File Systems Part 4
Implementation Concerns

• System-call access to files vs. mmap access
• File-based page indexing vs. file-system-based page indexing
• File-system block size vs. page size
  – conveniently identical on Weenix (4096 bytes)
Mapped File I/O

Process 1
Virtual Memory

Real Memory

Disk

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

File 1
Multi-Process Mapped File I/O

Process 2
Virtual Memory
Operating Systems in Depth

Real Memory

File 1

Disk

page 0
page 1
page 2
page 3
page 4
page 5
page 6
page 7

page 0
page 1
page 2
page 3
page 4
page 5
page 6
page 7
Traditional I/O involves explicit calls to read and write, which in turn means that data is accessed via a buffer; in fact, two buffers are usually employed: data is transferred between a user buffer and a kernel buffer, and between the kernel buffer and the I/O device.

An alternative approach is to map a file into a process's address space: the file provides the data for a portion of the address space and the kernel's virtual-memory system is responsible for the I/O. A major benefit of this approach is that data is transferred directly from the device to where the user needs it; there is no need for an extra system buffer.
Consistency

typedef struct
    {int flags; char morestuff[OSIZE];} object_t;

object_t object, *mregion;
int fd;
int buf;
fd = open("file", O_RDWR);
mregion = (object_t *)mmap(0, sizeof(object),
    PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
buf = 6;
write(fd, &buf, sizeof(buf));
if (mregion->flags != 6)
    fprintf(stderr, "something is wrong!\n");
Some Data Structures

• pframe_t
  – represents a page frame
    - points to actual frame
    - refers to frame in lists

• mmobj_t
  – refers to list of in-memory pages (page frames)
    of an object such as a file
  – page frames represented by pframe_t’s

• vmarea_t
  – represents a region within an address space
  – into which an object is mapped
    - represented by an mmobj_t
More

- vnode_t
  - represents an open file
  - isolates most of OS from details of file system
  - contains
    - function pointers for file ops
    - mmobj_t for in-memory file pages
    - inode for S5FS files
Caching

- A file’s list of cached pages is in its mmobj_t
- System-call file access
  - get to mmobj_t via containing vnode_t
- Mmap file access
  - get to mmobj_t via vmarea_t
Weenix Address Space Rep

vmmap_t

vmarea_t
0−7fff
x, shared

vmarea_t
8000−1afff
rw, private

vmarea_t
1b000−1bfff
rw, private

vmarea_t
7fffd000−7ffff
rw, private

mmobj_t

file
vnode_t

mmobj_t

mmobj_t

vframe_t

vframe_t

vframe_t

vnode_t

vnode_t

vnode_t
Metadata

- Page frames associated with a file are listed by file offset within mmobj_t
  - page 0, page 1, etc.
- What about indirect, doubly indirect, and triply indirect blocks?
  - don’t appear in file page space
  - can’t be cached with file’s mmobj_t
- What to do?
  - additional mmobj_t for file system
  - in-memory pages listed by file-system offset
  - used only for metadata
Important Kernel Procedures

- `pframe_get(mmobj_t *o, uint32_t pagenum, pframe_t **result)`
  - look up pagenum in o
  - if present, return pointer to pframe_t
  - else allocate page frame, call pframe_fill to fill
- `pframe_fill(pframe_t *pf)`
  - call pf's fill routine to fill
- `s5fs_fillpage(vnode_t *v, off_t offset, void *pagebuf)`
  - fill pagebuf with data from offset in v's file

Note: not all procedures used are mentioned!
More Important Kernel Procedures

- s5_seek_to_block(vnode_t *vnode, off_t seekptr, int alloc)
  - translate from file-based block indexing to file-system-based block indexing
  - allocate indirect blocks if alloc is set
  - calls pframe_get to fetch indirect blocks
    - uses file-system mmobj_t
- blockdev_fillpage(mmobj_t *o, pframe_t *pf)
  - fetches page identified by *o and *pf from disk