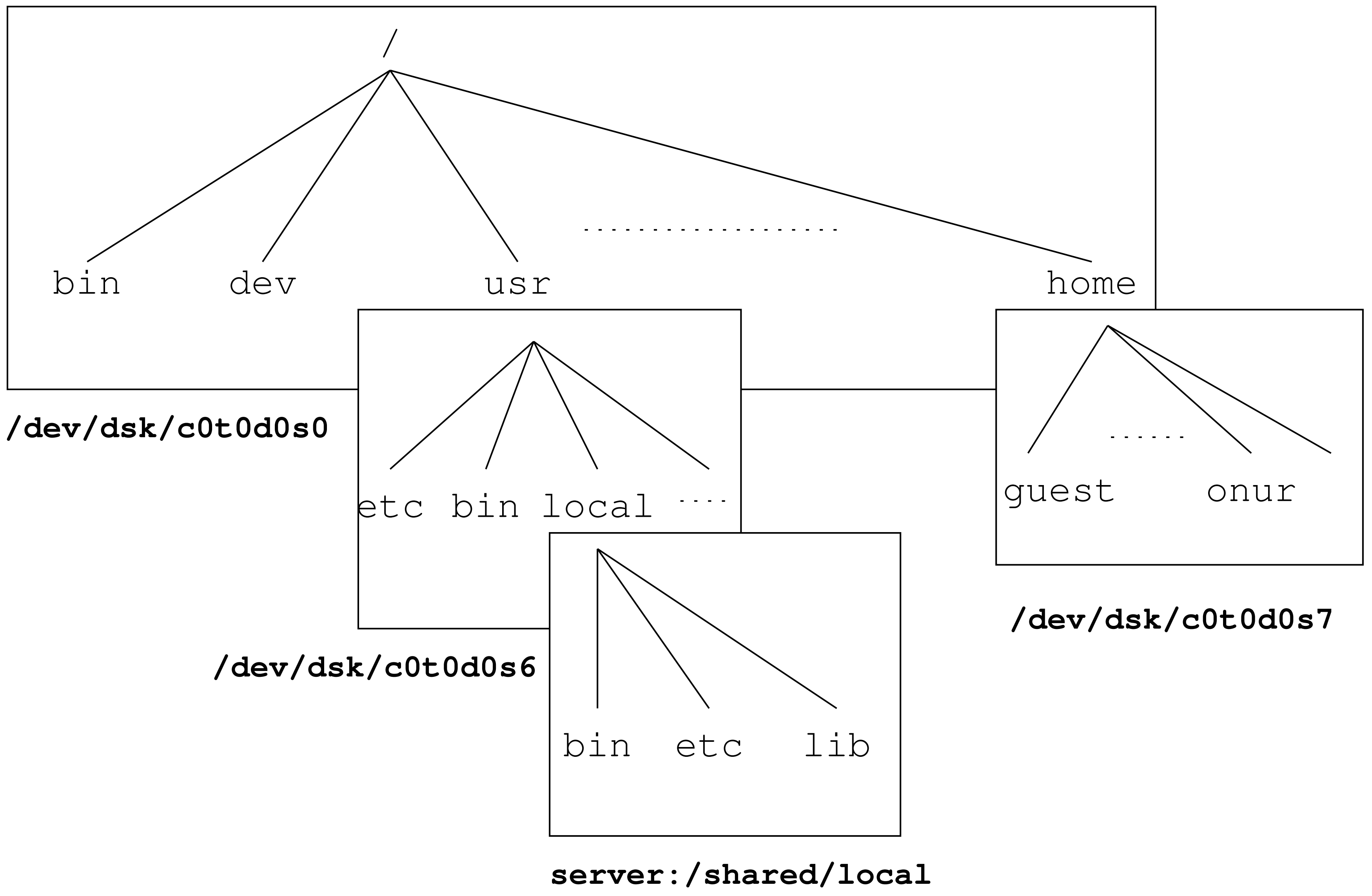
SYSTEM PROGRAMMING: FILES AND DIRECTORIES

Onur Tolga Şehitoğlu

METU, CEng 536 Lecture notes
Octobe, 2006

Filesystem

- Hierarchical structure of directories rooted by /
- A filesystem is a directory and file structure laying on the same logical entity (memory or disk partition, network resource).
- Different Unix based filesystems available like: *ufs, sys5fs, xfs, ext2, ext3,...*
- Other foreign filesystems are implemented: *dosfs, vfat, ntfs, iso9660,...*
- Network filesystems does not keep data structures but maps them to remote services: *nfs, rfs, smbfs,...*
- Memory based filesystems are kept on main memory: *tmpfs, ramfs,...*


/
bin
dev
usr
....
home
/dev/dsk/c0t0d0s0
etc
bin
local
....
/dev/dsk/c0t0d0s6
......
guest
onur
/dev/dsk/c0t0d0s7
bin
etc
lib
server:/shared/local

inodes

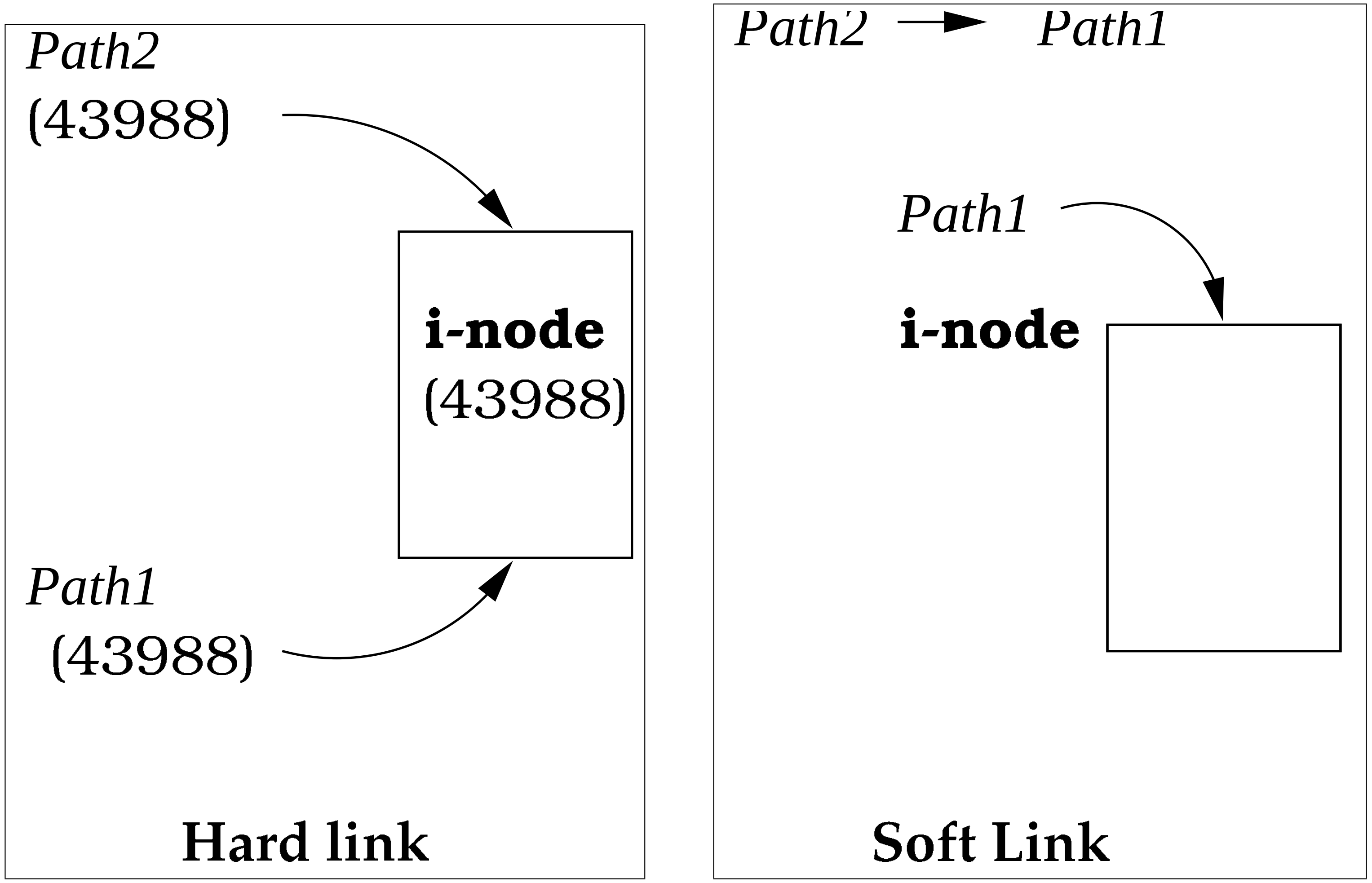
- *vnode* is the generic name for all fs types. Unix style fs's have i-nodes.
- For each file, there is a corresponding block keeping the vital information for accessing file called inode.
- inodes keep:
  filetype, permissions,number of hard links to file, owner uid, owner gid, size in bytes, access time, change time, modify time, addresses to data blocks
- What about filenames? They are inside data blocks of directories
- Directories are special typed filetypes including *filename/inode* pairs in data blocks.
- Data is stored in data blocks, inode of the file points to all data blocks of a file.

Accessing a File

- A file name like `/home/onur/test.c`
- Start from the root, "/" directory, access its i-node, get data block, load it.
- `home` exists, get i-node repeat above for that i-node.
- `onur` exists, get i-node and repeat above for that
- `test.c` exists, get i-node
- get data blocks, open file

Links

- Two type of links: *hard links* and *soft links* exist.
- *hard links*: Two directory entry can map to same i-node, thus same file. Transparent to user.
- *soft links*: A special file type, symbolic link exists. Data block contains a relative path. Kernel continue traversing by loading symbolic links. Not transparent, process can distinguish the link file and the original file.
- Hard links are faster but restricted: only can link a file in the same fs (inodes are unique to fs). Cyclic reference problems cannot be tracked. Only system administrator can link to a directory.
- Soft links can link file in another fs. They can dangle (original file can be deleted while link still exists). Cyclic references can be avoided by countint the maximum number of visited link.


Path2
(43988)
i-node
(43988)
Path1
(43988)
Hard link
Path2
Path1
Path1
i-node
Soft Link

Access rights

- There are 3 user classes: *owner user*, *owner group*, *others* and 3 access levels *read*, *write*, *execute*.
- The owner of the process trying to access the file is checked against these classes and permissions and permission is granted accordingly.
- Permissions for a regular file and directory differs:

	files	**directories**
`r`	read content	read content (file list) of directory
`w`	change content	add, move or delete file
`x`	execute file	access files under directory

- Octal notation. 12 bits of information. represented as integers:

$$\texttt{ugt}\ \overbrace{\_\ \_\ \_}^{\text{owner}}\ \overbrace{\_\ \_\ \_}^{\text{group}}\ \overbrace{\_\ \_\ \_}^{\text{others}}$$

```
r w x r w x r w x
```

- *setuid* bit: In an executable set effective user id of the owner user (not the actual user id of the process)
- *setgid* bit: In an executable set effective group id of the owner group (not the actual group id of the process)
- *sticky* bit: In an executable defines paging of the program memory for later use, it is somewhat obsolote. In a directory, permission to change (w) is granted only if the file to be changed is also writable.
- **ACL**: Access Control List:
  New implementations allow arbitrary (per user, per group) control of permissions (see setfacl and getfacl commands).