Daemon processes

Lecture 10

Introduction – 1/2

- A daemon (or service) is a background process that is designed to run autonomously, with little or no user intervention.
- A daemon should do one thing, and do it well.
- Daemons should never have direct communication with a user through a terminal.
- All communication should pass through some sort of interface

Introduction – 2/2

- There are numerous ways to start a daemon
  - The system initialization scripts (`/etc/rc` or `/etc`)
  - The *inetd* superserver
  - *cron* daemon
  - The *at* command
  - **from user terminals**

- Since **a daemon does not have a controlling terminal,** it needs some way to output message when something happens, either normal informational messages, or emergency messages that need to be handled by an administrator.

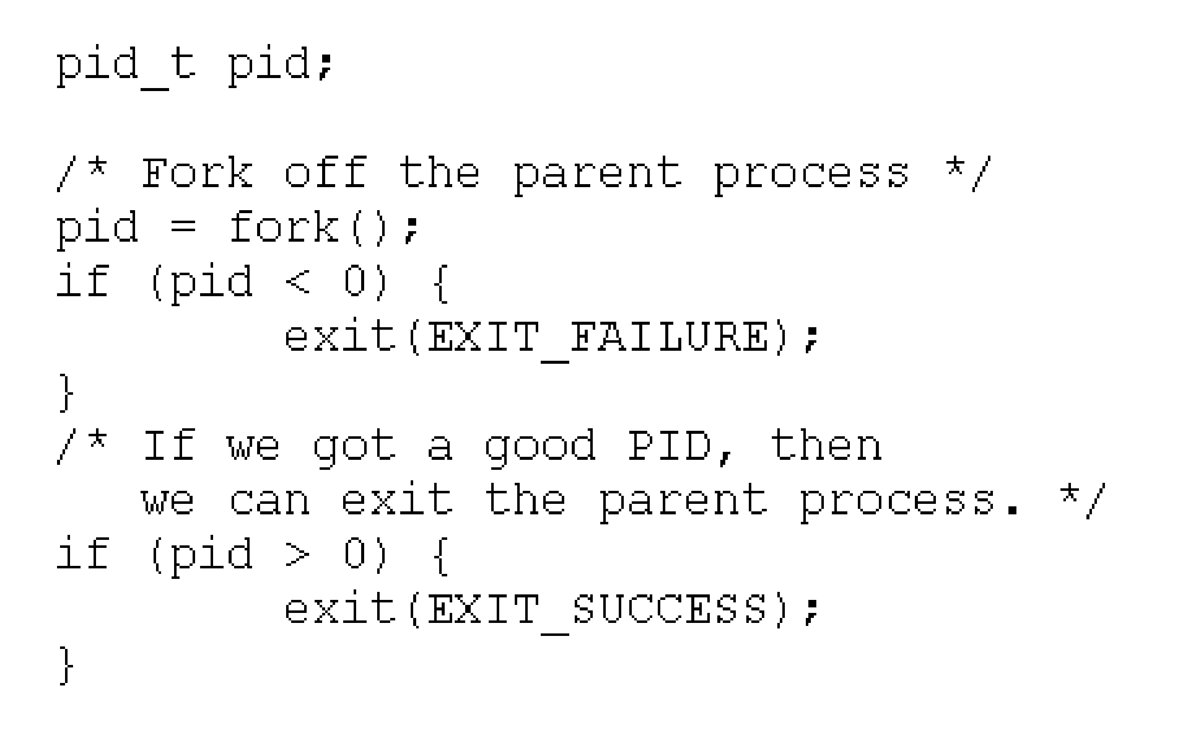
Basic daemon structure

1. Fork off the parent process
2. Change file mode mask
3. Open any logs for writing
4. Create a unique Session ID (SID)
5. Change the current working directory to a safe place
6. Close standard file descriptors
7. Enter actual daemon code

Forking the Parent process – 1/3

- **A daemon is started either by the system itself or a user in a terminal or script.**
- When it does start, the process is just like any other executable on the system.
- To make it truly autonomous, a *child process* must be created where the actual code is executed.
- This is known as **forking**, and it uses the *fork()* function

Forking the Parent process – 2/3



```
pid_t pid;

/* Fork off the parent process */
pid = fork();
if (pid < 0) {
        exit(EXIT_FAILURE);
}
/* If we got a good PID, then
   we can exit the parent process. */
if (pid > 0) {
        exit(EXIT_SUCCESS);
}
```

Forking the Parent process – 3/3

- The *fork()* function returns either the process id (PID) of the child process (not equal to zero), or -1 on failure.

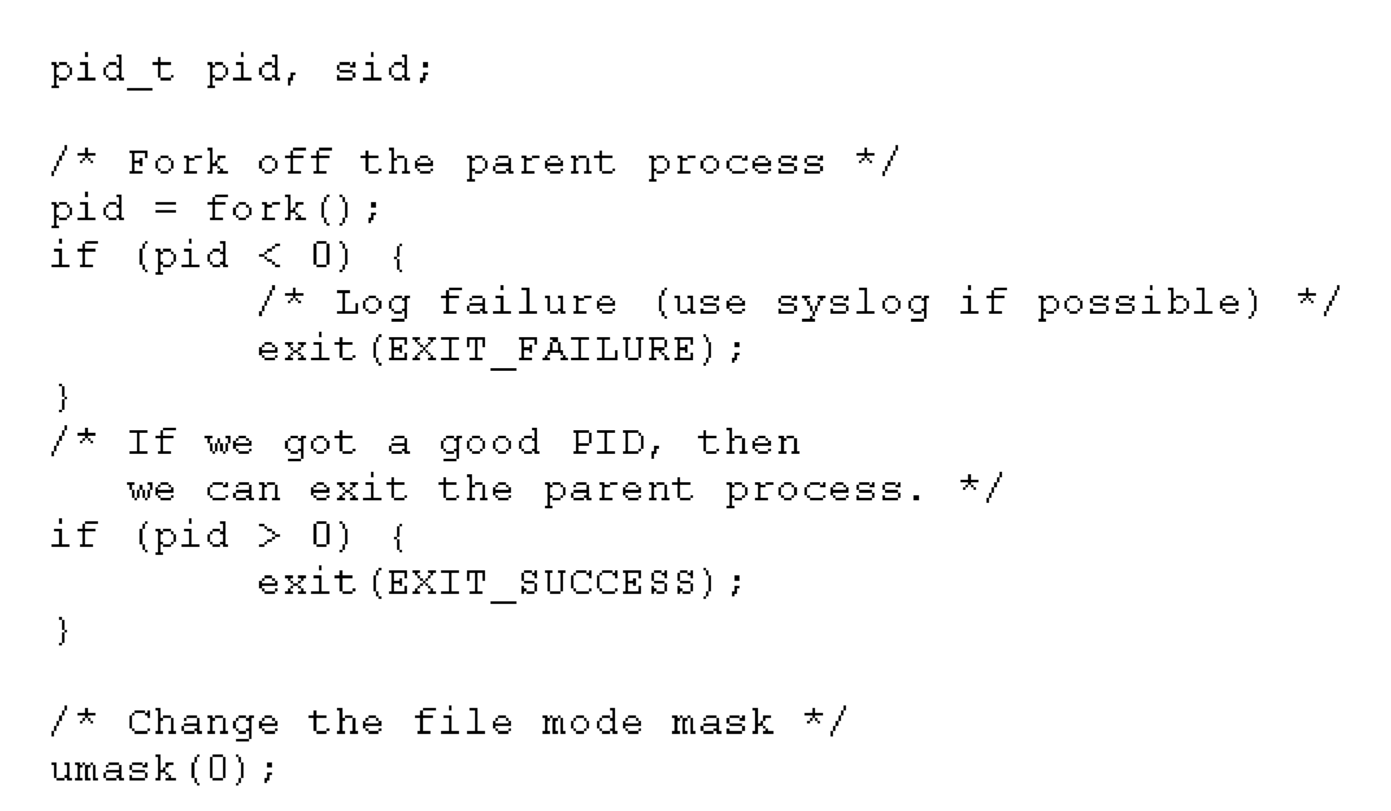
- If the process cannot fork a child, then the daemon should terminate right here.

- If the PID returned from *fork()* did succeed, the parent process must exit gracefully.

Changing file mode mask (umask) – 1/2

- In order to write to any files (including logs) created by the daemon, the file mode mask (umask) must be changed to ensure that they can be written to or read from properly.
- This is similar to running **umask** from the command line, but we do it programmatically here.
- We can use the ***umask()*** function

Changing file mode mask (umask) – 2/2



```
pid_t pid, sid;

/* Fork off the parent process */
pid = fork();
if (pid < 0) {
        /* Log failure (use syslog if possible) */
        exit(EXIT_FAILURE);
}
/* If we got a good PID, then
   we can exit the parent process. */
if (pid > 0) {
        exit(EXIT_SUCCESS);
}

/* Change the file mode mask */
umask(0);
```

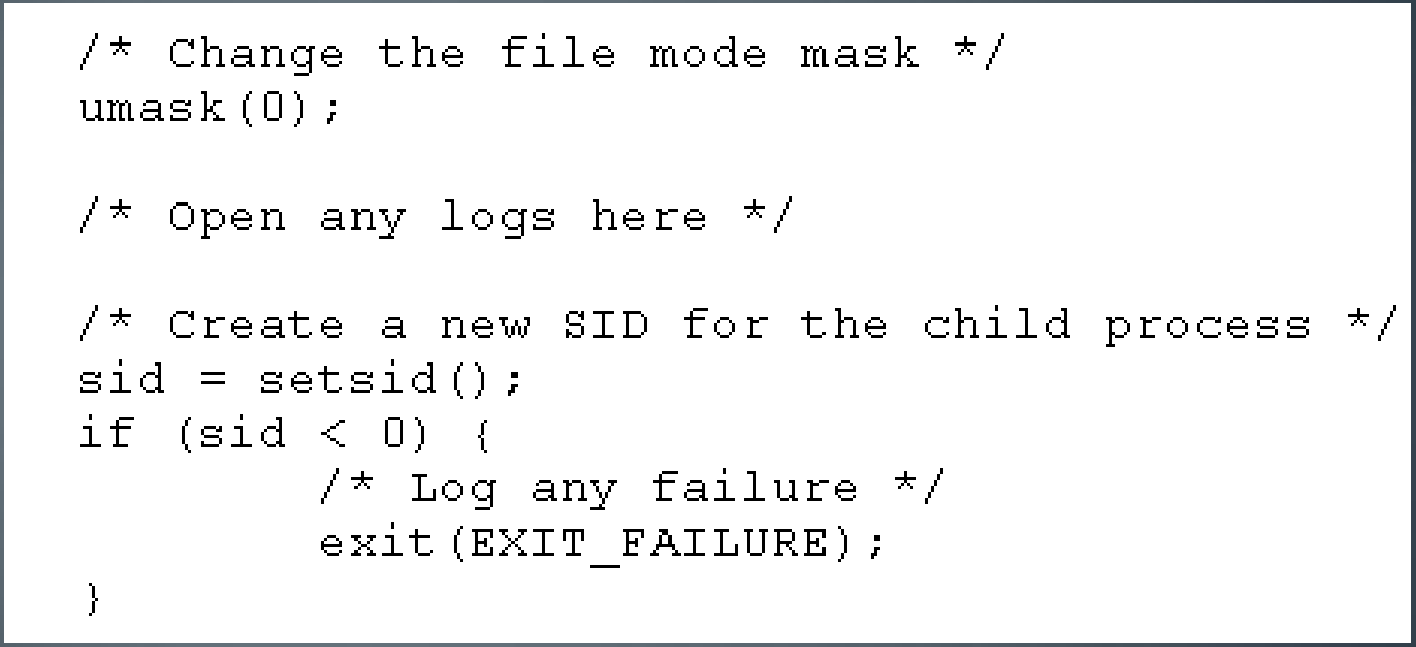
Opening logs for writing

- This part is **optional**, but it is recommended that you open a log file somewhere in the system for writing.
- It may be the only place you can look for debug information about your daemon

Creating unique session ID – 1/3

- From here, the child process must get a unique SID from the kernel in order to operate.
- Otherwise, the child process becomes an orphan in the system.
- The pid_t type, declared in the previous section, is also used to create a new SID for the child process
- *Note: please check the help file in Linux to understand more about its working*

Creating unique session ID – 2/3



```
/* Change the file mode mask */
umask(0);

/* Open any logs here */

/* Create a new SID for the child process */
sid = setsid();
if (sid < 0) {
        /* Log any failure */
        exit(EXIT_FAILURE);
}
```

Creating unique session ID – 3/3

- Again, the *setsid()* function has the same return type as *fork()*.
- We can apply the same error-checking routine here to see if the function created the SID for the child process.

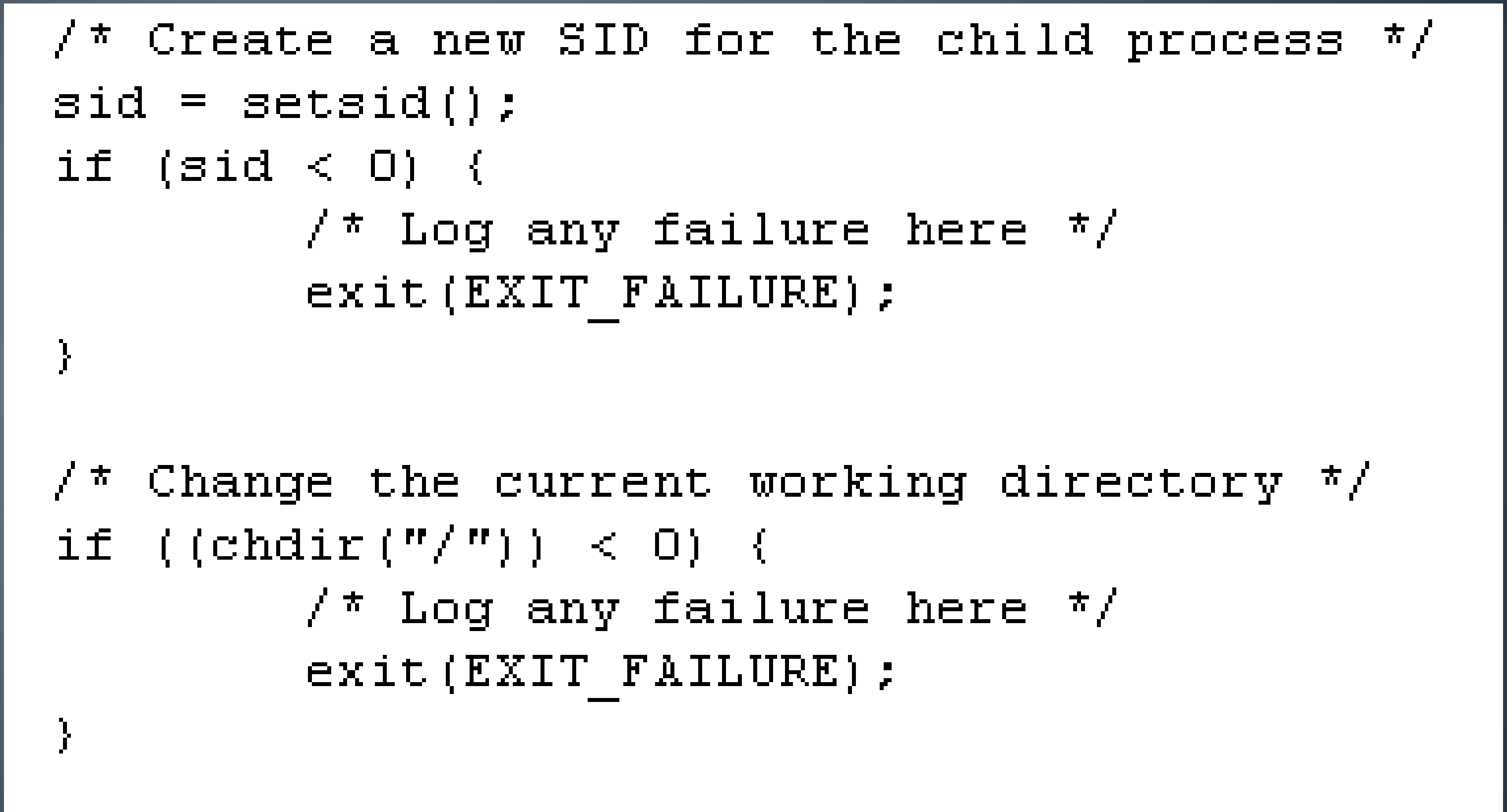
Change working directory – 1/3

- The current working directory should be changed to some place that is guaranteed to always be there.

- Since many Linux distributions do not completely follow the **Linux Filesystem Hierarchy** standard, the only directory that is guaranteed to be there is the **root directory (/)**

- We can do this using the ***chdir()*** function

Change working directory – 2/3



```
/* Create a new SID for the child process */
sid = setsid();
if (sid < 0) {
        /* Log any failure here */
        exit(EXIT_FAILURE);
}

/* Change the current working directory */
if ((chdir("/")) < 0) {
        /* Log any failure here */
        exit(EXIT_FAILURE);
}
```

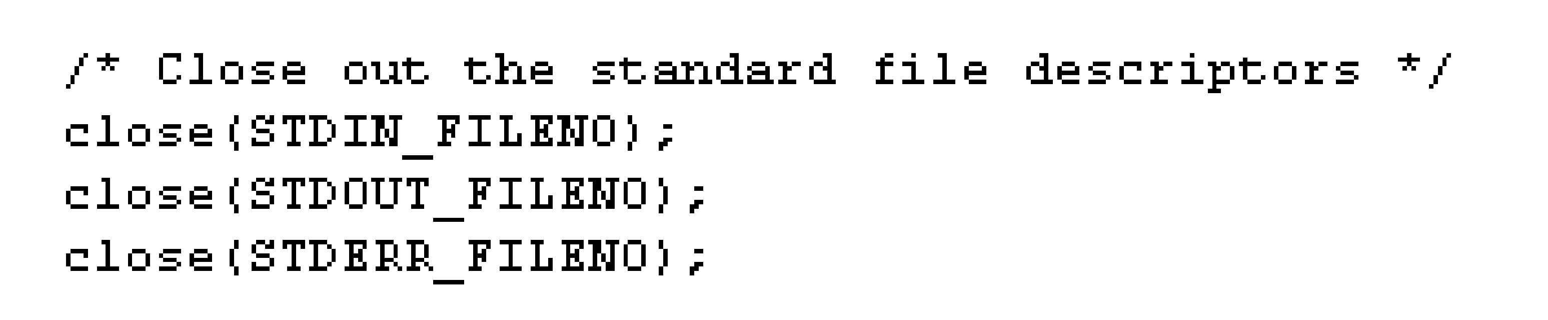
Change working directory – 3/3

- The *chdir()* function returns -1 on failure, so be sure to check for that after changing to the root directory within the daemon

Closing standard file descriptors – 1/2

- One of the last steps in setting up a daemon is **closing out the standard file descriptors (STDIN, STDOUT, STDERR)**
- Since a daemon cannot use the terminal, these file descriptors are redundant and a potential security hazard.
- The ***close()*** function can be used

Closing standard file descriptors – 2/2

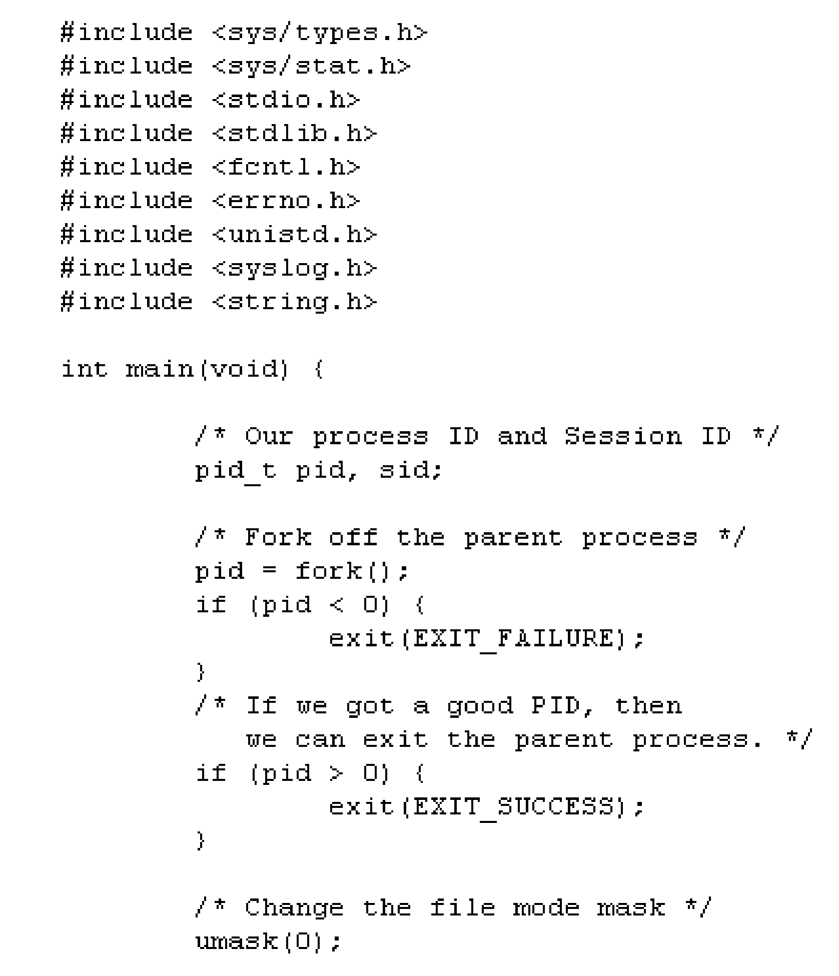


```
/* Close out the standard file descriptors */
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
```

Writing a daemon code

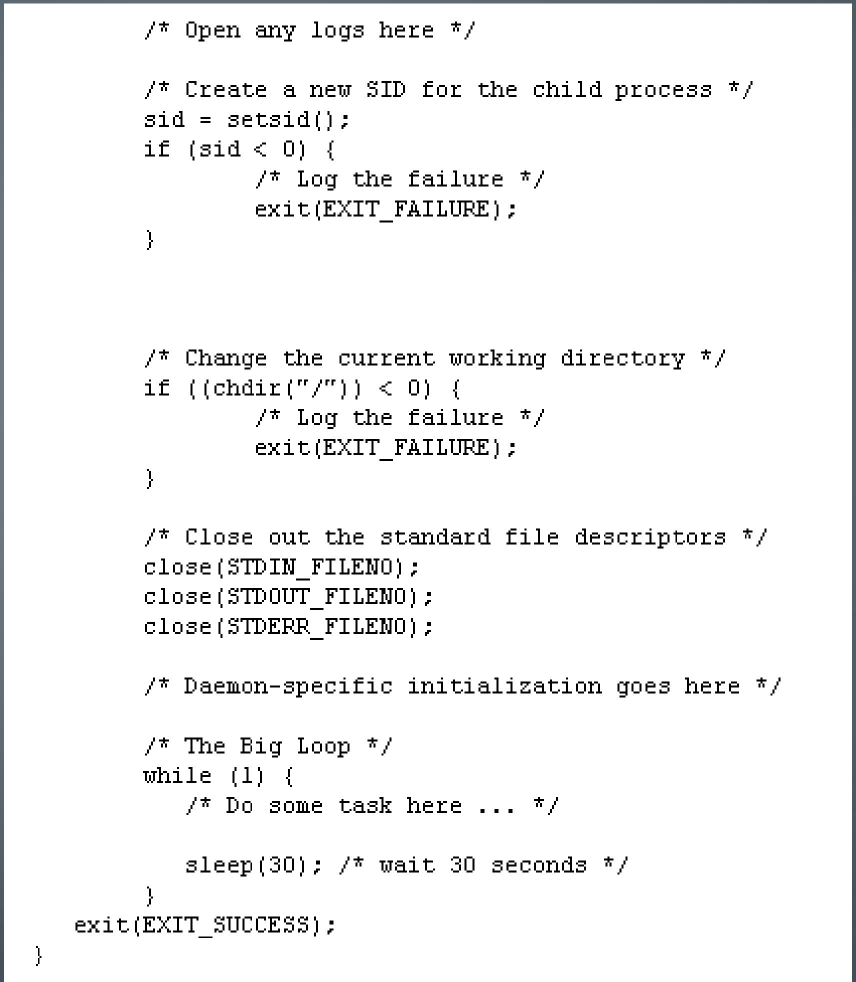
- At this point, you have basically told Linux that you're a daemon, so now it's time to write the actual daemon code.

Complete example – 1/2



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

int main(void) {

        /* Our process ID and Session ID */
        pid_t pid, sid;

        /* Fork off the parent process */
        pid = fork();
        if (pid < 0) {
                exit(EXIT_FAILURE);
        }
        /* If we got a good PID, then
           we can exit the parent process. */
        if (pid > 0) {
                exit(EXIT_SUCCESS);
        }

        /* Change the file mode mask */
        umask(0);
```

Complete example – 2/2



```
        /* Open any logs here */

        /* Create a new SID for the child process */
        sid = setsid();
        if (sid < 0) {
                /* Log the failure */
                exit(EXIT_FAILURE);
        }



        /* Change the current working directory */
        if ((chdir("/")) < 0) {
                /* Log the failure */
                exit(EXIT_FAILURE);
        }

        /* Close out the standard file descriptors */
        close(STDIN_FILENO);
        close(STDOUT_FILENO);
        close(STDERR_FILENO);

        /* Daemon-specific initialization goes here */

        /* The Big Loop */
        while (1) {
           /* Do some task here ... */

           sleep(30); /* wait 30 seconds */
        }
   exit(EXIT_SUCCESS);
}
```

Sample program