Unix Midterm-2

Team19 108072244 邱煒甯 108032053 陳凱揚 112065525 簡佩如

Usage

```
make

./q1
#
# Sample output:
# tm_sec: 55
# tm_sec: 55
# tm_sec: 56

./q2 {your_path}
#
# File tree:
# test
# ├── as.txt -> a.txt
# ├── a.txt
# ├── home -> /home
# ├── tmp1
# │   ├── as.txt -> a.txt
# │   └── a.txt
# └── tmp2 -> tmp1
#
# Sample output:
# a.txt
# /home
# a.txt
# tmp1

./q3
#
# Sample output:
# 23:45:14, Thursday December 07, 2023

./q4
#
# Sample output:
# 1
# 2
# 3
# 4
# 5
# 6
```

Q1

Start second (program start): 55

End second (after running 3hours): 0

We observe that the program starts at 55 (second), then it slowly drifts away (55…56…0…20…40…0). This is because the sleep(10) function does not guarantee exactly 10 seconds of sleep. The actual sleep time might be slightly longer due to the time taken by the system to schedule and execute the program's process after waking up.

For a daemon like `cron`, it is supposed to run tasks at precise time. Thus, it cannot rely on a function like `sleep` for accurate timing. Instead, `cron` would likely use system call to check the time at regular interval and adjust the sleep time dynamically to make sure that it wakes up exactly at the same second(with previous minutes) in a new minute. Also, for the drift issue, `cron` would use system clock but not accumulated time from repeated sleep intervals to recorrect the time.

Q2

- Implement `printSymbolicLinks(const char *path)` to print all content of symbolic link under the given path and call the function recursively if the entry is also a directory.
- In the `printSymbolicLinks` function
  - Use `opendir` to open the directory.
  - Use `readdir` to get the entries one by one.
  - Use `lstat` to get the status of the entry.
  - Use `S_ISLNK` check if the entry is a symbolic link, then use `readlink` to get the link.

Q3

- Use `localtime_r` to fetch the current time.
- Use `strftime` with `"%H:%M:%S, %A %B %d, %Y"` to format the output of time.

Q4

- Use `malloc` to dynamically allocate memory for new nodes.

- Implement `delete_node` function and use `free` to release the dynamic memory allocated by `malloc`.
- In the end of the `main` function, we use `delete_node` to release the nodes one by one.