User manual

Compiling and cleaning
From the top level project directory:

- `make` - builds all parts (P1, P2, P3, P4_P5)
- `make clean` - removes the compiled executables

P1 (cd p1)

- `./reverse_server`
- `./reverse_client <server_ip> <text...>`

P2 (cd p2)

- `./ls_server`
- `./ls_client <server_ip> <ls_args...>`

P3 (cd p3)

- `./disk_server <cylinders> <sectors> <track_delay_us> <backing_file>`
- `./disk_client <server_ip>`
- `./disk_rand <server_ip> <ops> <seed>`

P4_P5 (cd p4_p5)

- `./file_system_server <disk_server_ip>`
- `./file_system_client <fs_server_ip>`

Technical manual

Disk design (Part 3)
The disk server simulates a block device using one normal file as the “disk”.

Data layout:

- Cylinders (C) and sectors per cylinder (S) are passed args from the command line.
- Block size is 128 bytes. Total blocks = C × S.
- Each block is addressed by (cylinder c, sector s).
- Block number b = c × S + s, and byte offset in the backing file = b × 128.
- Every disk read/write is exactly 128 bytes at that offset.

In-memory state per client:

- `current_cylinder` – last cylinder accessed, used to simulate seek time.
- A line buffer for incoming text commands.
- A 128-byte buffer for one block of data.

Protocol and main algorithms:

For each TCP connection, the server loops: read one line, parse the command, check arguments, do the operation, send a reply, until the client closes.

Commands:

- `I`
  Client asks for geometry. Server replies with C and S on one line.
- `R c s` (read)
  Server checks that `c` and `s` are in range. If not, it returns an error.
  If valid, it computes how far the head moves (`abs(current_cylinder - c)`), sleeps for that distance times the track delay, updates `current_cylinder`, seeks to the correct offset in the backing file, reads up to 128 bytes (padding with zeros if needed), and sends a success flag plus 128 bytes of data.

- `W c s l` (write)
  Server checks `c`, `s`, and `l` (0 ≤ l ≤ 128). If invalid, it returns an error and ignores the write.
  If valid, it reads exactly `l` bytes of data from the client into a 128-byte buffer, fills the rest with zeros, reads a trailing newline, simulates seek time as in the read case, seeks to the correct offset, writes all 128 bytes, flushes the file, and then sends success or failure.

`disk_client` is an interactive client that sends these commands and prints the replies.
`disk_rand` gets C and S with `I` and then runs many random `R` and `W` requests to random blocks to test the disk.

Filesystem design (Parts 4 and 5)
The filesystem server builds a tiny filesystem on top of the disk server. It treats the disk as an array of 128-byte blocks.

Disk information:

- On startup it connects to the disk server, sends `I`, and uses C and S to compute total_blocks = C × S.
- Logical block numbers 0..total_blocks−1 are converted back to (c, s) when reading/writing via the disk protocol.

In-memory data structures:

- Free-space map:
  `block_used[]` with one entry per disk block. 0 = free, 1 = in use.
- File/directory table:
  Fixed-size array of entries. Each entry stores:
  - `used` (free/in use)
  - `is_dir` (file or directory)
  - `parent` (index of parent directory)
  - `name` (fixed-length string)
  - `first_block` (starting block index or −1 if no data)

    - `nblocks` (number of contiguous blocks)

    - `size` (file size in bytes)
      Entry 0 is the root directory “/” and has itself as parent.

- Current directory:
  `cwd_index` – index of the current directory entry.

Main algorithms:

- Format (`F`):
  Clear the entry table and free-space map, create root at entry 0, mark all blocks free, set `cwd_index` to root.

- Allocate / free blocks:
  To allocate k blocks, scan `block_used[]` for k consecutive zeros, mark them used, return the starting index.
  To free, given `first_block` and `nblocks`, mark that range as free.

- Create file (`C name`):
  Check that the name is not already used in the current directory, find a free entry, set it as a file with `parent = cwd_index`, no blocks, and size 0.

- Delete file (`D name`):
  Find the file in the current directory, free its blocks, and mark the entry unused.

- Write file (`W name len`):
  Find the file, read `len` bytes from the client, free any old blocks, compute how many 128-byte blocks are needed, allocate that many contiguous blocks, update `first_block`, `nblocks`, and `size`, and write the data into those blocks through the disk server.

- Read file (`R name`):
  Find the file, and if the size is > 0, read its blocks from disk and return exactly `size` bytes to the client.

- List directory (`L 0` / `L 1`):
  Scan entries with `used = 1` and `parent = cwd_index`. Print their names (and, in verbose mode, type and size).

- Directories (`mkdir`, `cd`, `pwd`, `rmdir`):
  `mkdir` adds a directory entry under the current directory.
  `cd /`, `cd ..`, `cd name` update `cwd_index` to root, parent, or a child directory.
  `pwd` follows parent indices back to root and builds a path.
  `rmdir` removes a directory only if it is empty.