Project: interactive backup management system

As an old IT proverb says: „There are those who make backups and those who will", but manually creating such backups can be time-consuming and not very interesting. Therefore, we will create a system that automatically manages backups on the computer.

After starting, the program prints the list of available commands and waits for the user's instruction. Entering an incorrect command results in an appropriate error message but does not terminate the program.

Creating a backup

The `add <source path> <target path>` command starts creating a backup of the `source path` folder at the location indicated by `target path`. You can provide multiple target paths, separated by a space. In that case, backups will be created in several locations.

If any of the specified target directories do not exist, the program should create them. If, however, the target directory already exists, it must be empty before copying begins. Otherwise, the program should report an error.

Creating a backup consists in copying the entire directory structure and files from the source folder to all indicated target directories. The program should handle both regular files and symlinks.

If a symbolic link in the source directory contains an absolute path leading to this directory, the backup should create a symlink pointing to the relevant file in the target directory. In all other cases, the path in the link should be copied unchanged.

After completing the backup, the program starts monitoring the source directory. All changes - including files, symbolic links, and subfolders - are immediately mirrored in the target directories.

Monitoring ends when the user finishes the selected backup, closes the program, or when the source directory is deleted.

It is crucial to maintain high program responsiveness - the user may enter subsequent commands immediately after confirming the previous one, regardless of ongoing copying operations. Each target directory is managed by an independent subprocess, providing parallel execution and no lag in command processing.

The program should prevent:

- creating a backup of a directory inside itself (otherwise, this would result in an endless cascade of file creation),
- creating multiple backups with the same source and target paths.

Ending backup creation

The `end <source path> <target paths>` command stops backup in the given target directories. Their contents remain intact.

Listing active backups

The `list` command outputs a summary to standard output, showing which folders have backups and where they are saved.

Restoring backups

The `restore <source path> <target path>` command restores the selected backup. The program copies the entire directory structure and files from the target folder back to the source, and also deletes from the source those files not present in the backup. To speed up the operation, only files that have changed since the backup was created are copied. The command accepts only one target path.

Remember to adjust the path as necessary when copying symbolic links.

The command should work in blocking mode - subsequent commands can be issued only after restoring the backup.

Program exit

The `exit` command terminates the program. The program must free used resources and terminate all child processes. The program should also exit correctly in response to `SIGINT` and `SIGTERM`. Other signals should be ignored.

Notes and recommendations

- Do not use the `system` function.
- Directory names may contain spaces, so quote handling analogous to bash is required.
- To monitor changes in the source folder, you can use the `inotify` API. More information can be found in the following man pages:
  - `man 7 inotify`
  - `man 2 inotify_init`
  - `man 2 inotify_add_watch`
  - `man 2 inotify_rm_watch`
- The `realpath` function is useful for comparing paths.

Scoring

The project will be assessed primarily based on fully functioning features as specified in the table below. The number of points can be reduced due to:

- Resource leaks, lack of error handling for system functions, and other issues potentially leading to incorrect program behavior
- Code style - e.g., unnecessary global variables, very long functions, so-called "magic numbers," etc.
- Inability of the student to explain how their program works

Pay particular attention to the last point - the work must remain independent. If the student cannot answer questions regarding their own code, points may be zeroed.

1. [1 pts] A backup to a single target folder without further monitoring and synchronization is possible.
2. [4 pts] Waiting for changes and mirroring them.
3. [3 pts] The ability to add multiple target directories.
4. [1 pts] Ending backup creation.
5. [1 pts] Listing active backups.
6. [2 pts] Restoring backups (for copying all files, a maximum of one point can be received).

The `exit` command is required in every program.