Clear File Creation Mask (umask(0))

By calling umask(0), the process clears its file mode creation mask. This ensures that the daemon has complete control over the permissions of files it creates. The 0 argument allows the daemon to set any file permission.

Get Maximum Number of File Descriptors (getrlimit and RLIMIT_NOFILE)

The getrlimit function is used to determine the maximum number of file descriptors that the process is allowed to have. This information is crucial for later steps when closing file descriptors.

Become a Session Leader (fork and setsid)

The process forks to create a child process, and the child becomes a session leader by calling setsid(). This ensures that the process is detached from its controlling terminal, preventing it from receiving terminal signals and allowing it to run independently in the background.

Ignore SIGHUP (sigaction and SIGHUP)

By ignoring the SIGHUP signal, the daemon becomes immune to hang-ups (e.g., when the terminal is closed). This is a common practice to ensure that the daemon continues running even if the terminal that started it is closed.

Fork Again and Change Working Directory to '/'

The process forks again to ensure that it is not a session leader (preventing the possibility of acquiring a new controlling terminal in the future). It then changes its working directory to the root directory (/). Changing the working directory ensures that the daemon does not prevent file systems from being unmounted.

Close All Open File Descriptors

The loop iterates through all possible file descriptors and closes them. This step is crucial to release any file descriptors inherited from the parent process that may interfere with the daemon's operation.

Attach File Descriptors 0, 1, and 2 to '/dev/null'

File descriptors 0 (stdin), 1 (stdout), and 2 (stderr) are redirected to /dev/null. This effectively detaches the daemon from the standard input/output, ensuring that it won't inadvertently read input or write output to the terminal.

Initialize the Log File (openlog)

The openlog function is used to initialize the logging facility. This is important for daemons because they often run in the background without a terminal, and logging to system logs becomes crucial for monitoring and debugging. The syslog facility allows the daemon to send log messages to the system logger.

The code also checks if file descriptors 0, 1, and 2 have been successfully redirected to /dev/null and logs an error if not. This is a safety check to ensure that unexpected file descriptors are not open, which could lead to unexpected behavior.

The daemonize function takes a series of steps to detach a process from its terminal, close unnecessary file descriptors, redirect standard file descriptors, and set up logging. These steps are essential for creating a well-behaved daemon process in a Unix-like environment.

Once the daemonize function has been executed, and the process has become a daemon, which parent process is init process, several changes occur that contribute to its behavior as a background process.

Detached from Controlling Terminal:

The process becomes a session leader and is detached from its controlling terminal. This means that it is no longer associated with any terminal, and it won't receive terminal-related signals (e.g., SIGHUP) that are typically sent when a terminal is closed.

Ignoring SIGHUP:

The daemon has set up a signal handler to ignore the SIGHUP signal. This signal is often sent to processes when the terminal is closed. Ignoring SIGHUP ensures that the daemon continues running even if the terminal that started it is closed.

Working Directory and File Permissions:

The daemon changes its working directory to the root directory ("/"). This ensures that the daemon doesn't prevent file systems from being unmounted, as it is not tied to any specific directory.

The file mode creation mask (umask) is set to 0, allowing the daemon to have full control over the permissions of files it creates.

Closing File Descriptors:

All open file descriptors are closed, except for standard input (0), standard output (1), and standard error (2). This prevents any file descriptors inherited from the parent process from interfering with the daemon's operation.

Redirecting Standard File Descriptors:

File descriptors 0, 1, and 2 are redirected to /dev/null. This effectively detaches the daemon from standard input, standard output, and standard error. Any input or output operations on these descriptors are discarded.

Initialization of Logging:

The daemon initializes the logging facility using openlog. This is important for daemons, as they often run in the background without a terminal. Logging to system logs (e.g., syslog) allows for monitoring and debugging the daemon's behavior.

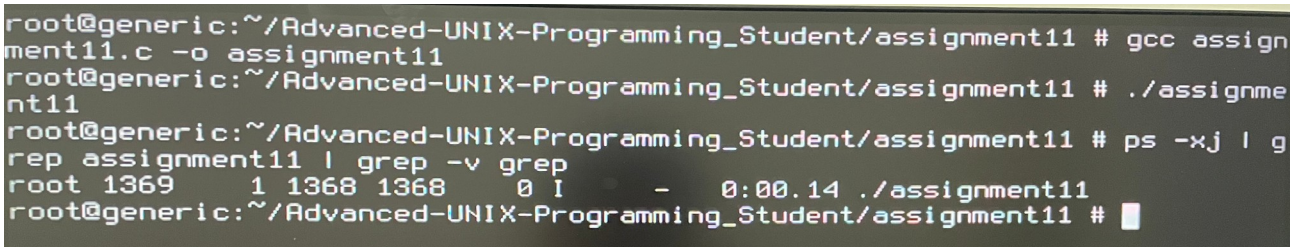
Process Hierarchy:

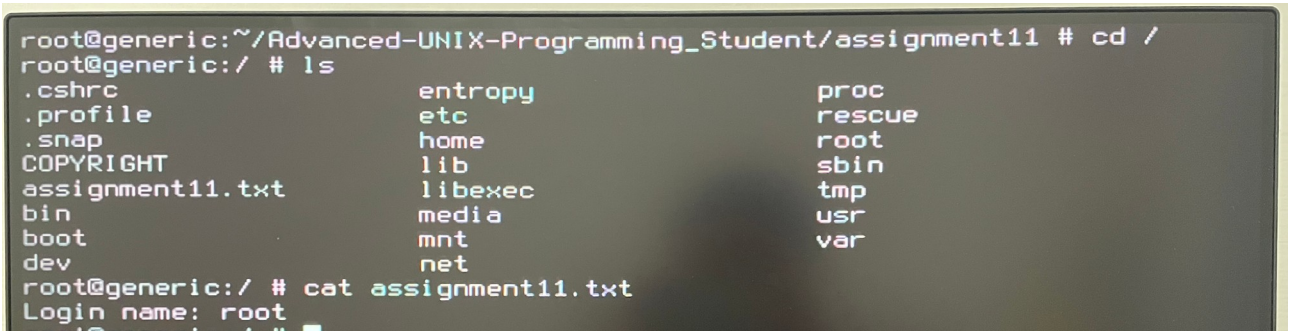
The process hierarchy is adjusted to ensure that the daemon is not a session leader, preventing the possibility of acquiring a new controlling terminal in the future.

Parent Process Exits:

The original parent process (the one that called daemonize) exits, leaving the daemonized process running in the background.

After these steps, the daemon process continues its execution independently of the terminal and is typically designed to perform its designated tasks as a background service. The daemon is resilient to terminal-related events, has a clean environment, and logs its activities for monitoring and debugging purposes. In short, when a process becomes a daemon, it will has those properties, **N*o Dependency on the Controlling Terminal, Background Execution, and Session Independence.***





The assignment11.txt will be written in “/” which is the directory that daemon switches to.