

## 26 Application/Filesystem Interface

How do application programs access file data?

- explicit read/write operations (conventional)
- memory-mapped files

Read/Write interface

- file data is explicitly copied between disk file and process memory
- programs cannot directly access file data
- potential for double paging (process pages containing file data are paged out to paging space, leading to redundant copies of file data on disk.)

```
FileHandle fhandle;  
int offset, length;  
char buffer[1024];
```

```
fhandle = Open("pathname");  
read(fhandle, offset, buffer, length);  
{read/write file data in buffer}  
write(fhandle, offset, buffer, length);  
close(fhandle);
```

Memory-mapped files

- file is “mapped” into application’s address space by initializing virtual memory so that the file serves as backing store for a region of the application’s address space.
- file data is demand paged upon access to the mapped file
- no copying

- program accesses file data directly
- no double paging
- processes that map the same file share physical memory that caches file data
- elegant integration of file system and virtual memory

```
FileHandle fhandle;  
int offset, length;  
char *address;
```

```
fhandle = Open("pathname");  
map(fhandle, offset, address, length);  
{read/write file data by accessing memory range  
  [address,address+length]}  
unmap(address, length);  
close(fhandle);
```

#### Other filesystem related operations

- create/delete file
- link/unlink: add or remove a name entry for a file
- get/set file attributes:
  - protection (access rights)
  - owner/creator
  - size
  - creation time
  - time of last access/modification
  - file type
- sync/flush: make sure all “dirty” cached file data is written to disk
- lock/unlock: file locking (discussed later)