Software Design Document

Back-up and Recovery for the HTTP server

Authors:

- Scott Melero
- Korbie Sevilla

Sources:

- Linux man pages

Introduction

Goals and objectives

- The goals for Assignment 3 are to modify the HTTP server to implement two additional features: backups and recovery. The HTTP server will be able to store a backup of all the files in the server and have the ability to recover to an earlier backup.

Requirements

- Multi-threading and Redundancy will NOT be tested, so we can utilize the base code from Asgn1.
- All source files MUST maintain a .cpp extension.
- Utilize the sys-calls from the previous assignments as necessary.
- We must use the the following special requests GET /b, GET /r, and GET /l
    - GET /b: Creates a new folder “./backup-[timestamp]”. The content of this new backup folder will be a copy of all the files(NOT FOLDERS) in the current folder.
    - GET /r: The HTTP server will restore the most recent backup. All files from the most recent backup folder will be copied to the current folder.
    - GET /l: The HTTP server will return a list of timestamps of all available backups. (There is a new line after each timestamp)

Program Structure

- This project was built on the server designed here. It is a single threaded server that supports basic GET and PUT operations as well as backup, recovery, and list. We do not re-explain the design of the base server, and only write about what we added to support the new features.

Data Design

- We have added a new field to the http object struct: char timestamp[], which holds the list of time backups that we need to send back to the client after a list request.

Pseudocode Outline

- Processing a special request:
    - In the request processing module, void process_request(), if we see that we have a GET request, we need to then check the sent filename to see if it is a special request (b, r, l).
    - Compare the filename string against the request indicators, and execute the proper function.
    - If not a special request, then check to see if the filename is 10 characters in length. If it is, then GET the file as normal. If not, then return 400 bad requests.
- Backup:
    - When parsing the request (curl http://localhost:8080/b, GET b) go to peform_backup().
    - Setup a folder name string that is in the form “./backup-[timestamp]”. USE mkdir() call to create the new folder.
    - Use the opendir() command, and scan all of the file in the servers home directory, skip past all of the folders.
    - Iterate through each file in the working directory, and skip past any that return an error as we will not back those up. If all the files cannot open, we will still have an empty backup folder, so return 200. For each good file:
        - Create a file path string that is in the form “./backup-[timestamp]/filename”
        - Use open() to create a new file in the backup folder us the file path char.

        - In a loop, write() the contents of the original file descriptor into the backup file descriptor. After the loop finished, close both files and repeat until all of the files have been backup up
        - Return 200.
- Recovery:
    - When parsing the request (curl http://localhost:8080/r, or curl http://localhost:8080/r/timestamp) go to perform_recovery().
    - If /r, then recover the most recent backup
        - Setup a folder name string that is in the form "./backup-%d".
        - Use opendir to iterate through each of the backup folders in the working directory. Skip all of the files.
        - Initialize a new int most_recent = 0, which will keep track of the most recent timestamp.
        - For each file:
            - Use strtok and strcmp to ensure that the current folder we are reading is a backup folder with a proper unix timestamp.
            - Skip past non backup folders
            - Use the second element returned by strtok get the current timestamp.
            - Compare the current timestamp to the most_recent int. If the current timestamp is larger than the most recent timestamp, then set the current timestamp equal to the most_recent.
        - Close the directory.
    - If /r/timestamp.
        - When we are reading the http request, and parsing the filename, we need to use strncmp to check if the first 3 bits of the filename are in the form "/r/". This means that the client has requested a specific timestamp to recover.
        - Use strtok to split the string. Set the filename to r, so the processing module can recognize the request, and put the following int past "r/" into the timestamp char we added to the http object struct,
        - Use the given atoi(timestamp) as the value to format the folder string we defined earlier.
    - After we have our timestamp, and a formatted folder to open, opendir() again.
    - Iterate through each file in the backup directory, and skip past any that return an error as we will not recover those up. For each file:
        - Open the backup file.

        - open/create and truncate the original file in the working directory.
        - In a loop, write() the contents of the backup file descriptor into the original file descriptor. After the loop finished, close both files and repeat until all of the files have been recovered.
        - Return 200.
- List:
    - When parsing the request (curl http://localhost:8080l) go to list_backups().
    - opendir() on the working directory. For each folder in the directory that is a backup folder:
        - Use strtok to split the folder name at the "-". Take the second element returned by strtok and check to see if it is a valid timestamp.
        - If it is, append it value to the char array timestamp[] that we added to our http object struct.
    - Once we've read each of the folders, we will have compiled all of the available timestamps into a single list that we can print to the client socket.
    - Set the status message to 200. In the construct_http_response() function, after we have sent the status to curl, then we need to check if (status == 2 && filename == l). This means that we successfully compiled the list of timestamps, and will send() the timestamp[] array to the client socket.

System Components

- Void perform_backup()
    - Takes in a pointer to the http object struct that we use to hold information about the request and response.
    - This function will perform a backup of the HTTP server data, which in this instance, is all files that can be accessed via a GET request.
    - The function begins by opening the original file(s) and creating the backup by copying the contents of the original into the backup.
    - As the backup of the file(s) occurs, a new folder will be created to store the copied file(s), called "./backup-[timestamp] where the timestamp is the exact time in seconds upon its creation.
- Void perform_recovery()
    - Takes in a pointer to the http object struct that we use to hold information about the request and response.
    - Process a system recovery.

    - This function will recover the file state that is specified by the client, or the most recent backup if no timestamp was given.
    - Open the original file and backup file, then write the contents of the backup file into the original. The backup overwrites any existing data on files that exist. If the file does not exist, then create it. DOES NOT remove any files that are not present in the backup.
- Void list_bakups()
    - Takes in a pointer to the http object struct that we use to hold information about the request and response.
    - This function gets the timestamps associated with each backup folder, and put them into a formatted list to send back to the client.
- Void process_request()
    - Takes in a pointer to the http object struct that we use to hold information about the request and response as well as the client socket.
    - This function is structured in the same way as was designed for asg1 and asg2. We added additional conditionals to check for the special requests if we see we have a get request.

Description of the User Interface

- Server terminal:
    - The user will need to interact with this program through a terminal by establishing a server connection via "./httpserver address port# ". The address must be specified, and an error will be returned if the given arg is not valid.
    - The port number is optional. If it is not defined, the server will default to port 80. You need to execute the code as a super user in order to run on port 80 "sudo ./httpserver address port# "
- Client terminal:
    - Once a server is established, the use will then be able to request a file/application from the server. All files names **MUST** be 10 character ascii strings containing only the char's [0-9] & [a/A - b/B] (unless the client requests one of the special operations b, r, or l). To perform an op on the server, the client can run the following for their requests:'
        - Standard GET & PUT:
            - curl -T file http://localhost:8080/filename, to PUT a file onto the server directory

            - curl http://localhost:8080/filename, to GET a file that is in the server directory.
        - Special ops:
            - Curl http://localhost:8080/b, to create a backup of the current working directory (Files only).
            - Curl http://localhost:8080/r, to recover the servers most recent backup.
            - Curl http://localhost:8080/r/timestamp, to recover the backup at the specified timestamp
            - Curl http://localhost:8080/l, to create a list of timestamps of all available backups. Each timestamp will be separated by a new line.
    - The server will send back a response to the client terminal containing a status code and content length.

Testing

Testing Issues

- No testing issues as far as we can tell.

Classes of tests

- Backup
    - GET/b with only the binary in that folder
    - GET/b with many files in that folder (+30)
    - GET/b with half of the files that have permission, half files with no permission
    - GET/b with all files having no permission
    - GET/b with no files (don't think this is possible but its an idea worth mentioning)
    - Recovery
    - GET/r on valid recovery folder
    - GET/r where no recovery folder exists
    - GET/r on recovery folder with no permissions
    - GET/r on valid recovery folder but files inside have no permission
    - GET/r on an empty recovery folder (do we erase all the items in the server folder then?)

    - GET/r on a valid recovery folder where the files already exist in the server folder. (overwrite the server folder files)
    - GET/r where the files in the recovery folder have full permissions but the same files already exist in the server folder and have no permission
- Specific Recovery Folder
    - GET/r/backup-number on valid recovery folder
    - GET/r/backup-number on recovery folder that doesn't exist
    - GET/r/backup-number on recovery folder with no permissions
    - GET/r/ on valid recovery folder but files inside have no permission
    - GET/r/backup-number where the recovery folder is empty (do we erase all the items in the server folder then?)
    - GET/r/ with no backup folder specified
    - GET/r/backup-number on a valid recovery folder where the files already exist in the server folder. (overwrite the server folder files)
    - GET/r/ where the files in the recovery folder have full permissions but the same files already exist in the server folder and have no permission
- List
    - GET/l on a single backup folder
    - GET/l on many backup folders (+30)
    - GET/l where no backup exists
    - GET/l on a many backup folders that have no permission

Expected software response

- Backup
    - GET/b with only the binary in that folder
        - 200. Creates the backup folders
    - GET/b with many files in that folder (+30)
        - 200. Creates the backup folder with all of the files
    - GET/b with half of the files that have permission, half files with no permission
        - 200. Creates the folder, and skips past any files that we have problems opening.
    - GET/b with all files having no permission
        - 200. Creates an empty folder
    - GET/b with no files (don't think this is possible but its an idea worth mentioning)
        - 200. Creates an empty folder
- Recovery

    - GET/r on valid recovery folder
        - 200. Resteroes the state of the most recent backup
    - GET/r where no recovery folder exists
        - 200. nothing happens to the directory since it is up to date.
    - GET/r on recovery folder with no permissions
        - 403. Our code does not consider the backups which we cannot open.
    - GET/r on valid recovery folder but files inside have no permission
        - 200. Skips past the files that we cannot recover and restores the ones we can
    - GET/r on an empty recovery folder
        - 200. Does not erase anything, nothing happens.
    - GET/r on a valid recovery folder where the files already exist in the server folder. (overwrite the server folder files)
        - 200. Overwrites each of the pre-existing files on the server.
    - GET/r where the files in the recovery folder have full permissions but the same files already exist in the server folder and have no permission
        - 200. Skips past any files that we cannot get from the original server and return an error?
- Specific Recovery Folder
    - GET/r/backup-number on valid recovery folder
        - 200. Successfully recovers the specified backup
    - GET/r/backup-number on recovery folder that doesn't exist
        - 404. Returns a 404 to the client
    - GET/r/backup-number on recovery folder with no permissions
        - 500. Returns a 500 to the client?
    - GET/r/ on valid recovery folder but files inside have no permission
        - 200. Skips past all of the files we cannot open.
    - GET/r/backup-number where the recovery folder is empty (do we erase all the items in the server folder then?)
        - 200. Does not remove any files.
    - GET/r/ with no backup folder specified
        - 200. Will just return the most recent files.
    - GET/r/backup-number on a valid recovery folder where the files already exist in the server folder. (overwrite the server folder files)
        - 200. Overwrites any of the pre-existing files.
    - GET/r/ where the files in the recovery folder have full permissions but the same files already exist in the server folder and have no permission
        - 200. Skips past any of the files that we cannot open.

- List
    - GET/l on a single backup folder
        - 200. Return one timestamp
    - GET/l on many backup folders (+30)
        - 200. Returns all of the valid timestamps available
    - GET/l where no backup exists
        - 200. Returns nothing
    - GET/l on a many backup folders that have no permission
        - 200. Skip past any folder we cannot open as it should not be available to recover.

Performance bounds

- This is a single threaded server and can only service one client request at a time.