Daemon Processes

Introduction

- A ***daemon*** is a process that runs in the background and is independent of control from all terminals.
- There are numerous ways to start a daemon
    1. the system initialization scripts (`/etc/rc`)
    2. the *inetd* superserver
    3. *cron* deamon
    4. the *at* command
    5. 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.

ps command

Execute ps –axj command

- -a option shows the status of processes owned by others,
- -x shows processes that don't have a controlling terminal.
- -j option displays the job-related information: the session ID, process group ID, controlling terminal, and terminal process group ID.

Coding rules

- Some basic rules to coding a daemon prevent unwanted interactions from happening. We state these rules and then show a function, daemonize, that implements them.

1) The first thing to do is call umask to set the file mode creation mask to 0.

   The file mode creation mask that's inherited could be set to deny certain permissions.

   If the daemon process is going to create files, it may want to set specific permissions.

   For example, if it specifically creates files with group-read and group-write enabled, a file mode creation mask that turns off either of these permissions would undo its efforts.

Coding rules

2. Call fork and have the parent exit. This does several things.

First, if the daemon was started as a simple shell command, having the parent terminate makes the shell think that the command

is done.

Second, the child inherits the process group ID of the parent but gets a new process ID, so we're guaranteed that the child is not a process group leader. This is a prerequisite for the call to setsid that is done next.

3. Call setsid to create a new session. The process

(a) becomes a session leader of a new session,

(b) becomes the process group leader of a new process group,

(c) has no controlling terminal.

Coding rules

4. Change the current working directory to the root directory.

The current working directory inherited from the parent could be on a mounted file system. Since daemons normally exist until the system is rebooted, if the daemon stays on a mounted file system, that file system cannot be unmounted.

Alternatively, some daemons might change the current working directory to some specific location, where they will do all their work.

For example, line printer spooling daemons often change to their spool directory.

Coding rules

5. Unneeded file descriptors should be closed.

This prevents the daemon from holding open any descriptors that it may have inherited from its parent (which could be a shell or some other process). We can use our open_max function or the getrlimit function to determine the highest descriptor and close all descriptors up to that value.

6. Some daemons open file descriptors 0, 1, and 2 to /dev/null so that any library routines that try to read from standard input or write to standard output or standard error will have no effect.

Since the daemon is not associated with a terminal device, there is nowhere for output to be displayed; nor is there anywhere to receive input from an interactive user. Even if the daemon

Error Logging

- One problem a daemon has is how to handle error messages. It can't simply write to standard error, since it shouldn't have a controlling terminal.
- We don't want all the daemons writing to the console device, since on many workstations, the console device runs a windowing system.
- We also don't want each daemon writing its own error messages into a separate file. It would be a headache for anyone
- administering the system to keep up with which daemon writes to which log file and to check these files on a regular basis.
- A central daemon error-logging facility is required.
- The BSD syslog facility was developed at Berkeley and used widely in 4.2BSD. Most systems derived from BSD support syslog.

Error Logging

There are three ways to generate log messages:

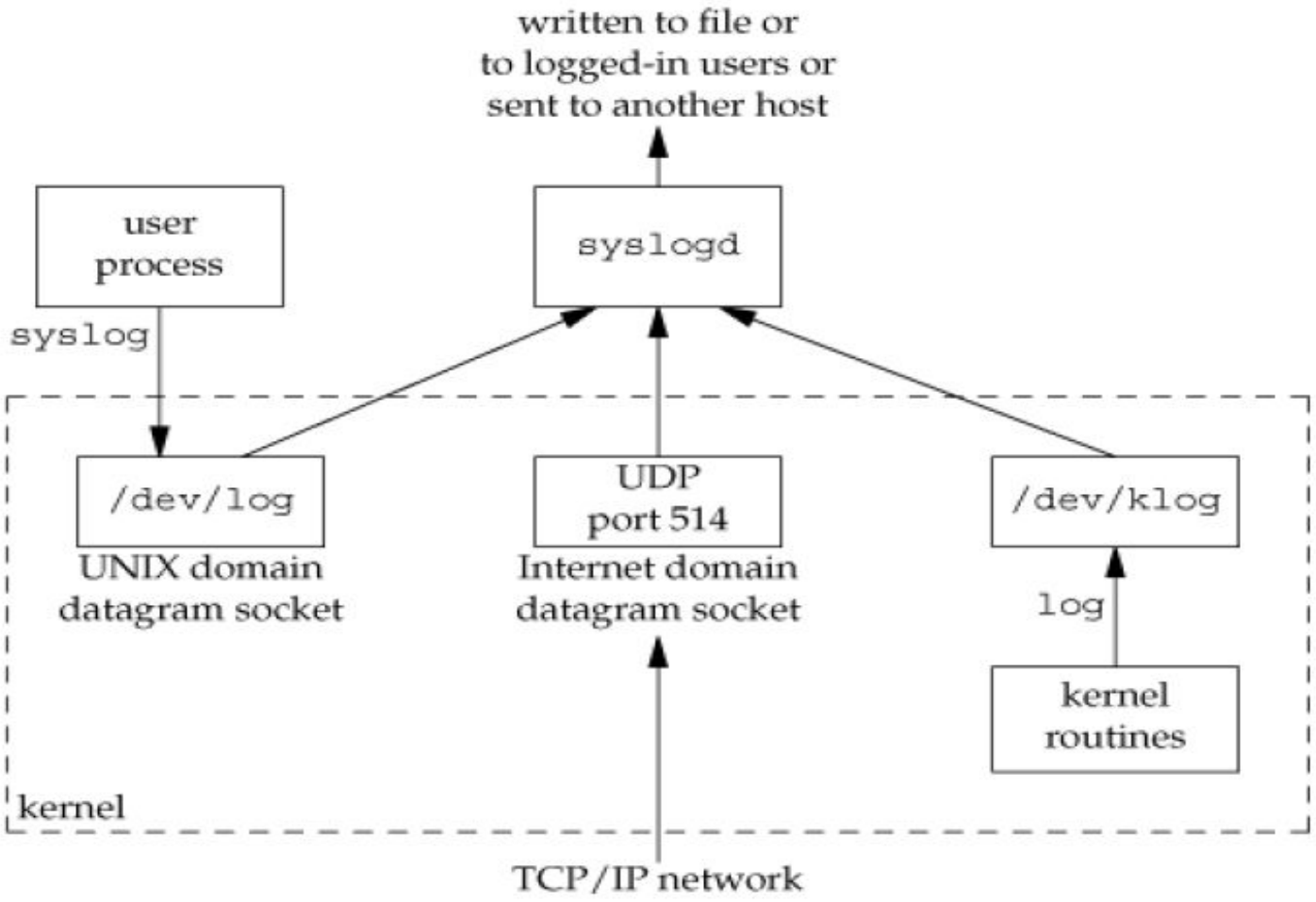
1. Kernel routines can call the log function. These messages can be read by any user process that opens and reads the /dev/klog device.
2. 2. Most user processes (daemons) call the syslog(3) function to generate log messages. This causes the message to be sent to the UNIX domain

datagram socket /dev/log.

3. A user process on this host, or on some other host that is connected to this host by a TCP/IP network, can send log messages to UDP port 514.

Note that the syslog function never generates these UDP datagrams: they require explicit network programming by the process generating the log message.

The BSD syslog facility



Error Logging

Normally, the syslogd daemon reads all three forms of log messages.

On start-up, this daemon reads a configuration file, usually /etc/syslog.conf, which determines where different classes of messages are to be sent. For example, urgent messages can be sent to the system administrator (if logged in) and printed on the console, whereas warnings may be logged to a file.

Our interface to this facility is through the syslog function.

Error Logging

```
#include <syslog.h>

void openlog(const char *ident, int option, int
➥facility);

void syslog(int priority, const char *format, ...);

void closelog(void);

int setlogmask(int maskpri);
```

Returns: previous log priority mask value

Syslog routines

- Calling openlog is optional. If it's not called, the first time syslog is called, openlog is called automatically.
- Calling closelog is also optional, it just closes the descriptor that was being used to communicate with the syslogd daemon.
- Calling openlog lets us specify an *ident that is added to each log message. This is normally the name* of the program (cron, inetd, etc.).
- The *option argument is a bitmask specifying various options.*

option argument in syslog

option	Description
LOG_CONS	If the log message can't be sent to syslogd via the UNIX domain datagram, the message is written to the console instead.
LOG_NDELAY	Open the UNIX domain datagram socket to the syslogd daemon immediately; don't wait until the first message is logged. Normally, the socket is not opened until the first message is logged.
LOG_NOWAIT	Do not wait for child processes that might have been created in the process of logging the message. This prevents conflicts with applications that catch SIGCHLD, since the application might have retrieved the child's status by the time that syslog calls wait.
LOG_ODELAY	Delay the open of the connection to the syslogd daemon until the first message is logged.
LOG_PERROR	Write the log message to standard error in addition to sending it to syslogd. (Unavailable on Solaris.)
LOG_PID	Log the process ID with each message. This is intended for daemons that fork a child process to handle different requests (as compared to daemons, such as syslogd, that never call fork).

facility argument in syslog

facility		Description
LOG_AUTH		authorization programs: login, su, getty, ...
LOG_AUTHPRIV		same as LOG_AUTH, but logged to file with restricted permissions
LOG_CRON		cron and at
LOG_DAEMON		system daemons: inetd, routed, ...
LOG_FTP		the FTP daemon (ftpd)
LOG_KERN		messages generated by the kernel
LOG_LOCAL0		reserved for local use
LOG_LOCAL1		reserved for local use
LOG_LOCAL2		reserved for local use
LOG_LOCAL3		reserved for local use
LOG_LOCAL4		reserved for local use
LOG_LOCAL5		reserved for local use
LOG_LOCAL6		reserved for local use
LOG_LOCAL7		reserved for local use

Facility argument in syslog & syslog levels

facility		Description
LOG_LPR		line printer system: `lpd`, `lpc`, ...
LOG_MAIL		the mail system
LOG_NEWS		the Usenet network news system
LOG_SYSLOG		the `syslogd` daemon itself
LOG_USER		messages from other user processes (default)
LOG_UUCP		the UUCP system

The `syslog` *levels* (ordered)

level	Description
LOG_EMERG	emergency (system is unusable) (highest priority)
LOG_ALERT	condition that must be fixed immediately
LOG_CRIT	critical condition (e.g., hard device error)
LOG_ERR	error condition
LOG_WARNING	warning condition
LOG_NOTICE	normal, but significant condition
LOG_INFO	informational message
LOG_DEBUG	debug message (lowest priority)

ClientServer Model

- A common use for a daemon process is as a server process.
- In general, a *server is a process that waits for a client to contact it, requesting some type of service.*
- For example, the service being provided by the syslogd server is the logging of an error message.
- The communication between the client and the server could be one-way. The client sends its service request to the server; the server sends nothing back to the client.
- Two-way communication between a client and a server is also possible. The client sends a request to the server, and the server sends a reply back to the client.