Systems Software

Week 4: Orphans, Zombies and Deamons

Notes By: Jonathan McCarthy

Overview

- Orphan Processes
- Zombie Processes
- Daemons
- Steps to create a daemon
- Required Reading

Process Groups

- Every process running is part of a unique process group (**PGID**)
- When a process is created, it becomes a member of the group its parent is in.
- The first process member in the group sets the PGID to be equal to its PID.
- The first member in the group is referred to as the **Group Leader**

Process Groups – System Calls

- Useful Process Group commands (see man for more details)
  - getpgrp()
  - getpgid(0)
  - getpgid(PID)

- Shell Commands
  - To list the PGID with the ps command use the –j flag

Sessions and Session Groups

- All processes are grouped be sessions.
- These are linked to types of groups, eg. A user logs in, all processes the user has running are in its Session Group.
- When a user logs out, the kernel will terminate all processes in the users session group.
- The session's ID is the same as the pid of the process that created the session through the setsid() system call.
- This session is referred to as the Session Leader.
- The setsid() system call takes no parameters and will return the new session id.

Controlling Terminal

- Every session is associated with a terminal
- Processes in the session get Input and Output from the terminal
- A session is linked to a terminal (this is called the Controlling Terminal or Controlling tty)
- A terminal can be the controlling terminal for only one session at a time.
- An individual process disconnects from its controlling terminal when it calls setsid to become the leader of a new session.

Controlling Terminal

- One of the attributes of a process is its controlling terminal.
- Child processes created with fork inherit the controlling terminal from their parent process.
- In this way, all the processes in a session inherit the controlling terminal from the session leader.
- A session leader that has control of a terminal is called the *controlling process* of that terminal.
- An individual process disconnects from its controlling terminal when it calls setsid to become the leader of a new session.

Source: https://www.gnu.org/software/libc/manual/html_node/Controlling-Terminal.html

File Descriptor

- Each time a file is opened, the OS creates a record to represent the file.
- This information is stored in the system kernel.
- An integer value is assigned to each record/entry.
- This integer value is the file descriptor. If a process has opened 5 files, there will be 5 file descriptors associated with the process.

Orphan Processes

- An orphan process is a process whose parent has terminated.
- The orphan will default back to init as its parent (PPID of 1).
- The orphan will continue to run until it is killed:
  - Kill -9 PID
  - Where PID is the process ID of the orphan

Example

Open ▾ *orphan.c
~/Documents/Apps/week4

```c
// Orphan Example
// The child process is adopted by init process, when parent process dies.
#include<stdio.h>
#include <stdlib.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0) {
        // if PID > 0 :: this is the parent
        // this process performs printf and finishes
        printf("Parent process");
        sleep(10);  // uncomment to wait 10 seconds before process ends
        exit(EXIT_SUCCESS); // Kill the parent
    } else if (pid == 0) {
       printf("Child process");

       // Keep process running with infinite loop
       // When the parent finishes after 10 seconds,
       // the getppid() will return 1 as the parent (init process)
       while(1) {
          sleep(1);
           printf("child 1: my parent is: %i\n", getppid());
       }
    }

    return 0;
}
```

Run Orphan

jmccarthy@debianJMC2017: ~/Docume

File Edit View Search Terminal Help

```
$./orphan
```

jmccarthy@debianJMC2017: ~/Doc ts/Apps/wee

File Edit View Search Terminal Help

```
$./orphan
Child processchild 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
```

jmccarthy@debianJMC2017: ~/Documents/Apps/w

File Edit View Search Terminal Help

```
child 1: my parent is: 5308
child 1: my parent is: 5308
child 1: my parent is: 5308
Parent process$child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
child 1: my parent is: 1
```

Run Orphan

jmccarthy@debianJMC2017: ~/Documents/Apps/week4

File Edit View Search Terminal Help

```
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
```

jmccarthy@debianJMC2017: ~/Documents/Apps/week4

File Edit View Search Terminal Help

```
 1398  1499  1499  1499 pts/0     5342 Ss    1000   0:00 bash
    1  1536  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/dconf/dconf-se
 1145  2178   968   968 ?           -1 Sl    1000   7:17 firefox-esr
 1398  2458  2458  2458 pts/1     5346 Ss    1000   0:00 bash
    1  3996  1046  1046 ?           -1 Sl    1000   0:15 /usr/bin/gedit --gappli
 1398  4122  4122  4122 pts/2     4122 Ss+   1000   0:00 bash
 1499  5342  5342  1499 pts/0     5342 S+    1000   0:00 ./orphan
 5342  5343  5342  1499 pts/0     5342 S+    1000   0:00 ./orphan
 2458  5346  5346  2458 pts/1     5346 R+    1000   0:00 ps -xj
jmccarthy@debianJMC2017:~/Documents/Apps/week4$
```

jmccarthy@debianJMC2017: ~/Documents/Apps/week4

File Edit View Search Terminal Help

```
 1398  1498  1046  1046 ?           -1 S     1000   0:00 gnome-pty-helper
 1398  1499  1499  1499 pts/0     1499 Ss+   1000   0:00 bash
    1  1536  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/dconf/dconf-se
 1145  2178   968   968 ?           -1 Sl    1000   7:17 firefox-esr
 1398  2458  2458  2458 pts/1     5348 Ss    1000   0:00 bash
    1  3996  1046  1046 ?           -1 Sl    1000   0:15 /usr/bin/gedit --gappli
 1398  4122  4122  4122 pts/2     4122 Ss+   1000   0:00 bash
    1  5343  5342  1499 pts/0     1499 S     1000   0:00 ./orphan
 2458  5348  5348  2458 pts/1     5348 R+    1000   0:00 ps -xj
jmccarthy@debianJMC2017:~/Documents/Apps/week4$
```

Parent terminates after 10 seconds

Note the PPID of the orphan process has changed to 1

Zombie Processes

- When a process terminates, it isn't removes straight away from memory.
- The process status becomes EXIT_ZOMBIE and its parent is notified via SIGCHLD signal.
- The parent should execute the wait() system call to read the child processes exit status.
- Once the parent has processed this information from the child, the terminated process can be removed from memory.
- If the parent is unable to process this information the child process becomes a Zombie Process.

Zombie Example

Open ▾ zombie.c
~/Documents/Apps/w

```c
// A C program to demonstrate Zombie Process.
// Child becomes Zombie as parent is sleeping
// when child process exits.
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0) {
        sleep(100);
    } else {  // Child process
      exit(1);
    }
    return 0;
}
```

Zombie Example

jmccarthy@debianJMC2017: ~/Documents/Apps/week4

File Edit View Search Terminal Help

```
 968  1233   968   968 ?           -1 SNl   1000   0:00 /usr/lib/tracker/tracke
   1  1238  1046  1046 ?           -1 Sl    1000   0:01 /usr/lib/tracker/tracke
1215  1239  1046  1046 ?           -1 S     1000   0:00 /bin/cat
   1  1265  1046  1046 ?           -1 S     1000   0:00 /usr/lib/x86_64-linux-g
   1  1332  1046  1046 ?           -1 Sl    1000   0:05 /usr/bin/nautilus --gap
   1  1338  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/gvfs/gvfsd-tra
   1  1347  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/gvfs/gvfsd-bur
   1  1359  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/gvfs/gvfsd-met
   1  1398  1046  1046 ?           -1 Sl    1000   0:21 /usr/lib/gnome-terminal
1398  1498  1046  1046 ?           -1 S     1000   0:00 gnome-pty-helper
1398  1499  1499  1499 pts/0     5510 Ss    1000   0:00 bash
   1  1536  1046  1046 ?           -1 Sl    1000   0:00 /usr/lib/dconf/dconf-se
1145  2178   968   968 ?           -1 Sl    1000   7:20 firefox-esr
1398  2458  2458  2458 pts/1     5513 Ss    1000   0:00 bash
   1  3996  1046  1046 ?           -1 Sl    1000   0:18 /usr/bin/gedit --gappli
1398  4122  4122  4122 pts/2     4122 Ss+   1000   0:00 bash
1499  5510  5510  1499 pts/0     5510 S+    1000   0:00 ./zombie
5510  5511  5510  1499 pts/0     5510 Z+    1000   0:00 [zombie] <defunct>
2458  5513  5513  2458 pts/1     5513 R+    1000   0:00 ps -xj
jmccarthy@debianJMC2017:~/Documents/Apps/week4$
```

- Note the Process state codes (Z+)
  - Defunct ("zombie") process, terminated but not reaped by its parent.

Daemons

- A Daemon is a process.
- It runs in the background not under the control of a user.
- A daemon has a PID of 1. They are usually started when the system is booted and will only terminate on shutdown.
- Example: crond, ftpd, rlogind, mysqld, apache

Steps to create a daemon

- Step 1: Create the orphan process
- Step 2: Elevate the orphan process to session leader, to loose controlling TTY
- Step 3: call umask() to set the file mode creation mask to 0 This will allow the daemon to read and write files with the permissions/access required
- Step 4: Change the current working dir to root. This will eliminate any issues of running on a mounted drive, that potentially could be removed etc..
- Step 5: Close all open file descriptors

Step 1: Create the orphan process

Open

step1.c

~/Documents/Apps/week4/ClassExample

```c
// Orphan Example
// The child process is adopted by init process, when parent process dies.
#include<stdio.h>
#include <stdlib.h>

int main()
{
    // Implementation for Singleton Pattern if desired (Only one instance running)

    // Create a child process
    int pid = fork();

    if (pid > 0) {
        // if PID > 0 :: this is the parent
        // this process performs printf and finishes
        printf("Parent process");
        sleep(10);  // uncomment to wait 10 seconds before process ends
        exit(EXIT_SUCCESS); // Kill the parent, needed to make orphan
    } else if (pid == 0) {
       // Step 1: Create the orphan process
       printf("Child process");

       // Orphan Logic goes here!!
       // Keep process running with infinite loop
       // When the parent finishes after 10 seconds,
       // the getppid() will return 1 as the parent (init process)
       while(1) {
          sleep(1);
          printf("child 1: my parent is: %i\n", getppid());
       }
    }

    return 0;
}
```

fork() to create an orphan. The orphan will not be a group leader.

The orphan will have the GPID of the parent process.

This is important, the next step will fail of the process is a Group Leader.

Step 2: Elevate the orphan process to session leader

```
// Step 2: Elevate the orphan process to session leader, to loose controlling TTY
// This command runs the process in a new session
if (setsid() < 0) { exit(EXIT_FAILURE); }
```

- the setsid() system call is used to get the process to be the session group leader and process group leader.
- Call setsid() to get the process to become a process group leader and session group leader.
- The new session will not be associated with a controlling terminal

Step 3: call umask() to set the file mode creation mask to 0

```
// Step 3: call umask() to set the file mode creation mask to 0
// This will allow the daemon to read and write files
// with the permissions/access required
umask(0);
```

- umask(0) so that we have the abilite to read and write files etc. If this is not done correctly the daemon may not have the appropriate permissions to access files.

Step 4: Change the current working dir to root.

```
// Step 4: Change the current working dir to root.
// This will eliminate any issues of running on a mounted drive,
// that potentially could be removed etc..
if (chdir("/") < 0 ) { exit(EXIT_FAILURE); }
```

- chdir("/") to ensure that our process doesn't keep any directory in use.

Step 5: Close all open file descriptors

```
// Step 5: Close all open file descriptors
/* Close all open file descriptors */
int x;
for (x = sysconf(_SC_OPEN_MAX); x>=0; x--)
{
   close (x);
}
```

- close() fds 0, 1, and 2.
- We are trying to release the IO connections inherited from the parent.
- use sysconf() to determine the limit _SC_OPEN_MAX. This gives the max value of file descriptors (int value).
- Create a loop and close all open file descriptors.

Process State Codes

```
PROCESS STATE CODES
       Here are the different values that the s, stat and state output
       specifiers (header "STAT" or "S") will display to describe the state of
       a process.
       D    Uninterruptible sleep (usually IO)
       R    Running or runnable (on run queue)
       S    Interruptible sleep (waiting for an event to complete)
       T    Stopped, either by a job control signal or because it is being
            traced.
       W    paging (not valid since the 2.6.xx kernel)
       X    dead (should never be seen)
       Z    Defunct ("zombie") process, terminated but not reaped by its
            parent.
```

PS

- PPID – Parent Process ID
- PID – Process ID
- PGID – Process Group ID
- SID – Session ID
- TTY – Controlling Terminal
- TPGID – Controlling tty process group ID
- STAT – Process Status Codes
- UID – Effective User Id

Process Status Codes

PROCESS STATE CODES

Here are the different values that the **s**, **stat** and **state** output specifiers (header "STAT" or "S") will display to describe the state of a process:

```
D    uninterruptible sleep (usually IO)
R    running or runnable (on run queue)
S    interruptible sleep (waiting for an event to complete)
T    stopped, either by a job control signal or because it is
     being traced
W    paging (not valid since the 2.6.xx kernel)
X    dead (should never be seen)
Z    defunct ("zombie") process, terminated but not reaped by
     its parent
```

Error Logging

- How can a daemon deal with error messages?
- It doesn’t have access to standard_error, as it doesn’t have a controlling terminal.
- One option could be to write the error messages to a file, but this could get messy if there were loads of daemons writing to loads of files.
- A central solution would be best.

Error Logging

- Most daemons use the syslog function to generate log messages.

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

NAME
       closelog, openlog, syslog, vsyslog - send messages to the system logger

SYNOPSIS
       #include <syslog.h>

       void openlog(const char *ident, int option, int facility);
       void syslog(int priority, const char *format, ...);
       void closelog(void);

       #include <stdarg.h>

       void vsyslog(int priority, const char *format, va_list ap);
```

Walkthrough Example – In Class Demo

Run……

Required Reading

- **How To Write a UNIX Daemon by Dave Lennert** (Hewlett-Packard Company)
  - http://cjh.polyplex.org/software/daemon.pdf

- **Advanced Programming in the Unix Environment** – Second Edition – W. Richard Stevens and Stephen A. Rago.
  - **Chapter 13.** Daemon Processes

Questions