DATA STORAGE TECHNOLOGIES & NETWORKS

(CS C446, CS F446 & IS C446)

LECTURE 31 – STORAGE

NFS – Operation

- NFS protocol is stateless
  - Each request is independent
    - Server need not maintain information about clients, their requests and the files they have opened.
    - No need of state recovery after a client / server crash and reboot or after network failure
- Every RPC request a server receives is completely self contained
  - Server does not need any additional information to fulfill the request
- In practice, server caches recently accessed file data: Caching
  - improves performance
  - and is useful for detecting *retrials* of previously serviced requests
- Compatibility Issues:
  - Local file systems are statefull:
    - e.g. link removal, locking,
  - Transaction-Commit semantics implies high overhead:
    - All operations that modify the file system must be committed to stable storage before the RPC can be acknowledged
    - Synchronous writes [up to 3 synchronous writes] are required for *write* operations.

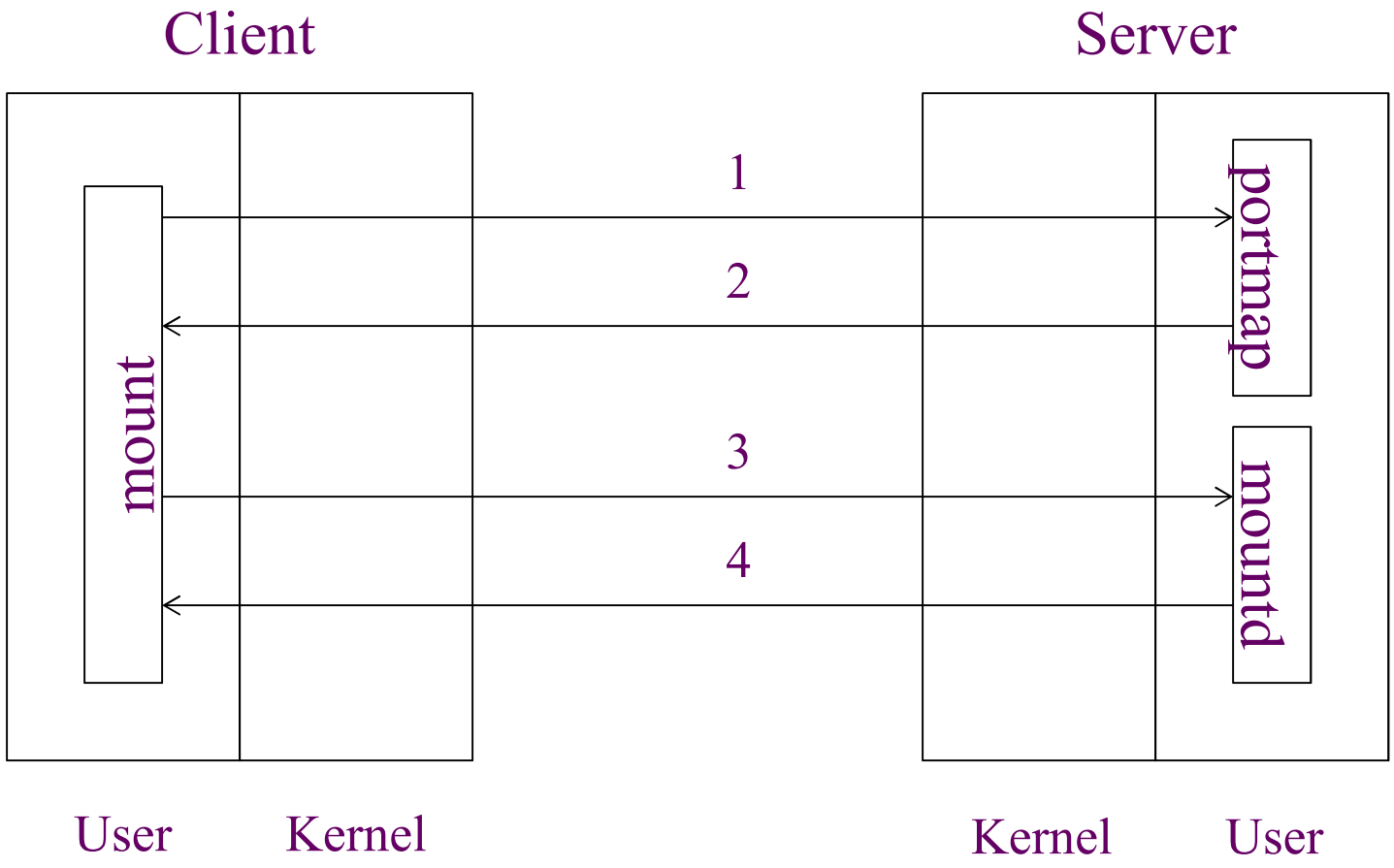
NFS – Operation

- NFS is a client-server protocol
  - No discovery or registry is used.
    - i.e. client must know the server and the filesystem
  - Server's filesystem has to be exported (by mounting).
    - Refer to *mount* system call on UNIX
    - Each server has a globally unique id (*guid*)
  - Client application need not distinguish between local and remote files
    - i.e. once a filesystem is mounted applications use the same interface for both types of files
- For server side to function portmap, mountd and nfsd daemons must be running

NFS – Operation

- NFS is a client-server protocol
  - *File manager* part of client file system remains the same as that for the local file system
    - Only the *file store* is local or remote
  - Each remote file that is *active* manifests as an *nfsnode* in the client's filesystem
    - Recall the *active file* data structures:
      - Per-Process File Descriptor
      - --> Kernel File Entry
      - --> *vnode*
      - --> file-store-specific data structure
        - *inode* for local file store and
        - *nfsnode* for remote filestore

- Portmap daemon
  - Acts as a registration service for programs that provide RPC based services
  - RPC daemon tells portmap daemon to what port number it will listen and what RPC service it will prepare to serve
- When a client wishes to make an RPC call to a given service
  - It will first contact the portmap daemon on the server machine to determine the port number to which RPC messages should be sent


Client
Server
mount
portmap
mountd
1
2
3
4
User
Kernel
Kernel
User

NFS – Operation

- Basic Protocol (Mounting): C and server S
  - C (mount process) -----> S(*portmap* daemon)
    - request port address of *mountd*
  - S (*portmap* daemon) ----> C
    - return the port address of its *mountd*
  - C (mount process) ----> S(*mountd* daemon)
    - request mounting of filesystem [pathname]
  - S (*mountd* daemon):
    - get file handle for desired mount point from kernel
  - S (*mountd* daemon) ---> C
    - return filehandle of mountpoint OR error

Portmap daemon has a well known port

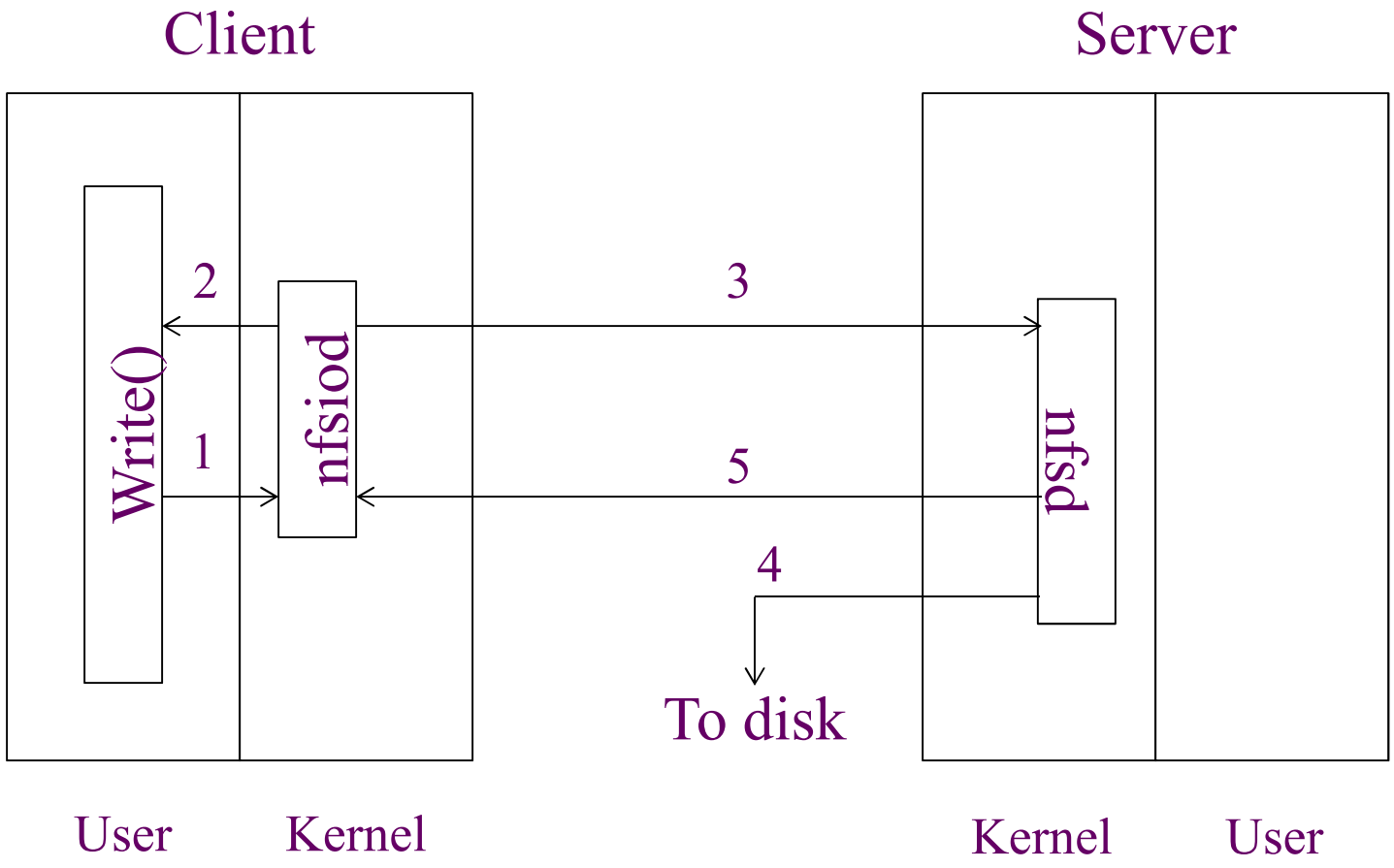
RPC parameter

- On startup
  - mountd reads the /etc/exports file and creates a list of hosts and networks to which each local file system may be exported
  - It passes the list into the kernel [mount system call]
  - Kernel links the list to the associated local file system mount structure
    - List will be readily available before the NFS request
- Client mount requests are directed to mountd
  - Mountd returns a file handle [if permission is satisfactory] for the requested mount point

NFS – Operation

- Basic Protocol (I/O Operation): Client C, Server S
  - C (app. process): *write* system call
    - Data is copied into kernel buffer of the client and call returns
    - *nfsiod* daemon wakes up
  - C (*nfsiod* daemon) -----> *S(nfs_rcv*
    - Request write [buffer contents]
  - S delegates request to *nfsd* daemon
    - *nfsd* daemon invokes write on disk and waits for I/O
  - S (*nfsd* daemon) ---> C (*nfsiod* daemon)
    - return *ack* for the *write* operation

RPC parameter


Client
Server
Write()
nfsiod
nfsd
1
2
3
4
5
To disk
User
Kernel
Kernel
User

- nfsd master daemon forks children that enter the kernel using nfssvc system call
  - Usually a system runs 4 to 6 nfsd daemons [this determines the maximum concurrency in server]
- When a request arrives on datagram [connectionless] or stream [connection oriented] socket, system invokes nfsrv_rcv()
  - nfsrv_rcv() takes message from the socket receive queue and dispatches that message to an available nfsd daemon
  - nfsd verifies the sender and passes the request to the appropriate local file system for processing

- For connection oriented [TCP] transport protocol – 1 connection for each client to server mount point
- For connection less [UDP] transport protocol – A fixed number of incoming RPC sockets when it starts its nfsd daemon
  - Clients create 1 socket for each imported mount point

- Clients can operate without any daemon running
  - System performance improves if several nfsiod daemons running
  - nfsiod daemon does asynchronous read aheads and write behinds
  - Starts when kernel runs multiuser [enters the kernel using nfssvc system call]

- Client machines of the NFS must have ways to handle processes that are accessing remote files [when client is running and server is unreachable / crash]
  - Hard mount that continue to try connecting to server forever [Default]
  - Soft mount that retries an RPC a specific number of times then returns transient error
  - Interruptible mount that waits forever [like hard] but checks to see whether termination signal is pending for any process that is waiting for a server response