ADVANCED UNIX PROGRAMMING ASSIGNMENT REPORT

ASSIGNMENT 11

TEAM 9 — 林禾堃、馬毓昇、陳曦

Dec 2023

1. Code Implementation

First, get and store the current working directory.

```
char cwd[1024];
getcwd(cwd, sizeof(cwd));
printf("Current working dir: %s\n", cwd);
```

Then, call the daemonize function.

```
daemonize("daemonize");
```

Finally, use `getlogin` to check the login name of the process and save it to a file under the directory we stored before the `daemonize` function change the current working directory.

```
char *login = getlogin();

// open a file in the origin working directory
char *file_path = strcat(cwd, "/assignment11.txt");
FILE *file = fopen(file_path, "w");
// otherwise, the file would exist under / (root)
// because we changed the current working directory to the
// in the daemonize function
if (file != NULL) {
    if (login != NULL) {
        fprintf(file, "Login name: %s\n", login);
    } else {
        fprintf(file, "No login name.\n");
    }
    fclose(file);
}
```

2. Result

The desired output `Login name: root` was printed into the file, as expected.

```
root@genet0:~/Advanced-UNIX-Programming/HW11 # ./assignment11
Current working dir: /root/Advanced-UNIX-Programming/HW11
root@genet0:~/Advanced-UNIX-Programming/HW11 # cat assignment11.txt
Login name: root
root@genet0:~/Advanced-UNIX-Programming/HW11 #
```

3. The daemonize function

First, `umask` was used to reset permission masks, which lets the daemon process have full control over file permission.

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

Then, we get as much file descriptors as we can, in order to close all of the opened ones later, and prevent any resource leak.

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

We also fork and terminate the parent process, prevent shell from being blocked, and the process will then be able to be a session leader, while avoiding becoming a process group leader. Then, call `setsid` to become a session leader and a process group leader, to get rid of controlling terminal.

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

After that, we set the signal handler to ignore the `SIGHUP`, so the process will not be affected when the terminal is closed. Moreover, the process forks and terminate the parent again to prevent the child process from acquiring controlling terminal.

```
 /*
  * 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);
```

And then, change the current working directory to root and close those file descriptors we got, and redirect descriptors `{0, 1, 2}` to `/dev/null`, since daemon process shouldn't access stdin, stdout, stderr.

```
/*
 * 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);
```

Finally, setup log files or log systems to store output from a daemon.

```
/*
 * 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", fd0, fd1, fd2);
    exit(1);
}
```

4. What would happen to the daemonized process

In short, as what the `daemonize` function do to the process, the process will become a session leader, lose controlling terminal, work in the background, and handle all the inherited file descriptors. If it produced any outputs, it will be stored in system log now.