

# **Embedded Linux Systems Programming Bash, C, Python**

**2023-08-22**

**NORMAN MCENTIRE**

# Languages for Embedded Linux Systems Programs

- Any language can be use for Linux Systems Programs
- But three languages are used most often
  - Bash - Bourne Again Shell
    - On nearly every embedded Linux system
  - C - C Language
    - Requires toolchain
  - Python - Modern scripting language
    - Often not available on embedded Linux systems

# Processes

- When a program starts executing it is a Process
  - A process is a “program in execution”
- Use ps (Process Status) command to see processes
- Use pstree command to tree hierarchy of process
  - PID (Process ID) 1 is the first process in the system
  - First process started by Linux kernel after boot up

# Process Types

- Foreground Process
  - Standard output goes to console
  - Runs to completion (or press Control-C to stop)
    - NOTE: Control+c is called SIGINT (Signal, Interrupt)
      - (More on signals later)
- Background Process
  - Standard output goes to console
  - Runs to completion (or do a “kill” to terminate the process)
- Daemon Process
  - Like background process but not attached to console output
  - Console output goes to “/dev/null”

# Tour of Systems Programming Languages

- We want to gain hands-on experience with the three key Embedded Linux Systems Programs
  - Bash
  - C
  - Python
- For Each we want to understand
  - Foreground
  - Background
  - Daemon

# man bash

BASH(1)	General Commands Manual	BASH(1)
NAME	<div><div>bash</div><div>GNU Bourne-Again SHell</div></div> <div>← Written by Steven Bourne</div>	
SYNOPSIS	<b>bash</b> [options] [command_string   file]	
COPYRIGHT	Bash is Copyright (C) 1989–2020 by the Free Software Foundation, Inc.	
DESCRIPTION	<p><b>Bash</b> is an <b>sh-compatible</b> command language interpreter that executes commands read from the standard input or from a file. <del><b>Bash</b> also incorporates useful features from the</del> <b>Korn</b> and <b>C</b> shells (<b>ksh</b> and <b>cs</b>h).</p> <p><b>Bash</b> is intended to be a conformant implementation of the Shell and Utilities portion of the IEEE POSIX specification (IEEE Standard 1003.1). <b>Bash</b> can be configured to be <b>POSIX-conformant</b> by default.</p>	

POSIX = Portable Operating System Interface

# bash, sh, dash

Nearly **1MB**  
In size

```
$ ls -l /usr/bin/bash
-rwxr-xr-x 1 root root 974312 Mar 27  2022 /usr/bin/bash
```

```
$ ls -l /usr/bin/bash
-rwxr-xr-x 1 root root 974312 Mar 27  2022 /usr/bin/bash
```

```
$ which sh
/usr/bin/sh
```

```
$ ls -l /usr/bin/sh
lrwxrwxrwx 1 root root 4 Feb 20 17:02 /usr/bin/sh -> dash
```

About **91K**  
In Size

```
$ ls -l /usr/bin/dash
-rwxr-xr-x 1 root root 91904 Dec 10  2020 /usr/bin/dash
```



# man sh

DASH(1)	BSD General Commands Manual	DASH(1)
<b>NAME</b>		
dash - command interpreter (shell)		
<b>SYNOPSIS</b>		
dash [-aCefnuvxIimqVEbp] [+aCefnuvxIimqVEbp] [-o option_name] [+o option_name] [command_file [argument ...]]		
dash -c [-aCefnuvxIimqVEbp] [+aCefnuvxIimqVEbp] [-o option_name] [+o option_name] command_string [command_name [argument ...]]		
dash -s [-aCefnuvxIimqVEbp] [+aCefnuvxIimqVEbp] [-o option_name] [+o option_name] [argument ...]		
<b>DESCRIPTION</b>		
<p>dash is the standard command interpreter for the system. The current version of dash is in the process of being changed to conform with the POSIX 1003.2 and 1003.2a specifications for the shell. This version has many features which make it appear similar in some respects to the Korn shell, but it is not a Korn shell clone (see ksh(1)). Only features designated by POSIX, plus a few Berkeley extensions, are being incorporated into this shell. This man page is not intended to be a tutorial or a complete specification of the shell.</p>		
POSIX - Portable Operating System Interface	"Berkeley Extensions - From University of California at Berkeley BSD - Berkeley Standard Distribution of Unix	



# Busybox

- Later in the course we will cover Busybox
- Busybox is a single executable that implements both the shell and also many common shell commands
  - arch, uname, ls, cat, ps, etc. (100s of commands)
- Busybox often used in embedded Linux systems to reduce the memory footprint

# Hello Bash - Demo 1

## Foreground Program Runs to Completion

```
$ cat hello-bash-1.sh
```

```
#!/bin/bash
```

```
echo "Hello Bash!"
```

```
echo "My pid: $$"
```

```
$ bash hello-bash.sh
```

```
Hello Bash!
```

```
My pid: 4675
```

```
$ chmod +x hello-bash.sh
```

```
$ ./hello-bash.sh
```

```
Hello Bash!
```

```
My pid: 4730
```

```
$ echo $?
```

```
0
```

# Hello Bash - Demo 2

## Foreground Program Ended by Pressing ENTER or Control+c or kill

```
$ cp hello-bash-1.sh hello-bash-2.sh
```

```
$ vi hello-bash-2.sh
```

```
$ cat hello-bash-2.sh  
#!/bin/bash
```

```
echo "Hello Bash!"  
echo "My pid: $$"
```

```
echo "Press ENTER to end"  
read ANSWER
```

```
$ bash hello-bash-2.sh  
Hello Bash!  
My pid: 4723  
Press ENTER to end
```

```
$ chmod +x hello-bash-2.sh  
./hello-bash-2.sh  
Hello Bash!  
My pid: 4750  
Press ENTER to end  
^C
```

```
$ echo $?  
0
```

```
metaembedded@raspberrypi:~ $ ps aux | head -1  
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
metaembedded@raspberrypi:~ $ ps aux | grep hello  
metaemb+  4773  0.0  0.0   7768   588 pts/0    S+   04:17   0:00 /bin/bash ./hello-bash-2.sh  
metaemb+  4870  0.0  0.0   7452   492 pts/1    S+   04:18   0:00 grep --color=auto hello  
metaembedded@raspberrypi:~ $
```

VSZ = Virtual Memory Size

RSS = Resident Storage Size

TTY = Teletype (Console)

```
$ kill 4773
```

```
$ ./hello-bash-2.sh  
Hello Bash!  
My pid: 4773  
Press ENTER to end  
Terminated
```

# The “kill” command sends a signal to a process

## kill PID

KILL(1)	User Commands	KILL(1)
<b>NAME</b>		
kill - send a signal to a process		
<b>SYNOPSIS</b>		
kill [options] <pid> [...]		
<b>DESCRIPTION</b>		
The default signal for kill is TERM. Use -1 or -L to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and 0. Alternate signals may be specified in three ways: -9, -SIGKILL or -KILL. Negative PID values may be used to choose whole process groups; see the PGID column in ps command output. A PID of -1 is special; it indicates all processes except the kill process itself and init.		



# kill -l

## List of Signals

```
metaembedded@raspberrypi:~ $ kill -l
1) SIGHUP          2) SIGINT          3) SIGQUIT         4) SIGILL          5) SIGTRAP
6) SIGABRT         7) SIGBUS         8) SIGFPE          9) SIGKILL         10) SIGUSR1
11) SIGSEGV        12) SIGUSR2       13) SIGPIPE        14) SIGALRM         15) SIGTERM
16) SIGSTKFLT      17) SIGCHLD       18) SIGCONT        19) SIGSTOP         20) SIGTSTP
21) SIGTTIN        22) SIGTTOU       23) SIGURG         24) SIGXCPU         25) SIGXFSZ
26) SIGVTALRM      27) SIGPROF       28) SIGWINCH       29) SIGIO           30) SIGPWR
31) SIGSYS         34) SIGRTMIN      35) SIGRTMIN+1     36) SIGRTMIN+2     37) SIGRTMIN+3
38) SIGRTMIN+4     39) SIGRTMIN+5    40) SIGRTMIN+6     41) SIGRTMIN+7     42) SIGRTMIN+8
43) SIGRTMIN+9     44) SIGRTMIN+10   45) SIGRTMIN+11    46) SIGRTMIN+12    47) SIGRTMIN+13
48) SIGRTMIN+14    49) SIGRTMIN+15   50) SIGRTMAX-14    51) SIGRTMAX-13    52) SIGRTMAX-12
53) SIGRTMAX-11    54) SIGRTMAX-10   55) SIGRTMAX-9     56) SIGRTMAX-8     57) SIGRTMAX-7
58) SIGRTMAX-6     59) SIGRTMAX-5    60) SIGRTMAX-4     61) SIGRTMAX-3     62) SIGRTMAX-2
63) SIGRTMAX-1     64) SIGRTMAX
```



# Hello Bash - Demo 3

## Run in Background

```
$ cp hello-bash-2.sh hello-bash-3.sh
```

```
$ vi hello-bash-3.sh
```

```
$ cat hello-bash-3.sh
#!/bin/bash
```

```
echo "Hello Bash!"
echo "My pid: $$"
```

```
COUNT=1
while true ; do
    COUNT=$((COUNT+1))
    echo "COUNT: $COUNT"
    sleep 15
done
```

```
$ ./hello-bash-3.sh &
[1] 5350
```

```
$ Hello Bash!
My pid: 5350
COUNT: 2
```

```
$ pstree 5350
hello-bash-3.sh--sleep
```

```
$ COUNT: 3
```

```
metaembedded@raspberrypi:~ $ ps
  PID TTY          TIME CMD
 5350 pts/0    00:00:00 hello-bash-3.sh
 5353 pts/0    00:00:00 sleep
 5354 pts/0    00:00:00 ps
25808 pts/0    00:00:01 bash
```

Use & symbol  
To run in background

NOTE: Can also press Ctrl+z  
When a process is in the Foreground  
To put it in the background.  
Then use “foreground” command  
To return to foreground

```
$ kill 5350
$ ps
  PID TTY          TIME CMD
 5359 pts/0    00:00:00 sleep
 5360 pts/0    00:00:00 ps
25808 pts/0    00:00:01 bash
[1]+  Terminated                  ./hello-bash-3.sh
```

# Hello C - Demo 1

## Foreground Program Runs to Completion

```
$ vi hello-c-1.c
```

```
$ cat hello-c-1.c
#include <stdio.h>
#include <unistd.h>
```

```
int main(int argc, char *argv[]) {
    printf("Hello C!\n");
    printf("My PID: %d\n", getpid());
    return 0;
}
```

```
$ gcc -Wall -o hello-c-1 hello-c-1.c
```

```
$ file hello-c-1.c
hello-c-1.c: C source, ASCII text
```

```
$ file hello-c-1
hello-c-1: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), . . .
```

```
$ ./hello-c-1
Hello C!
My PID: 5628
```

```
$ echo $?
0
```

# Hello C - Demo 2

## Foreground Program Ended by Pressing ENTER or Control+c or kill

```
$ cp hello-c-1.c hello-c-2.c
```

```
$ vi hello-c-2.c
```

```
$ cat hello-c-2.c
```

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
int main(int argc, char *argv[]) {  
    printf("Hello C!\n");  
    printf("My PID: %d\n", getpid());
```

```
    printf("Press ENTER to end: ");  
    getchar();
```

```
    return 0;  
}
```

```
$ gcc -Wall -o hello-c-2 hello-c-2.c
```

```
$ ./hello-c-2
```

```
Hello C!
```

```
My PID: 5651
```

```
Press ENTER to end:
```

```
$ echo $?
```

```
0
```

```
$ ./hello-c-2
```

```
Hello C!
```

```
My PID: 5652
```

```
Press ENTER to end: ^C
```

```
$ echo $?
```

```
130
```

```
$ ps aux | head -1
```

```
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

```
$ ps aux | grep hello
```

```
metaemb+  5722  0.0  0.0   1884    356 pts/0    S+   04:59   0:00 ./hello-c-2
```

```
metaemb+  5727  0.0  0.0   7452    540 pts/1    S+   04:59   0:00 grep --color=auto hello
```

VSZ = Virtual Memory Size

RSS = Resident Storage Size

TTY = Teletype (Console)

```
$ kill 5722
```

```
$ ./hello-c-2
```

```
Hello C!
```

```
My PID: 5722
```

```
Press ENTER to end: Terminated
```

# Hello C - Demo 3

## Run in Background

```
$ cp hello-c-2.c hello-c-3.c
$ vi hello-c-3.c
$ cat hello-c-3.c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    printf("Hello C!\n");
    printf("My PID: %d\n", getpid());

    int count = 0;
    while (1) {
        count++;
        printf("count: %d\n", count);
        sleep(15);
    }

    return 0;
}
```

```
$ gcc -Wall -o hello-c-3 hello-c-3.c
$ ./hello-c-3 &
[1] 5834
```

```
$ Hello C!
My PID: 5834
count: 1
```

```
$ ps
  PID TTY          TIME CMD
 5834 pts/0        00:00:00 hello-c-3
 5835 pts/0        00:00:00 ps
25808 pts/0        00:00:01 bash
```

```
$ count: 2
```

```
$ kill 5834
```

```
$ ps
  PID TTY          TIME CMD
 5838 pts/0        00:00:00 ps
25808 pts/0        00:00:01 bash
[1]+  Terminated                  ./hello-c-3
```

NOTE: Can also press Ctrl+z  
When a process is in the Foreground  
To put it in the background.  
Then use “foreground” command  
To return to foreground

Use “&” to run process  
in background



# man daemon

## Run in the background detached from console

```
DAEMON(3)                                Linux Programmer's Manual                                DAEMON(3)

NAME
    daemon - run in the background

SYNOPSIS
    #include <unistd.h>

    int daemon(int nochdir, int noclose);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    daemon():
        Since glibc 2.21:
            _DEFAULT_SOURCE
        In glibc 2.19 and 2.20:
            _DEFAULT_SOURCE || (_XOPEN_SOURCE && _XOPEN_SOURCE < 500)
        Up to and including glibc 2.19:
            _BSD_SOURCE || (_XOPEN_SOURCE && _XOPEN_SOURCE < 500)

DESCRIPTION
    The daemon() function is for programs wishing to detach themselves from the controlling terminal and run in the background as system daemons.

    If nochdir is zero, daemon() changes the process's current working directory to the root directory ("/"); otherwise, the current working directory is left unchanged.

    If noclose is zero, daemon() redirects standard input, standard output and standard error to /dev/null; otherwise, no changes are made to these file descriptors.
```



# Hello C - Demo 4

## Run as Daemon

```
$ cp hello-c-3.c hello-c-4.c
$ vi hello-c-4.c
$ cat hello-c-4.c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    printf("Calling daemon()\n");
    int rc = daemon(0, 0);
    if (rc < 0) {
        perror("daemon");
        return 1;
    }

    printf("Hello C!\n");
    printf("My PID: %d\n", getpid());

    int count = 0;
    while (1) {
        count++;
        printf("count: %d\n", count);
        sleep(15);
    }

    return 0;
}
```

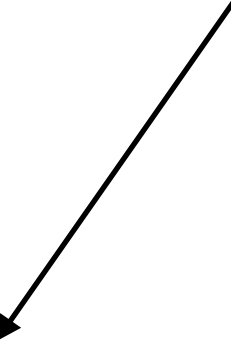
```
$ gcc -Wall -o hello-c-4 hello-c-4.c
$ ./hello-c-4
Calling daemon()

\ $ ps
  PID TTY          TIME CMD
  6087 pts/0        00:00:00 ps
 25808 pts/0        00:00:02 bash

$ ps aux | grep hello
metaemb+  6086  0.0  0.0   1884    68 ?        Ss   05:13   0:00 ./hello-c-4
metaemb+  6089  0.0  0.0   7452   540 pts/0    S+   05:13   0:00 grep --color=auto hello

$ ps aux | head -1
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
$ kill 6086
```

A daemon process  
not attached to any console



# Hello Python - Demo 1

## Foreground program runs to completion

```
$ vi hello-python-1.py

$ cat hello-python-1.py
#!/usr/bin/python

import os
import sys

def main():
    print("Hello Python")
    print("My PID: " + str(os.getpid()))
    sys.exit(0)

if __name__ == "__main__":
    main()
)

$ python hello-python-1.py
Hello Python
My PID: 6288

$ chmod +x hello-python-1.py
$ ./hello-python-1.py
Hello Python
My PID: 6297

$ echo $?
0
```

# Hello Python - Demo 2

## Foreground Program Ended by Pressing ENTER or Control+c or kill

```
$ cp hello-python-1.py hello-python-2.py
$ vi hello-python-2.py
$ cat hello-python-2.py
#!/usr/bin/python

import os
import sys

def main():
    print("Hello Python")
    print("My PID: " + str(os.getpid()))

    answer = input("Press ENTER to end");

    sys.exit(0)

if __name__ == "__main__":
    main()

$ ./hello-python-2.py
Hello Python
My PID: 6527
Press ENTER to end

$ ./hello-python-2.py
Hello Python
My PID: 6528
Press ENTER to end^C
Traceback (most recent call last):
  File "/home/metaembedded/./hello-python-2.py", line 15, in <module>
    main()
  File "/home/metaembedded/./hello-python-2.py", line 10, in main
    answer = input("Press ENTER to end");
KeyboardInterrupt
```

```
$ ps aux | head -1
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND

$ ps aux | grep hello
metaemb+  6583  0.6  0.3 14688  7240 pts/0    S+   05:33   0:00 /usr/bin/python ./hello-python-2.py
metaemb+  6589  0.0  0.0   7452   500 pts/1    S+   05:33   0:00 grep --color=auto hello
```

VSZ = Virtual Memory Size

RSS = Resident Storage Size

TTY = Teletype (Console)

```
$ kill 6583
```

```
$ ./hello-python-2.py
Hello Python
My PID: 6583
Press ENTER to endTerminated
```

# Hello Python - Demo 3

## Run Python Program as Daemon - First Try (did not work)

```
$ cp hello-python-2.py hello-python-3.py
$ vi hello-python-3.py
```

```
$ cat hello-python-3.py
#!/usr/bin/python
```

```
import os
import sys
import time

from daemonize import Daemonize

def main():
    print("Hello Python")
    print("My PID: " + str(os.getpid()))

    count = 0
    while True:
        count += 1
        print("count: " + str(count))
        time.sleep(15)

    sys.exit(0)

if __name__ == "__main__":
    main()
    daemon = Daemonize(app="hello_python_3", pid="/tmp/my_daemon.pid", action=main)
    daemon.start()
```

```
$ ./hello-python-3.py
Traceback (most recent call last):
  File "/home/metaembedded/./hello-python-3.py", line 7, in <module>
    from daemonize import Daemonize
ModuleNotFoundError: No module named 'daemonize'
```

```
$ pip install daemonize
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting daemonize
  Downloading https://www.piwheels.org/simple/daemonize/daemonize-2.5.0-py2.py3-none-any.whl (5
Installing collected packages: daemonize
Successfully installed daemonize-2.5.0
```

```
$ ./hello-python-3.py
Hello Python
My PID: 7020
count: 1
```

NOTE: This demonize did not work  
4or me. See Demo 4

# Hello Python - Demo 4

## Run Python Program as Daemon - 2nd Try - Works

```
$ cat hello-python-4.py
#!/usr/bin/python

import os
import sys
import time

def daemonize():
    # Fork the process
    pid = os.fork()

    if pid > 0:
        sys.exit() # Exit from the parent process

    os.chdir("/")
    os.umask(0)
    os.setsid() # Detach from the controlling terminal

    # Close standard file descriptors
    sys.stdout.close()
    sys.stderr.close()
    sys.stdin.close()

    # Redirect standard file descriptors to /dev/null
    sys.stdout = open("/dev/null", "a+")
    sys.stderr = open("/dev/null", "a+")
    sys.stdin = open("/dev/null", "r")
```

```
def main():
    print("Hello Python")
    print("My PID: " + str(os.getpid()))

    count = 0
    while True:
        count += 1
        print("count: " + str(count))
        time.sleep(15)

    sys.exit(0)

if __name__ == "__main__":
    daemonize()
    main()
```

```
$ ./hello-python-4.py
$ ps aux | head -1
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
$ ps aux | grep hello
metaemb+ 6940  0.0  0.2  14688  5680 ?        Ss   05:48   0:00 /usr/bin/python ./hello-python-4.py
metaemb+ 7039  0.0  0.0   7452   544 pts/0    S+   05:53   0:00 grep --color=auto hello
$ kill 6940
$ ps aux | grep hello
metaemb+ 7041  1.0  0.0   7452   536 pts/0    S+   05:54   0:00 grep --color=auto hello
```



# Summary

- Systems Programs in three languages
  - Bash
  - C
  - Python
- Three Process Types
  - Foreground
  - Background
  - Daemon