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)
Traditional I/O

User Process 1
1: read f1, p0
3: read f1, p1
5: read f3, p0

User Process 2
2: read f2, p0
4: read f2, p1
5: read f3, p0

Buffer Cache
Kernel Memory

File 1
page 0
page 1
page 0

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

File 3
page 0
page 1
page 2
page 3
page 4
page 5
page 6
page 7
Multi-Process Mapped File I/O

Process 2
Virtual Memory
Operating Systems in Depth

Real Memory

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

Disk

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

- **Traditional File I/O**
  ```c
  char buf[BigEnough];
  fd = open(file, O_RDWR);
  for (i=0; i<n_recs; i++) {
    read(fd, buf, sizeof(buf));
    use(buf);
  }
  ```

- **Mapped File I/O**
  ```c
  void *MappedFile;
  fd = open(file, O_RDWR);
  MappedFile = mmap(..., fd, ...);
  for (i=0; i<n_recs; i++)
    use(MappedFile[i]);
  ```

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.
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(buf, &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
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

- \texttt{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
- \texttt{blockdev\_fillpage(mmobj\_t *o, pframe\_t *pf)}
  - fetches page identified by *o and *pf from disk