High Performance Computing
Lecture 37

Matthew Jacob

Indian Institute of Science

File System Performance Ideas

- The main concern: A disk read/write takes a few msecs to complete
- Sequential access: Reading a file byte by byte
  - Using read(): One system call per byte read
  - Alternative: Standard I/O library (stdio)
    - fopen(), fread(), fwrite(), etc
    - Maintains a buffer data read from disk
    - Overhead: There is more copying of the data that was read from the disk
    - Disk -> Stdio buffer -> Your program's buffer

File System Performance Ideas

- Disk Block Caching
  - The system maintains in main memory copies of recently used disk blocks
  - Called the disk block buffer cache
  - They can be accessed from this buffer the next time they are required
  - Could be managed using an LRU like policy
  - Problem: If the system crashes, file data in the buffer cache will be lost
    - The OS periodically flushes the contents of the cache to disk (say once every 30 secs)

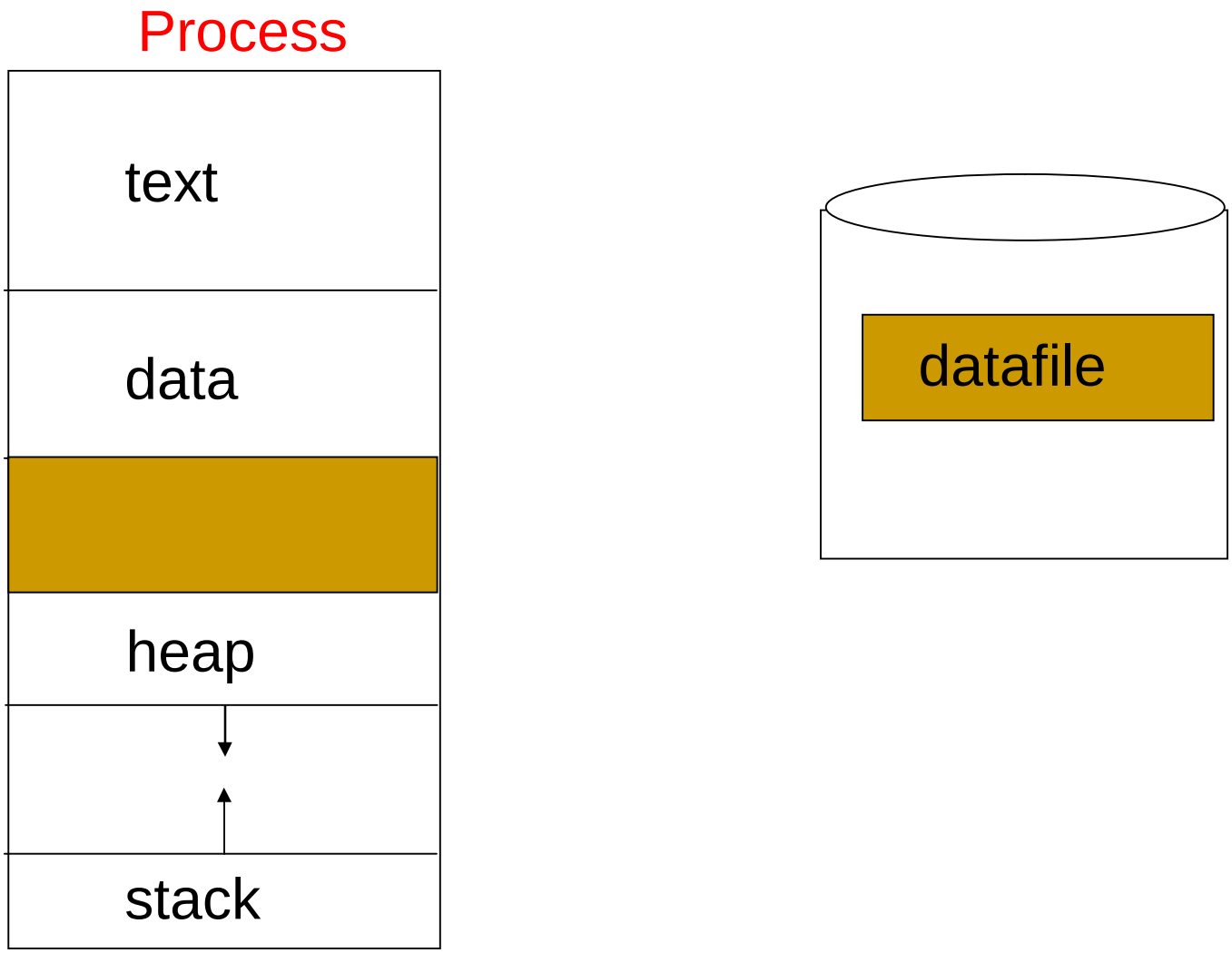
File System Performance Ideas

- Disk Block Buffering/Caching
- Disk Block Pre-fetching
  - Prefetching: Reading something into a cache or buffer before it is actually required
  - Idea: If a file is being read sequentially, a few blocks can be read ahead from the disk

File System Performance Ideas

- Disk Block Buffering/Caching
- Disk Block Pre-fetching
- Memory Mapped Files
  - Idea: Map file into the virtual address space of the process that is accessing it

Memory Mapped Files



Memory Mapped Files

- Traditional open, /lseek/read/write, close are inefficient due to system calls, data copying
- Alternative: map file into process virtual address space
- Access file contents using memory addresses (variables, pointers)
- Could result in a page fault if that part of the file is not currently in memory
- System call: mmap(addr,len,prot,flags,fd,off)
- Some OS's: cat, cp use mmap for file access

File System Performance Ideas

- Disk Block Buffering/Caching
- Disk Block Pre-fetching
- Memory Mapped Files
- Asynchronous I/O
  - Idea: In ordinary file I/O operations (e.g., read), the process has to wait (block) for the disk operations to complete before proceeding to use the file data
  - But the I/O could be made non-blocking

Asynchronous I/O

- Objective: allow programmer to write his/her program to perform I/O without blocking
- eg: SunOS aioread, aiowrite library calls
  - aioread(*fd, buff, numbytes, offset, whence, result*)
    - Reads *numbytes* bytes of data from the file descriptor *fd* into the buffer *buff*, without blocking the process
    - The buffer should not be referenced until after the operation is completed; until then it is in use by the OS
    - Mechanisms are provided for the process to be notified of the actual completion of the I/O operation

File System Performance Ideas

- Disk Block Buffering/Caching
- Disk Block Pre-fetching
- Memory Mapped Files
- Asynchronous I/O