftt_close(dto);
    return 1;
}
```



```
/*
** mainline -- parse arguments and call tapecopy
```

```
*/
int
main(int argc, char **argv) {

    if(argc != 3) {
        fprintf(stderr, "usage: tapecopy from_drive to_drive");
    }

    return !tapecopy(argv[1], argv[2]);
}
```