# Creating a Daemon Process in C Language with an Example Program

*by* HIMANSHU ARORA *on* FEBRUARY 24, 2012

A daemon process is a process which runs in background and has no controlling terminal.

Since a daemon process usually has no controlling terminal so almost no user interaction is required. Daemon processes are used to provide services that can well be done in background without any user interaction.

For example a process that runs in background and observes network activity and logs any suspicious communication can be developed as a daemon process.

## Daemon Process Design

A daemon process can be developed just like any other process but there is one thing that differentiates it with any other normal process ie having no controlling terminal. This is a major design aspect in creating a daemon process. This can be achieved by :

- Create a normal process (Parent process)
- Create a child process from within the above parent process
- The process hierarchy at this stage looks like : TERMINAL -> PARENT PROCESS -> CHILD PROCESS
- Terminate the the parent process.
- The child process now becomes orphan and is taken over by the init process.
- Call setsid() function to run the process in new session and have a new group.
- After the above step we can say that now this process becomes a daemon process without having a controlling terminal.
- Change the working directory of the daemon process to root and close stdin, stdout and stderr file descriptors.
- Let the main logic of daemon process run.

So we see that above steps mark basic design steps for creating a daemon.

## C fork() Function

Before creating an actual running daemon following the above stated design steps, lets first learn a bit about the fork() system call.

fork() system creates a child process that is exact replica of the parent process. This new process is referred as 'child' process.

This system call gets called once (in parent process) but returns twice (once in parent and second time in child). Note that after the fork() system call, whether the parent will run first or the child is non-deterministic. It purely depends on the context switch mechanism. This call returns zero in child while returns PID of child process in the parent process.

Following are some important aspects of this call :

- The child has its own unique process ID, and this PID does not match the ID of any existing process group.
- The child's parent process ID is the same as the parent's process ID.
- The child does not inherit its parent's memory locks.
- Process resource utilization and CPU time counters are reset to zero in the child.
- The child's set of pending signals is initially empty.
- The child does not inherit semaphore adjustments from its parent.
- The child does not inherit record locks from its parent.
- The child does not inherit timers from its parent.
- The child does not inherit outstanding asynchronous I/O operations from its parent, nor does it inherit any asynchronous I/O contexts from its parent.

For more insight information, please read the man page of this system call.

## The Implementation

Based on the design as mentioned in the first section. Here is the complete implementation :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
int main(int argc, char* argv[])
{
FILE *fp= NULL;
pid_t process_id = 0;
pid_t sid = 0;
// Create child process
process_id = fork();
// Indication of fork() failure
if (process_id < 0)
{
printf("fork failed!\n");
// Return failure in exit status
exit(1);
}
// PARENT PROCESS. Need to kill it.
if (process_id > 0)
{
printf("process_id of child process %d \n", process_id);
// return success in exit status
exit(0);
}
//unmask the file mode
umask(0);
//set new session
sid = setsid();
if(sid < 0)
{
```

Top 10 Open Source Bug Tracking System
Vi and Vim Macro Tutorial: How To Record and Play
Mommy, I found it! -- 15 Practical Linux Find Command Examples
15 Awesome Gmail Tips and Tricks
15 Awesome Google Search Tips and Tricks
RAID 0, RAID 1, RAID 5, RAID 10 Explained with Diagrams
Can You Top This? 15 Practical Linux Top Command Examples
Top 5 Best System Monitoring Tools
Top 5 Best Linux OS Distributions
How To Monitor Remote Linux Host using Nagios 3.0
Awk Introduction Tutorial – 7 Awk Print Examples
How to Backup Linux? 15 rsync Command Examples
The Ultimate Wget Download Guide With 15 Awesome Examples
Top 5 Best Linux Text Editors
Packet Analyzer: 15 TCPDUMP Command Examples
The Ultimate Bash Array Tutorial with 15 Examples
3 Steps to Perform SSH Login Without Password Using ssh-keygen & ssh-copy-id
Unix Sed Tutorial: Advanced Sed Substitution Examples
UNIX / Linux: 10 Netstat Command Examples
The Ultimate Guide for Creating Strong Passwords
6 Steps to Secure Your Home Wireless Network
Turbocharge PuTTY with 12 Powerful Add-Ons

## CATEGORIES

Linux Tutorials
Vim Editor
Sed Scripting
Awk Scripting
Bash Shell Scripting
Nagios Monitoring
OpenSSH
IPTables Firewall
Apache Web Server
MySQL Database
Perl Programming
Google Tutorials
Ubuntu Tutorials
PostgreSQL DB
Hello World Examples
C Programming
C++ Programming
DELL Server Tutorials
Oracle Database
VMware Tutorials

```
// Return failure
exit(1);
}
// Change the current working directory to root.
chdir("/");
// Close stdin. stdout and stderr
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
// Open a log file in write mode.
fp = fopen ("Log.txt", "w+");
while (1)
{
//Dont block context switches, let the process sleep for some time
sleep(1);
fprintf(fp, "Logging info...\n");
fflush(fp);
// Implement and call some function that does core work for this daemon.
}
fclose(fp);
return (0);
}
```

Following is the way through which the code was compiled and executed:

```
$ gcc -Wall deamon.c -o deamon
$ sudo ./deamon
process_id of child process 2936
```

Just observe that the control immediately came back to the terminal ie the daemon is now not associated to any terminal.

When you check the log.txt file located in the root directory, you could see that this daemon process is running.

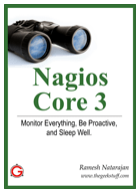
```
$
$ tail -f /Log.txt
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
```

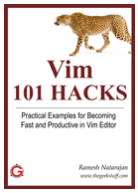


If you enjoyed this article, you might also like..

1. 50 Linux Sysadmin Tutorials
2. 50 Most Frequently Used Linux Commands (With Examples)
3. Top 25 Best Linux Performance Monitoring and Debugging Tools
4. Mommy, I found it! – 15 Practical Linux Find Command Examples
5. Linux 101 Hacks 2nd Edition eBook Free

- Awk Introduction – 7 Awk Print Examples
- Advanced Sed Substitution Examples
- 8 Essential Vim Editor Navigation Fundamentals
- 25 Most Frequently Used Linux IPTables Rules Examples
- Turbocharge PuTTY with 12 Powerful Add-Ons






Comments on this entry are closed.

**bob**
February 24, 2012, 8:27 am

Thank you.
You have a knack of explaining things in the simplest way by taking out all the fluff.
Very few engineers have this ability to explain things this straighforwardly.
Very well written.
Really learned something today without having to google a million search items and get more confused in the process.

∞

**Astorre**
February 24, 2012, 4:55 pm

Excellent article !!!
Thank you very much.

∞

**Jack**
February 25, 2012, 9:19 am

Nice article. I wish you were my professor when I was in college.

∞

**Joe Klemmer**
February 25, 2012, 1:01 pm

The only change I would make to this is that I would not run it in the root directory. e.g. change –
chdir("/");
to –
chdir("/tmp");

But that's just me.

∞

**himanshu**
February 25, 2012, 9:37 pm

Thank you all for your appreciation.

∞

**Nathan**
March 10, 2012, 2:05 am

Do I need to use the 'kill' command to end the child process? Not seeing anything in the while{} loop to automatically end the example program.

∞

**Himanshu**
March 10, 2012, 11:36 pm

@Nathan
Since this is an example of a daemon process and daemon processes are meant to run like background services for indefinite time so they are usually not ended from within the process code. That is the reason I have used the kill command to terminate the process.

∞

**Greg A. Woods**
July 31, 2012, 6:03 pm

Note that most unixy systems, including even those weird ones with Glibc [;-)], have a daemon(3) function which is a bit more portable and safe to use in real-world programs.

As with popen(3) or system(3) it's not going to teach people as much about the lower-level calls, of course, but I think it's remiss to not mention it.

∞

**Dario**
October 2, 2012, 12:33 pm

Thanks a lot.
Very clear and full explanation.
You solved all my doubts about it.

∞

**Ravikanth G Reddy**
December 17, 2012, 1:24 am

Couldn't find an easier one than this. Gateway to entirely new turf. Thanks a ton.

∞

**srinivas**
January 30, 2013, 12:59 am

Thank you for your excellent articles really useful….
-Srinivas

∞

**Degen**
March 19, 2013, 6:30 am

it is nice tutorial.
but where should i put my program if i want to run it at system startup and with root privileges.

∞

**John**
May 8, 2013, 9:49 pm

This is a great article. My coworker is trying to find a way to create a timing system that runs perfectly at once a second. His explanation is absolutely crazy. This is simple and straight to the point.

∞

**Anonymous**
June 13, 2013, 11:59 am

Hi, I'm sorry but this is not the correct execution, 0,1,2 file descriptors must be closed and signal captures where is?

∞

**ilian**
June 18, 2013, 8:47 am

Thank you for that great tutorial. I was able to make my daemon for deleting files form directory with that skeleton. Also changing /tmp is good idea too.

∞

**codemaster**
November 5, 2013, 11:10 am

Great!, been scanning around for such info. Apparently am in love ith linux, opensource, and everything that ships with that.

∞

**Markus Elfring**
November 22, 2013, 8:33 am

I wonder that the function "exit" (and "printf") is shown after the fork() call. How do you
think about to use the function "_exit" (and "write") instead?
How much does async-signal-safety (from POSIX view) matter here?

∞

**alias**
May 17, 2014, 4:31 am

I run the code but I get the errors: undefined reference to fork, setsid and sleep. I have
included unistd.h, and I am using windows os.Does that have anything to do with these errors?

∞

**AyeChan**
May 29, 2014, 3:54 am

Nice explanation.This article is very useful for me.Thank you very much.

∞

**karan**
August 9, 2014, 12:32 am

how can i execute the each step in daemon process while running the daemon in some file ..using c program .

∞

**Kunal**
October 9, 2014, 5:28 am

Reallly good explanation

∞

**isahak**
March 29, 2015, 6:52 am

Thanks Friend

∞

**Daniel**
May 21, 2015, 2:39 pm

Thank you! Very nice tutorial. Easy to follow. Worked perfectly for me!

∞

**Divya**
July 20, 2015, 3:25 am

How can we interact with mysql database in daemon ? can we use web services as well ?

∞

**Sangeetha**
February 17, 2016, 4:05 am

Thank you for such simple and clear explanation!!!

∞

**Ghansham**
December 10, 2016, 11:05 pm

The above program is not getting executed as shown in the post.
It say file cant be opened with the errorno 13 (Permission denied)

∞

Next post: Linux Memory Management – Swapping, Caches and Shared VM
Previous post: How to Install GIT for Windows and Create / Clone Remote Repositories