Assignment11

- Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <syslog.h>
#include <fcntl.h>
#include <sys/resource.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>

#define err_quit(fmt, ...)                               \
    do                                                   \
    {                                                    \
        fprintf(stderr, fmt "\n", ##__VA_ARGS__); \
        exit(1);                                         \
    } while (0)

void daemonize(const char *cmd)
{
    int i, fd0, fd1, fd2;
    pid_t pid;
    struct rlimit rl = {};
    struct sigaction sa = {};

    /*
     * Clear file creation mask.
     */
    umask(0);

    /*
     * Get maximum number of file descriptors.
     */
    if (getrlimit(RLIMIT_NOFILE, &rl) < 0)
        err_quit("%s: can’t get file limit", cmd);

    /*
     * Become a session leader to lose controlling TTY.
     */
    if ((pid = fork()) < 0)
        err_quit("%s: can’t fork", cmd);
```

```c
    else if (pid != 0) /* parent */
        exit(0);
    setsid();

    /*
     * Ensure future opens won’t allocate controlling TTYs.
     */
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);

    sa.sa_flags = 0;
    if (sigaction(SIGHUP, &sa, NULL) < 0)
        err_quit("%s: can’t ignore SIGHUP", cmd);
    if ((pid = fork()) < 0)
        err_quit("%s: can’t fork", cmd);
    else if (pid != 0) /* parent */
        exit(0);
    /*
     * Change the current working directory to the root so
     * we won’t prevent file systems from being unmounted.
     */
    if (chdir("/") < 0)
        err_quit("%s: can’t change directory to /", cmd);
    /*
     * Close all open file descriptors.
     */
    if (rl.rlim_max == RLIM_INFINITY)
        rl.rlim_max = 1024;
    for (i = 0; i < rl.rlim_max; i++)
        close(i);

    /*
     * Attach file descriptors 0, 1, and 2 to /dev/null.
     */
    fd0 = open("/dev/null", O_RDWR);
    fd1 = dup(0);
    fd2 = dup(0);

    /*
     * Initialize the log file.
     */
    openlog(cmd, LOG_CONS, LOG_DAEMON);
    if (fd0 != 0 || fd1 != 1 || fd2 != 2)
    {
        syslog(LOG_ERR, "unexpected file descriptors %d %d %d", f
        exit(1);
    }
}
```

```
91   int main() {
92       char path[1024];
93       const char *filename = "/assignment11.txt";
94       getcwd(path, sizeof(path));
95       strcat(path, filename);
96       daemonize("konichiwa");
97       int fd = open(path, O_CREAT | O_RDWR);
98       const char* LOGINNAME = "Login name: ";
99       char* LoginName = getlogin();
100      write(fd, LOGINNAME, strlen(LOGINNAME));
101      write(fd, LoginName, strlen(LoginName));
102      write(fd, "\n", strlen("\n"));
103      close(fd);
104      return 0;
105  }
106
```

- Line 92 to Line 95
  We record the current path to `path`, and catenate `"/assignment11.txt"` to it. This help us to create a file named `assignment11.txt` under the current folder after we execute `daemonize()`.
- Line 96
  (1) Call the daemonize function correctly.
- Line 97
  (2) Create a text file called “assignment11.txt”.
- Line 99
  (2) Use getlogin() function to get the login name.
- Line 100 to Line 103
  (2) Write “Login name: [login name]” in “assignment11.txt”

(3) Explain the purpose of every step executed in the `daemonize` function.

- Line 29
  Set `umask` so that the permissions of all following files created are 777.
- Line 34
  Get the maximum number of file descriptors for the current process.
- Line 40 to Line 44
  Fork a child and let the child process does not have a control terminal. (Like we have done in assignment 7)
- Line 46 to Line 54
  Set the behavior of the signal handler. Setting `sa_handler` to SIG_IGN means that when a SIGHUP signal is received, it will be ignored instead of executing

the default handler. Other signals will not be blocked while the `SIGHUP` signal is being processed since `sigemptyset` initializes the `sa_mask` to an empty signal set. Setting `sa_flags=0` means that no special behavior flags. `sigaction` sets the SIGHUP signal handler to the previously defined `sa`.

- Line 55 to Line 58
  Fork again here. This makes the daemon process no longer the first process of the session, so there will never be a chance to obtain the controlling terminal. If there is no fork here, the session first process may still open the control terminal.
- Line 63 to Line 64
  Change the current directory to `root`.
- Line 68 to Line 71
  Close all file descriptors.
- Line 76 to Line 78
  Redirect `stdin`, `stdout`, `stderr to /dev/null`.
- Line 83
  Initialize the log file.
- Line 84 to Line 88
  Check whether all the file descriptor had closed correctly and the attached file descriptors are correct while executing Line 76 to Line 78.

(4) Discuss what would happen to the process after becoming a **daemon process**.

a. **Parent Process**: The parent process of the daemon becomes the init process. This prevents unintentional termination due to the parent process exiting.
b. **Session and Process Group**: The daemon creates a new session and process group to become independent of the controlling terminal. (Like assignment 7)
c. **Signals Handling**: Signal handling might be adjusted to handle signals relevant to the daemon's purpose, ignoring or acting upon specific signals as required.
d. **Working Directory**: The daemon process changes its working directory to the root directory. This helps avoid issues related to unmounting or changing file system directories.
e. **File Descriptors**: The daemon closes all file descriptors(here at most 1024) inherited from the parent process to prevent potential issues.
f. **Standard I/O**: The daemon redirects `stdin`, `stdout`, and `stderr to` `/dev/null`, detaching from the terminal.
g. **Running in Background**: This daemon also runs in the background, detached from the terminal.