System Architecture Overview

The project is a distributed file storage system implemented in C, consisting of four servers (s1.c, s2.c, s3.c, s4.c) and a client (w25clients.c):

- S1 (Port 9086): The primary server that handles .c files locally and forwards .pdf, .txt, and .zip files to S2, S3, and S4, respectively. It coordinates client requests and communicates with secondary servers.
- S2 (Port 9087): Manages .pdf files, supporting upload, download, removal, listing, and tar archiving.
- S3 (Port 9088): Manages .txt files, with similar functionality to S2.
- S4 (Port 9089): Manages .zip files, mirroring S2 and S3's operations.
- Client (w25clients.c): Connects to S1 to perform file operations: upload (uploadf), download (downlf), remove (removef), list files (dispfnames), and download tar archives (downltar).

Each server uses TCP sockets for reliable communication, stores files in designated directories (./S1, ./S2, ./S3, ./S4), and maintains a log file (S1.log, S2.log, etc.). The system is designed to mimic a distributed storage system, with S1 acting as a gateway and secondary servers specializing in specific file types.

Libraries Used Across All Files

- Standard Libraries:
  - <stdio.h>: For I/O operations (printf, fopen, fprintf, fclose).
  - <stdlib.h>: For memory management and exit (malloc, exit).
  - <string.h>: For string manipulation (strcmp, strncpy, snprintf, strstr).
- System and Networking:
  - <unistd.h>: For system calls (close, read, write, fork, usleep, access).
  - <sys/socket.h>: For socket operations (socket, bind, listen, accept, connect, send, setsockopt).
  - <netinet/in.h>: For Internet address structures (struct sockaddr_in).
  - <arpa/inet.h>: For IP address conversion (inet_pton, inet_ntop, htons).
  - <sys/stat.h>: For file and directory operations (stat, mkdir, fstat).
  - <fcntl.h>: For file control (open, O_WRONLY, O_CREAT, O_RDONLY).
  - <sys/wait.h>: For process management in S1 (waitpid).
  - <sys/time.h>: For socket timeouts (struct timeval).
- Directory and Time:
  - <dirent.h>: For directory operations (opendir, readdir, closedir).
  - <time.h>: For timestamps (time, ctime).
  - <errno.h>: For error details (strerror, errno).

Constants

- Ports: PORT_S1 (9086), PORT_S2 (9087), PORT_S3 (9088), PORT_S4 (9089).
- Buffer Size: BUFFER_SIZE (4096 bytes) for data transfers.
- Path Length: MAX_PATH (1024) for file paths.
- Directories: S1_DIR (./S1), S2_DIR (./S2), S3_DIR (./S3), S4_DIR (./S4).
- Log Files: S1.log, S2.log, S3.log, S4.log.
- Server IPs (in s1.c): S2_IP, S3_IP, S4_IP as 127.0.0.1 (localhost).
- Client: SERVER_IP (127.0.0.1), PORT_S1 (9086).

Function Explanations

Functions Common to s1.c, s2.c, s3.c, s4.c

These functions are nearly identical across servers, differing only in directory names (S1_DIR, S2_DIR, etc.) and file types in list_files and tar_files.

1. log_message(const char *msg)

- Purpose: Logs a message with a timestamp to the server's log file (S1.log, S2.log, etc.).
- Implementation:
  - Opens the log file in append mode using fopen(LOG_FILE, "a").
  - If fopen fails (e.g., disk full, permissions), silently returns to keep the server running, prioritizing robustness over log loss.
  - Gets the current time with time(&now) and converts to a string using ctime.
  - Removes the trailing newline from the time string by setting time_str[strlen(time_str) - 1] = '\0'.
  - Writes the message in the format [timestamp] msg\n using fprintf.
  - Closes the file with fclose.
  - Example log entry: [Mon Apr 14 10:00:00 2025] Server started.
- Libraries Used:
  - <stdio.h>: fopen, fprintf, fclose.
  - <time.h>: time, ctime.
  - <string.h>: strlen.
- Error Handling:
  - Silent failure on fopen to avoid crashing.
- Edge Cases:
  - Disk full: Log entry is skipped.
  - Long messages: Handled by fprintf, no truncation risk within typical log message lengths.
- Performance Note:
  - Opening/closing the file per log is I/O-intensive. Buffering or keeping the file open could optimize high-frequency logging (not implemented).
- Viva Tips:
  - Explain logging's role in debugging distributed systems, tracing S1's interactions with S2–S4.
  - Mention the silent failure trade-off: ensures server uptime but risks missing logs.
- Example:
  - Input: log_message("New connection").
  - Output in S1.log: [Mon Apr 14 10:00:01 2025] New connection.

2. receive_file(int socket, const char *filepath)

- Purpose: Receives a file from a socket and saves it to filepath.
- Implementation:
  - Sets a 5-second receive timeout using setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) with tv.tv_sec = 5.
  - Sends "READY" (5 bytes) using send(socket, "READY", 5, 0) to signal readiness.
  - Logs "Sent READY signal to client".
  - Reads the file size (off_t file_size) using read(socket, &file_size, sizeof(off_t)).
    - If read fails (e.g., connection closed, timeout), logs the error with snprintf and strerror, sends "ERROR: Size not received", and returns.

    - Opens the file using open(filepath, O_WRONLY | O_CREAT | O_TRUNC, 0666).
        - If open fails (e.g., no permissions), logs the error, sends "ERROR: File creation failed", and returns.
    - Reads data in chunks (BUFFER_SIZE = 4096) using read and writes to the file using write.
        - Tracks bytes received (received).
        - If read fails (e.g., client disconnects), logs and sends "ERROR: Read failed".
        - If write fails (e.g., disk full), logs and sends "ERROR: Write failed".
    - Closes the file with close(fd).
    - If received == file_size, sends "SUCCESS: File uploaded"; else, sends "ERROR: Incomplete transfer" and logs details.
    - Resets timeout to 0 using setsockopt.
    - Logs key steps: READY signal, file size received, success/failure.
- Libraries Used:
    - <sys/socket.h>: send, read, setsockopt.
    - <fcntl.h>: open, write, close.
    - <sys/time.h>: struct timeval.
    - <string.h>: snprintf.
    - <errno.h>: strerror.
- Error Handling:
    - Handles socket errors (timeout, disconnect), file open/write failures, and incomplete transfers.
    - Sends descriptive error messages to the client.
- Edge Cases:
    - Empty File: If file_size == 0, creates an empty file and sends "SUCCESS".
    - Large Files: Loops handle any size, as chunks are 4096 bytes.
    - Invalid Path: Assumes caller validates filepath (starts with S1_DIR).
- Security:
    - Relies on caller to ensure filepath is within server directory, preventing writes outside (e.g., /etc/passwd).
- Performance:
    - Buffer size (4096) balances memory and I/O efficiency, common for network transfers.
- Viva Tips:
    - Highlight the READY-size-data protocol as a handshake for reliable transfers.
    - Discuss why timeout is critical (prevents server hanging on slow clients).
    - Explain buffer size choice (standard for TCP transfers).
- Example:
    - Input: receive_file(sock, "./S1/dir/file.c").
    - Process: Receives size, saves data, sends "SUCCESS: File uploaded".

3. send_file(int socket, const char *filepath)

- Purpose: Sends a file over a socket to the client.
- Implementation:
    - Opens the file using open(filepath, O_RDONLY).
        - If open fails (e.g., file not found), logs "File open failed", sends file_size = -1, sends "ERROR: File not found", and returns.
    - Gets file size using fstat(fd, &st) and st.st_size.
    - Sends "READY" using send(socket, "READY", 5, 0).

    - Waits 10ms with usleep(10000) to ensure client readiness.
    - Sends file size (off_t) using send(socket, &file_size, sizeof(off_t), 0).
    - Reads file in chunks (BUFFER_SIZE) using read and sends using send.
        - Continues until read returns 0 (end of file).
    - Closes file with close(fd).
    - Logs "File sent successfully" on completion.
- Libraries Used:
    - <fcntl.h>: open, read, close.
    - <sys/stat.h>: fstat, struct stat.
    - <sys/socket.h>: send.
    - <unistd.h>: usleep.
- Error Handling:
    - Handles file open failure with clear client feedback.
    - Assumes send succeeds (TCP retries internally).
- Edge Cases:
    - Empty File: Sends file_size = 0, handled correctly by client.
    - File Deleted: Sends file_size = -1 and error message.
- Performance:
    - usleep delay is heuristic; could be tuned for network conditions.
- Viva Tips:
    - Explain TCP's reliability vs. UDP (no packet loss handling needed).
    - Discuss usleep as a sync mechanism, potential for removal with better client coordination.
- Example:
    - Input: send_file(sock, "./S1/dir/file.c").
    - Output: Sends size and data, logs success.

4. create_directory(const char *path)

- Purpose: Creates a directory and its parents recursively.
- Implementation:
    - Copies path to a temporary buffer using snprintf(tmp, MAX_PATH, "%s", path).
    - Starts parsing after base directory (S1_DIR, S2_DIR, etc.) using p = tmp + strlen(S1_DIR) + 1.
    - Iterates through tmp, finding slashes (/).
        - For each slash, null-terminates (*p = '\0'), calls mkdir(tmp, 0777), restores slash (*p = '/').
    - Creates final directory with mkdir(tmp, 0777).
    - Logs "Directory created or exists".
- Libraries Used:
    - <string.h>: snprintf, strlen.
    - <sys/stat.h>: mkdir.
- Error Handling:
    - mkdir fails silently if directory exists (EEXIST).
    - No explicit error for permission issues (logs success anyway).
- Edge Cases:
    - Invalid Path: Caller validates path starts with S1_DIR, preventing issues like ...
    - Empty Path: Not expected, as path includes S1_DIR.

- Security:
    - Ensures directories are within server scope by skipping S1_DIR.
- Viva Tips:
    - Explain recursive creation for nested paths (e.g., ./S1/dir/subdir).
    - Note silent error handling as a design choice for simplicity.
- Example:
    - Input: create_directory("./S1/dir/subdir").
    - Process: Creates ./S1/dir, then ./S1/dir/subdir.

5. list_files(const char *path, char *file_list, size_t list_size)

- Purpose: Lists files in a directory recursively, filtering by file type.
- Implementation:
    - Opens directory with opendir(path).
        - If fails (e.g., path invalid), sets file_list to "ERROR: Directory not found" and returns.
    - Iterates entries using readdir.
    - Skips . and .. with strcmp(entry->d_name, ".") == 0.
    - Builds full path with snprintf(full_path, MAX_PATH, "%s/%s", path, entry->d_name).
    - For regular files (entry->d_type == DT_REG):
        - S2: Checks .pdf with strstr(entry->d_name, ".pdf").
        - S3: Checks .txt.
        - S4: Checks .zip.
        - S1: No filter, includes all files.
        - Adds path (without S1_DIR) to file_list using snprintf(file_list + offset, list_size - offset, "%s\n", full_path + strlen(S1_DIR) + 1).
    - For directories (DT_DIR), recursively calls list_files.
    - Tracks offset to prevent buffer overflow.
    - If no files found, sets file_list to "No files found\n".
    - Closes directory with closedir.
- Libraries Used:
    - <dirent.h>: opendir, readdir, closedir, struct dirent.
    - <string.h>: snprintf, strcmp, strstr.
- Error Handling:
    - Handles invalid directories.
    - Truncates list if list_size exceeded.
- Edge Cases:
    - Empty Directory: Returns "No files found\n".
    - Buffer Overflow: Stops adding files, potentially truncating list.
    - Symbolic Links: Skipped, as DT_REG checks for regular files.
- Viva Tips:
    - Highlight server-specific filtering (e.g., S2 for .pdf only).
    - Discuss recursion for nested directories.
    - Mention buffer size limitation (4096 bytes).
- Example:
    - Input: list_files("./S2/dir", file_list, BUFFER_SIZE).
    - Output: file_list contains dir/file1.pdf\n.

6. tar_files(const char *filetype, const char *output_tar)

- Purpose: Creates a tar archive of files with a given extension.
- Implementation:
  - Builds command with snprintf:
    - Format: find %s -name "*.%s" -type f | tar -cvf %s -T - 2>/dev/null.
    - Example (S2): find ./S2 -name "*.pdf" -type f | tar -cvf pdf.tar -T - 2>/dev/null.
  - Executes command using system.
  - Checks if output_tar exists with access(output_tar, F_OK).
    - If fails (no files or error), creates empty file with fopen(output_tar, "w") to avoid send_file errors.
  - Logs "Tar created successfully" or "Tar creation failed or no files found".
- Libraries Used:
  - <string.h>: snprintf.
  - <stdlib.h>: system.
  - <unistd.h>: access.
  - <stdio.h>: fopen, fclose.
- Error Handling:
  - Suppresses tar errors with 2>/dev/null.
  - Creates empty file as fallback.
- Edge Cases:
  - No Files: Results in empty or missing tar, handled by creating empty file.
  - Large Files: tar handles any size, no code-level limit.
- Security:
  - Hardcoded filetype (pdf, txt, zip, c) prevents injection.
  - Operates within S_DIR, ensuring safety.
- Performance:
  - system call is heavyweight; in-memory tar could be faster.
- Viva Tips:
  - Explain why tar was used (standard tool, simplifies archiving).
  - Discuss empty file workaround for robustness.
  - Suggest alternatives like libtar for efficiency.
- Example:
  - Input: tar_files("pdf", "pdf.tar").
  - Output: Creates pdf.tar with .pdf files.

Functions Specific to s1.c

7**. send_to_server(const char *ip, int port, const char *command, char *response, size_t resp_size, int is_file_transfer, int client_socket)**

- Purpose: Sends a command to a secondary server (S2, S3, S4) and relays file data if needed.
- Implementation:
  - Creates socket with socket(AF_INET, SOCK_STREAM, 0).
    - If fails, sets response to "ERROR: Socket failed", logs, and returns.
  - Sets 5-second send/receive timeouts with setsockopt(SO_SNDTIMEO, SO_RCVTIMEO).
  - Sets server address (struct sockaddr_in) with serv_addr.sin_family = AF_INET, serv_addr.sin_port = htons(port), and inet_pton(AF_INET, ip, &serv_addr.sin_addr).
  - Connects using connect.
    - If fails, sets response to "ERROR: Connection failed", logs, and closes socket.
  - For uploadf, adjusts path:
    - Parses command with sscanf(command, "%*s %s %s", filename, dest_path).
    - Strips S1_DIR using snprintf(new_cmd, BUFFER_SIZE, "uploadf %s %s", filename, dest_path + strlen(S1_DIR) + 1).
    - Else, copies command to new_cmd.
  - Sends command with send(sock, new_cmd, strlen(new_cmd), 0).
  - If is_file_transfer (for uploadf):
    - Reads "READY" with read(sock, ready_signal, 5).
      - If not "READY", sets response to "ERROR: Server not ready", logs, and returns.
    - Reads file size from client_socket with read(client_socket, &file_size, sizeof(off_t)).
      - If fails, sets response to "ERROR: Failed to receive file size", logs, and returns.
    - Sends file size to server with send(sock, &file_size, sizeof(off_t), 0).
    - Relays data in chunks:
      - Reads from client_socket with read.
      - Sends to server with send.
      - Tracks received bytes.
      - If read or send fails, sets appropriate error in response and returns.
  - Reads server response with read(sock, response, resp_size - 1).
    - If valread_resp > 0, null-terminates response.
    - Else, sets "ERROR: No response from secondary server".
  - Closes socket with close(sock).
  - Logs command, READY signal, file transfer, and response.
- Libraries Used:
  - <sys/socket.h>: socket, connect, send, read, setsockopt.
  - <arpa/inet.h>: inet_pton, htons.
  - <sys/time.h>: struct timeval.
  - <string.h>: snprintf, strncpy, strcmp.
  - <errno.h>: strerror.
- Error Handling:
  - Handles socket creation, connection, READY signal, file transfer, and response failures.
- Edge Cases:
  - Invalid client_socket: Assumed valid by caller.

    - Server Down: Timeout triggers error response.
    - Empty File: Relays file_size = 0 correctly.
- Security:
    - Path adjustment ensures correct directory mapping.
- Viva Tips:
    - Emphasize S1's proxy role, hiding S2–S4 from clients.
    - Discuss timeouts for robustness.
    - Explain path adjustment for uploadf.
- Example:
    - Input: send_to_server("127.0.0.1", 9087, "uploadf file.pdf dir", response, BUFFER_SIZE, 1, client_sock).
    - Output: Forwards file to S2, sets response to "SUCCESS: File uploaded".

8. **validate_path(const char *path)**
- Purpose: Ensures a path starts with S1_DIR for security.
- Implementation:
    - Checks path != NULL and compares prefix with strncmp(path, S1_DIR, strlen(S1_DIR)) == 0.
    - Returns 1 if valid, 0 otherwise.
- Libraries Used:
    - <string.h>: strncmp, strlen.
- Error Handling:
    - Handles NULL path.
- Edge Cases:
    - NULL Path: Returns 0.
    - Short Path: If path is shorter than S1_DIR, returns 0.
- Security:
    - Prevents access outside ./S1 (e.g., /etc).
- Viva Tips:
    - Highlight as a simple security measure.
    - Discuss importance of sandboxing in file servers.
- Example:
    - Input: validate_path("./S1/dir/file.c") → 1.
    - Input: validate_path("./S2/dir") → 0.

9. prcclient(int client_socket)
- Purpose: Processes client commands in a child process.
- Implementation:
    - Loops to read commands with read(client_socket, buffer, BUFFER_SIZE - 1).
        - If valread <= 0 (disconnect), logs "Client disconnected" and breaks.
    - Null-terminates buffer: buffer[valread] = '\0'.
    - Parses with sscanf(buffer, "%s %s %s", command, arg1, arg2), stores number of args.
    - Logs command.
    - Handles commands:
        - uploadf (needs 3 args):
            - Validates arg2 with validate_path.
            - For .c files (strstr(arg1, ".c")):
                - Calls create_directory(arg2).
                - Calls receive_file(client_socket, filepath) with filepath = S1_DIR/arg2/arg1.

        - For others:
            - Sends "READY" to client.
            - For .pdf: Calls send_to_server(S2_IP, PORT_S2, buffer, response, BUFFER_SIZE, 1, client_socket).
            - For .txt: Uses S3.
            - For .zip: Uses S4.
            - Else: Sends "ERROR: Invalid file type".
            - Sends response to client.
    - downlf (needs 2 args):
        - Validates arg1.
        - For .c files: Calls send_file(client_socket, arg1).
        - For others:
            - Adjusts path (e.g., ./S1/dir/file.pdf → ./S2/dir/file.pdf).
            - Connects to server (S2, S3, or S4).
            - Sends downlf adjusted_path.
            - Checks "READY" and file_size.
            - Relays file to client with send(client_socket, "READY", 5, 0), sends size, and forwards data chunks.
            - Sends error if READY/size fails.
    - removef (needs 2 args):
        - Validates arg1.
        - For .c files: Calls remove(arg1), sends "SUCCESS: File removed" or "ERROR: Removal failed".
        - For others: Adjusts path, calls send_to_server with removef, sends response.
    - downltar (needs 2 args):
        - For c: Calls tar_files("c", "cfiles.tar"), sends file, removes tar.
        - For pdf, txt, zip:
            - Calls send_to_server to S2, S3, or S4.
            - If response is "READY", connects again, sends command, calls receive_file(client_socket, "pdf.tar") (or text.tar, zip.tar).
            - Else, sends response (e.g., "ERROR: No files to tar").
        - Else: Sends "ERROR: Invalid filetype".
    - dispfnames (needs 2 args):
        - Validates arg1.
        - Initializes file_list with "Files in directory:\n".
        - Calls list_files(arg1) for local files.
        - Queries S2, S3, S4 with adjusted paths (e.g., ./S2/dir).
        - Combines responses, ignoring errors.
        - If no files, sets "No files found\n".
        - Sends file_list to client.
    - Else: Sends "ERROR: Invalid command".
- Libraries Used:
    - <sys/socket.h>: read, send, socket, connect.
    - <string.h>: sscanf, strcmp, snprintf, strstr.
    - <arpa/inet.h>: inet_pton, htons.
    - <unistd.h>: usleep, close.
    - <stdlib.h>: system (via tar_files).

- Error Handling:
    - Invalid paths, file types, commands, and disconnects.
    - Server connection failures for forwarded commands.
- Edge Cases:
    - Partial Tar Download: Handled by TCP reliability.
    - Large File Lists: Limited by BUFFER_SIZE * 4, may truncate.
    - Invalid Args: Checks args count.
- Concurrency:
    - Runs in child process, allowing multiple clients.
- Viva Tips:
    - Explain S1's delegation model.
    - Discuss fork-based concurrency vs. threading.
    - Highlight unified client interface.
- Example:
    - Input: uploadf file.c ./S1/dir.
    - Output: Saves file, sends "SUCCESS: File uploaded".

10. main() in s1.c

- Purpose: Sets up and runs S1 server.
- Implementation:
    - Creates S1_DIR with mkdir(S1_DIR, 0777).
    - Initializes log file with fopen(LOG_FILE, "a"), closes it.
    - Creates socket with socket(AF_INET, SOCK_STREAM, 0).
        - If fails, logs and exits.
    - Sets SO_REUSEADDR with setsockopt to allow port reuse.
    - Binds to port 9086 with bind.
        - If fails, logs and exits.
    - Listens with listen(server_fd, 10).
        - If fails, logs and exits.
    - Logs and prints "S1 started, listening on port 9086".
    - Loops:
        - Accepts clients with accept.
            - If fails, logs and continues.
        - Logs client IP/port using inet_ntop.
        - Forks child process:
            - Child: Closes server_fd, calls prcclient(client_socket), exits.
            - Parent: Closes client_socket.
        - Cleans zombies with waitpid(-1, NULL, WNOHANG).
    - Closes server_fd (unreachable).
- Libraries Used:
    - <sys/socket.h>: socket, bind, listen, accept, setsockopt.
    - <arpa/inet.h>: inet_ntop, htons.
    - <sys/stat.h>: mkdir.
    - <stdio.h>: printf, fopen, fclose.
    - <sys/wait.h>: waitpid.
    - <unistd.h>: fork, close.
- Error Handling:
    - Socket setup and accept failures.
- Edge Cases:

    - Fork Failure: Not handled (rare, process limit).
    - Port in Use: SO_REUSEADDR mitigates.
- Viva Tips:
    - Discuss fork model for concurrency.
    - Explain SO_REUSEADDR for quick restarts.
    - Highlight zombie cleanup.
- Example:
    - Output: "S1 server listening on port 9086".
    - Process: Accepts clients, forks handlers.

Functions in s2.c, s3.c, s4.c

11. main() in s2.c, s3.c, s4.c

- Purpose: Sets up and runs S2, S3, or S4.
- Implementation:
    - Creates directory (S2_DIR, S3_DIR, S4_DIR) with mkdir.
    - Initializes log file.
    - Creates socket with socket(AF_INET, SOCK_STREAM, 0).
        - If fails, logs and exits.
    - Binds to port (9087, 9088, 9089).
        - If fails, logs and exits.
    - Listens with listen(server_fd, 10).
        - If fails, logs and exits.
    - Logs and prints "S2 listening on port 9087" (or S3/S4).
    - Loops:
        - Accepts connections with accept.
            - If fails, logs and continues.
        - Logs "New connection from S1".
        - Reads command with read(client_socket, buffer, BUFFER_SIZE - 1).
            - If valread <= 0, closes socket and continues.
        - Parses with sscanf(buffer, "%s %s %s", command, arg1, arg2).
        - Handles commands:
            - uploadf: Calls create_directory(dirpath) (S2_DIR/arg2), receive_file(client_socket, filepath) (S2_DIR/arg2/arg1).
            - downlf: Validates arg1 starts with S2_DIR, calls send_file.
            - removef: Validates arg1, calls remove, sends "SUCCESS" or "ERROR".
            - downltar: Calls tar_files("pdf", "pdf.tar") (or txt, zip), sends file, removes tar.
            - dispfnames: Calls list_files(arg1, file_list, BUFFER_SIZE), sends list.
        - Closes client_socket.
    - Closes server_fd (unreachable).
- Libraries Used:
    - <sys/socket.h>: socket, bind, listen, accept.
    - <arpa/inet.h>: htons.
    - <sys/stat.h>: mkdir.
    - <stdio.h>: printf, fopen, fclose.
    - <string.h>: sscanf, strcmp, snprintf.
- Error Handling:

    - Socket setup and command read failures.
- Edge Cases:
    - Buffer Overflow: Truncates at BUFFER_SIZE - 1, unlikely for commands.
    - Single Client: Handles one S1 request at a time.
- Viva Tips:
    - Note specialization (S2 for .pdf).
    - Discuss single-threaded design (sufficient for S1).
- Example:
    - Output: "S2 listening on port 9087".
    - Process: Handles .pdf requests.

Functions in w25clients.c

12. send_file(int socket, const char *filepath)
- Purpose: Sends a file to the server.
- Implementation:
    - Opens file with open(filepath, O_RDONLY).
        - If fails, prints error with strerror, sends file_size = -1, returns.
    - Gets size with fstat(fd, &st).
    - Sends size with send(socket, &file_size, sizeof(off_t), 0).
        - If fails, prints error and returns.
    - Reads chunks with read(fd, buffer, BUFFER_SIZE) and sends with send.
        - If read or send fails, prints error.
    - Closes file with close(fd).
    - Prints "File sent successfully".
- Libraries Used:
    - <fcntl.h>: open, read, close.
    - <sys/stat.h>: fstat.
    - <sys/socket.h>: send.
    - <stdio.h>: printf.
    - <errno.h>: strerror.
- Error Handling:
    - File open, read, and send failures.
- Edge Cases:
    - Empty File: Sends file_size = 0.
- Viva Tips:
    - Note protocol consistency with servers.
    - Discuss error feedback for usability.
- Example:
    - Input: send_file(sock, "file.c").
    - Output: "File sent successfully".

13. receive_file(int socket, const char *filename)
- Purpose: Receives a file and saves it locally.
- Implementation:
    - Reads "READY" with read(socket, ready_signal, 5).
        - If not "READY", reads error message, prints, and returns.
    - Reads size with read(socket, &file_size, sizeof(off_t)).
        - If fails, prints error and returns.

    - If file_size == -1, reads and prints error message, returns.
    - Opens file with open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0666).
        - If fails, prints error and returns.
    - Reads chunks with read and writes with write.
        - Tracks received bytes.
        - If write fails, prints error and returns.
    - Closes file with close(fd).
    - Prints "SUCCESS: File downloaded as filename" if received == file_size, else "ERROR: Incomplete file download".
- Libraries Used:
    - <sys/socket.h>: read.
    - <fcntl.h>: open, write, close.
    - <string.h>: strcmp.
    - <stdio.h>: printf.
    - <errno.h>: strerror.
- Error Handling:
    - READY, size, and write failures.
- Edge Cases:
    - Empty File: Handled correctly (file_size = 0).
- Viva Tips:
    - Highlight transfer verification.
    - Discuss error messages for user clarity.
- Example:
    - Input: receive_file(sock, "file.c").
    - Output: Saves file, prints "SUCCESS: File downloaded as file.c".

14. read_full_response(int socket, char *buffer, size_t size)
    - Purpose: Reads a response until newline or buffer limit.
    - Implementation:
        - Reads with read(socket, buffer + total_read, size - total_read - 1).
        - Loops until newline (strchr(buffer, '\n')), buffer full, or no data.
        - Null-terminates buffer.
        - Returns bytes read.
    - Libraries Used:
        - <sys/socket.h>: read.
        - <string.h>: strchr.
    - Error Handling:
        - Handles no data (timeout).
    - Edge Cases:
        - No Newline: Reads until size - 1.
        - Empty Response: Returns 0.
    - Viva Tips:
        - Note use for text responses (dispfnames, errors).
        - Contrast with receive_file for binary data.
    - Example:
        - Input: read_full_response(sock, buffer, BUFFER_SIZE).
        - Output: buffer has "SUCCESS: File removed\n".
15. main() in w25clients.c
    - Purpose: Runs client interface.

- Implementation:
    - Prints "Welcome to w25clients".
    - Loops:
        - Prompts "w25clients$".
        - Reads command with fgets(command, BUFFER_SIZE, stdin).
            - Removes newline with strcspn(command, "\n").
            - Breaks on EOF.
        - Creates socket with socket(AF_INET, SOCK_STREAM, 0).
        - Sets 5-second timeouts with setsockopt.
        - Connects to S1 (127.0.0.1:9086).
        - Parses command with sscanf(command, "%s %s %s", cmd, arg1, arg2).
        - Handles commands:
            - uploadf (3 args):
                - Checks file exists with stat(arg1, &st).
                - Sends command.
                - Waits for "READY".
                - Calls send_file(arg1).
                - Reads and prints response.
            - downlf (2 args):
                - Sends command.
                - Extracts filename with strrchr(arg1, '/').
                - Calls receive_file(socket, filename).
            - removef (2 args):
                - Sends command.
                - Reads and prints response.
            - downltar (2 args):
                - Validates filetype (c, pdf, txt, zip).
                - Sends command.
                - Sets filename (cfiles.tar, pdf.tar, etc.).
                - Calls receive_file.
                - Verifies file with stat, prints size.
            - dispfnames (2 args):
                - Sends command.
                - Reads and prints response.
            - exit: Closes socket, prints "Exiting client", breaks.
            - Else: Prints help message with valid commands.
        - Closes socket.
- Libraries Used:
    - <stdio.h>: printf, fgets.
    - <string.h>: sscanf, strcmp, strrchr, strcspn.
    - <sys/socket.h>: socket, connect, send, read, setsockopt.
    - <arpa/inet.h>: inet_pton, htons.
    - <sys/stat.h>: stat.
    - <sys/time.h>: struct timeval.
    - <errno.h>: strerror.
- Error Handling:
    - Socket, connection, file, and command errors.
- Edge Cases:

    - Invalid Filetype: Prints error for downltar.
    - EOF Input: Exits gracefully.
- Viva Tips:
    - Highlight user-friendly interface.
    - Discuss abstraction of distributed system.
- Example:
    - Input: uploadf file.c ./S1/dir.
    - Output: Sends file, prints "SUCCESS: File uploaded".

Viva Preparation and Additional Insights

Key Points to Emphasize

1. Distributed Design:
    - S1 coordinates, S2–S4 specialize, mimicking cloud storage (e.g., AWS S3).
    - S1's proxy role simplifies client interactions.
2. Socket Programming:
    - Uses TCP (SOCK_STREAM, AF_INET) for reliability.
    - S1 forks for concurrency; S2–S4 are single-threaded for simplicity.
3. File Operations:
    - READY-size-data protocol ensures robust transfers.
    - Hierarchical directories (./S1/dir) support organization.
    - Tar archiving uses system for simplicity.
4. Security:
    - Path validation prevents unauthorized access.
    - Hardcoded filetypes avoid command injection.
5. Robustness:
    - 5-second timeouts prevent hangs.
    - Detailed error messages aid debugging.
    - Logs capture all actions for tracing.
6. Scalability:
    - S1's forking supports multiple clients but has memory overhead.
    - S2–S4 could be scaled with threading if direct client access is added.

Potential Viva Questions

- Q: How does S1 route files?
    - A: Checks file extension in prcclient: .c stays local, .pdf to S2, .txt to S3, .zip to S4, using send_to_server.
- Q: What if a server is down?
    - A: send_to_server times out after 5 seconds, sends "ERROR: Connection failed" to client, logged for debugging.
- Q: Why use system for tar?
    - A: Simplifies archiving vs. parsing tar format manually. Trade-off is performance; libtar could improve efficiency.
- Q: How are large files handled?
    - A: Chunked transfers (4096 bytes) in send_file/receive_file handle any size, limited only by disk.
- Q: Security measures?
    - A: validate_path ensures files stay in server directories. Commands are parsed to prevent misuse.

Improvements to Suggest

- Use threading instead of forking in S1 for lower memory use.
- Buffer logs or keep file open for efficiency.
- Dynamic buffer sizes for list_files to avoid truncation.
- Add authentication for client-server interactions.