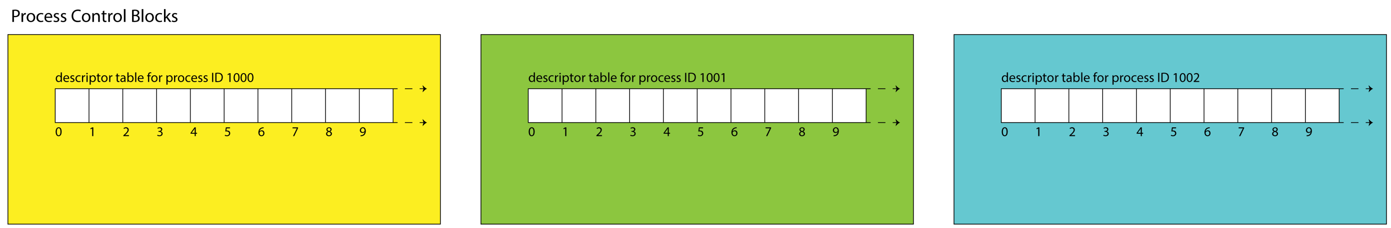
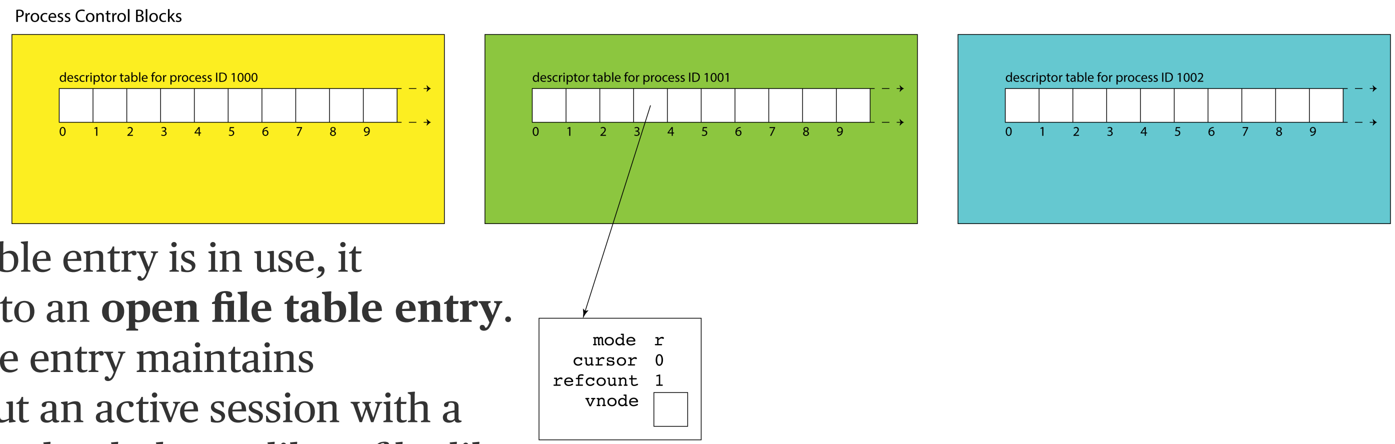
Lecture 04: Filesystem Data Structures



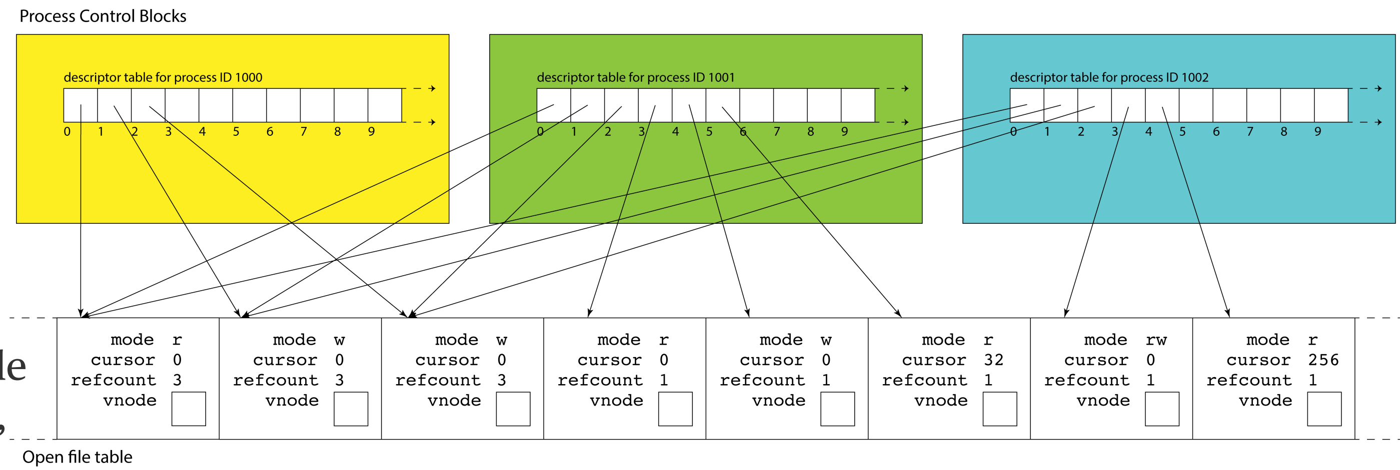
- The OS maintains a data structure for each active process. Those data structures are called **process control blocks**, and they are stored in a **process table**.
- Process control blocks store many things (the user who launched it, what time it was launched, CPU state, etc.). Among the many items it stores is the **descriptor table**.
- Each process maintains its own set of descriptors. Descriptors 0, 1, and 2 generally refer to standard input, standard output, and standard error, but there are no predefined meanings for descriptors 3 and up. Descriptors 0, 1, and 2 are most often bound to the terminal.
- A user program treats the descriptor as the identifier needed to interact with a resource (most often a file) via **`read`**, **`write`** and **`close`** calls. Internally, that descriptor is an index into the descriptor table.
- The process control block tracks which descriptors are in use and which ones aren't. When allocating a new descriptor for a process, the OS typically chooses the smallest available number in that process's descriptor table.

Lecture 04: Filesystem Data Structures



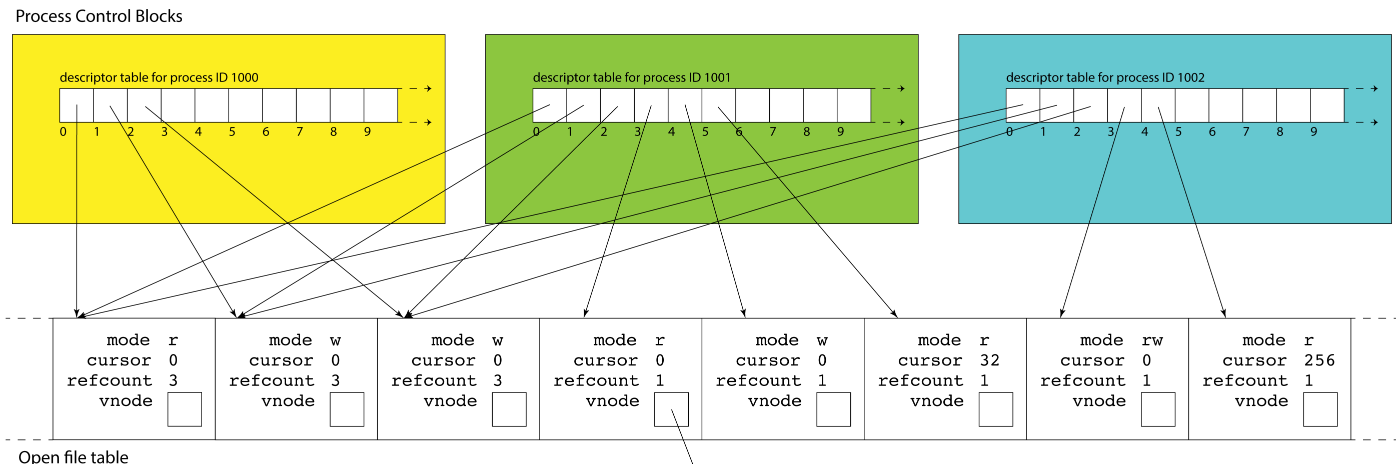
- If a descriptor table entry is in use, it maintains a link to an **open file table entry**. An open file table entry maintains information about an active session with a file (or something that behaves like a file, like terminal, or a network connection).
- Each table entry tracks information specific to the dynamics of that session. **`mode`** tracks whether we're reading, writing, or both. **`cursor`** tracks a position within the file payload. **`refcount`** tracks the number of descriptors across all processes that refer to that entry. (We'll discuss the **`vnode`** field in a moment.)
- The illustration here calls out one file table entry referenced by process 1001, descriptor 3. A call to **`open(filename, O_RDONLY)`** from that process might result in the above.

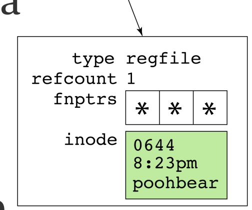
Lecture 04: Filesystem Data Structures



- At any one time, there are multiple active processes, each of which typically has at least three open descriptors, and possibly many more.
- Each process maintains its own descriptor table, but there is only one, system-wide open file table. This allows for file resources to be shared between processes, and we'll soon see just how common shared file resources really are.
- As drawn above, descriptors 0, 1, and 2 in each of the three PCBs alias the same three sessions. That's why each of the referred table entries have refcounts of 3 instead of 1.
  - This shouldn't surprise you. If your **`bash`** shell calls **`make`**, which itself calls **`g++`**, each of them inserts text into the same terminal window.

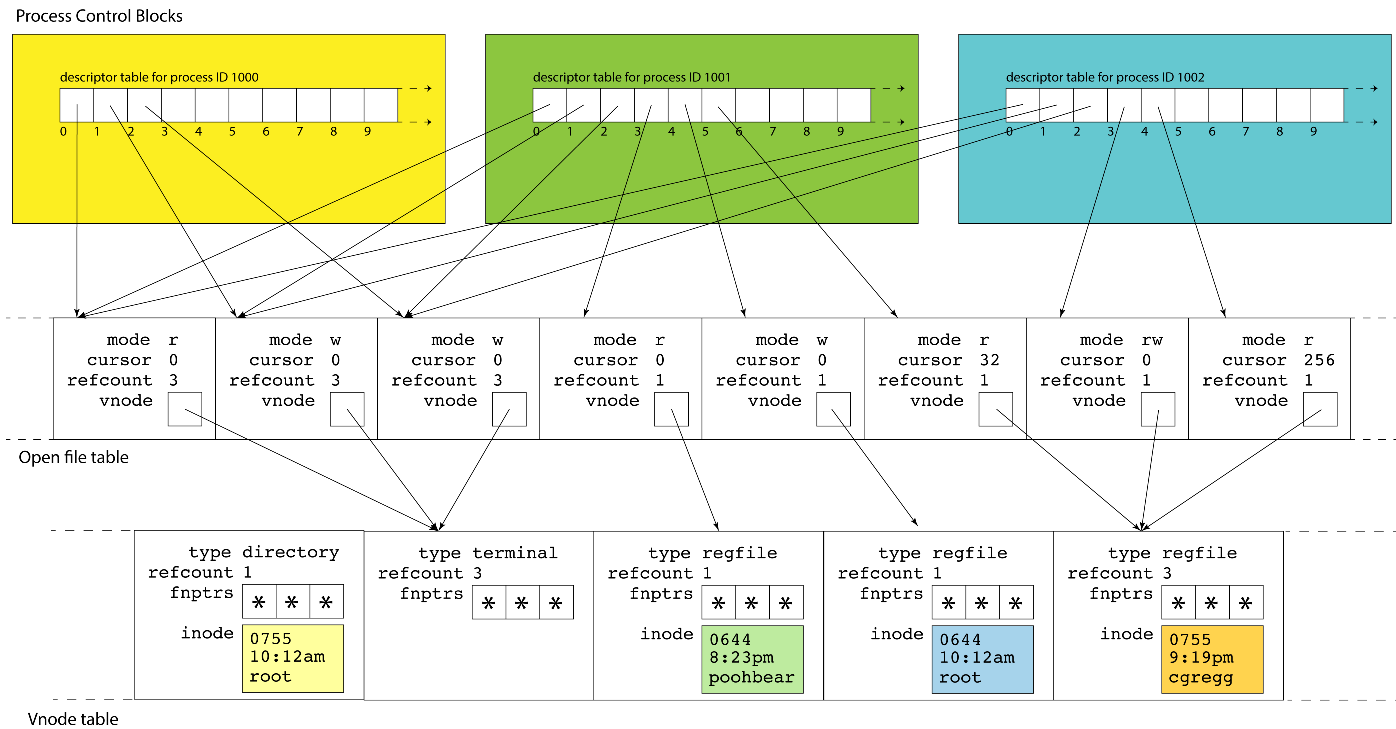
Lecture 04: Filesystem Data Structures





- Each of the open file entries maintains access to a vnode, which itself is a structure housing static information about a file or file-like resource.
- The data structure stores file type (e.g. regular file, directory, symlink, terminal), a refcount, the collection of function pointers that should be used to read, write, and otherwise interact with the resource, and, if applicable, a copy of the inode that resides on the filesystem on behalf of that file. In this sense, the vnode is an **inode cache** that stores information about the file (e.g. file size, owner, permissions, etc) so that it can be accessed more quickly.

Lecture 04: Filesystem Data Structures



- There is one, system-wide vnode table for the same reason there is one system-wide open file table. Independent file sessions reading from the same file don't need independent copies of the vnode. They can all alias the same one.

Lecture 04: Filesystem Data Structures

None of these kernel-resident data structures are visible to users. Note the filesystem itself is a completely different component, and that filesystem inodes of open files are loaded into vnode table entries. The yellow inode in the vnode is an in-memory replica of the yellow sliver of memory in the filesystem.