Accessing Files

Pradipta De
pradipta.de@sunykorea.ac.kr
Today’s Topic

• Understand the techniques to
  – Deliver data when a read call is issued
  – Complete a write call by writing to buffers ready for writing to disk
Recap: VFS objects

Recap: Complete perspective

Types of file access

- Read/write system calls
  - Read is a blocking call; waits till data is transferred to user mode address space
  - write: copies data to page cache and returns
- Synchronous mode
  - Write becomes blocking; wait till data written to disk
  - Read is not affected in this mode
- Memory map
  - File is mapped in memory; accessed by addressing bytes in RAM; no read/write call used
- Direct I/O
  - Direct transfer from user mode space to disk
  - Page cache is bypassed
- Asynchronous mode
  - Non-blocking APIs for read/write; transfer happens in background
File Read

- File reading is page-based
  - Kernel transfers whole page of data at once

- Process issues a read system call, and if data is not in RAM, then
  - Kernel allocates a new page frame
  - Fills the page with suitable portion of file
  - Adds page to page cache
  - Finally copies the requested bytes to process address space
File Read: Implementation Details

• Arguments: filp, buf, count, ppos

• Important steps:
  – Create 2 descriptors: track buf and count; track the progress of data transfer to user buf
  – Calls do_generic_file read: takes the ppos
    • Read pages from disk and copy to user mode buf

• Flow of operation
  – Filp \rightarrow address\_space\_object \rightarrow inode \rightarrow uses the ppos to compute logical number of page that includes first byte of file \rightarrow start the transfer
File Write: Implementation Details

• Arguments: filp, buf, count, ppos

• Write to the page cache and return

• If O-SYNC is set, then transfer to disk

• Flow of operation
  – Filp $\rightarrow$ address_space object $\rightarrow$ inode $\rightarrow$ get a lock on the file $\rightarrow$ mark affected page dirty $\rightarrow$ start copy from user mode space to page cache
Memory Mapping

- Shared mapping
  - Each write operation changes file on disk

- Private mapping
  - Useful for reading the file only;
  - If write is performed, then the mapping stops
  - On write, duplicate is created and the page table is modified to point to duplicate
Data Structures in Memory Map
Back to PST and Reverse Mapping

Process 1: maps 7 to 12
Process 2: maps 7 to 10
Process 3: maps 10 to 30

We may want to find out all processes that map 10 to 12

Priority Search Tree stores the VMA such that the search to find the VMAs covering overlapping regions is faster
Direct I/O Transfer

• Some programs require full control over the I/O data transfer mechanism
  – Without the page cache coming in between
  – Caching is done by the application itself

• Advantage:
  – No page frame wasted due to duplication of caching at page cache and user level cache
  – Read/write calls are faster
  – The data transfer is directly from disk to user buffer … no transfer to kernel buffers
Asynchronous Read-Write

- User mode invokes read/write → function terminates as soon as call is enqueued, but data is not yet available
- POSIX library functions
  - aio_read, aio_write, aio_fsync, ….
- Implementation:
  - Clone the current process, and invoke synchronous read and write system calls
Putting It Together

- File Access Mechanisms
  - Typical Read-Writes
  - Memory Mapping
  - Direct I/O
  - Asynchronous I/O