Chapter 13
Daemon Processes

Cheng-Hsin Hsu
National Tsing Hua University
Department of Computer Science

Parts of the course materials are courtesy of Prof. Chun-Ying Huang

Outline

- Introduction
- Daemon conventions
- Client-server model
- Coding rules
- Error logging
- Single-instance daemon

Introduction

- Daemons are processes that live for a long time
- They are often started when the system is bootstrapped and terminate only when the system is shut down
- They don’t have a controlling terminal – They run in background
- Partial output from 'ps ajx' command
  - The 'x’ option also shows processes without a terminal
  - The 'j’ option shows job related information
  - Process 1 is usually the init process
  - Processes enclosed by brackets [] are kernel processes

Daemon Conventions

- If the daemon uses a lock file, the file is usually stored in /var/run/name.pid
  - The lock file is often used to check the existence of a running daemon
  - The daemon might need superuser permissions to create a file here
- If the daemon supports configuration options, they are usually stored in /etc
  - The configuration file is often named name.conf
- Daemons can be started from the command line
  - They are usually started from one of the system initialization scripts (/etc/rc* or /etc/init.d/*)
- If a daemon has a configuration file, the daemon reads it when it starts, but usually won't look at it again
  - Some daemons will catch SIGHUP and reread their configuration files when they receive the signal

Client-Server Model

- A common use for a daemon process is as a server process
- For example, the syslog library call (client) and the syslogd daemon (server)
  - The connection between the client and the server is created by using UNIX domain datagram socket
- In general, a server is a process that waits for a client to contact it, requesting some types of service
- The communication channel between a client and a server can be one-way or two-way

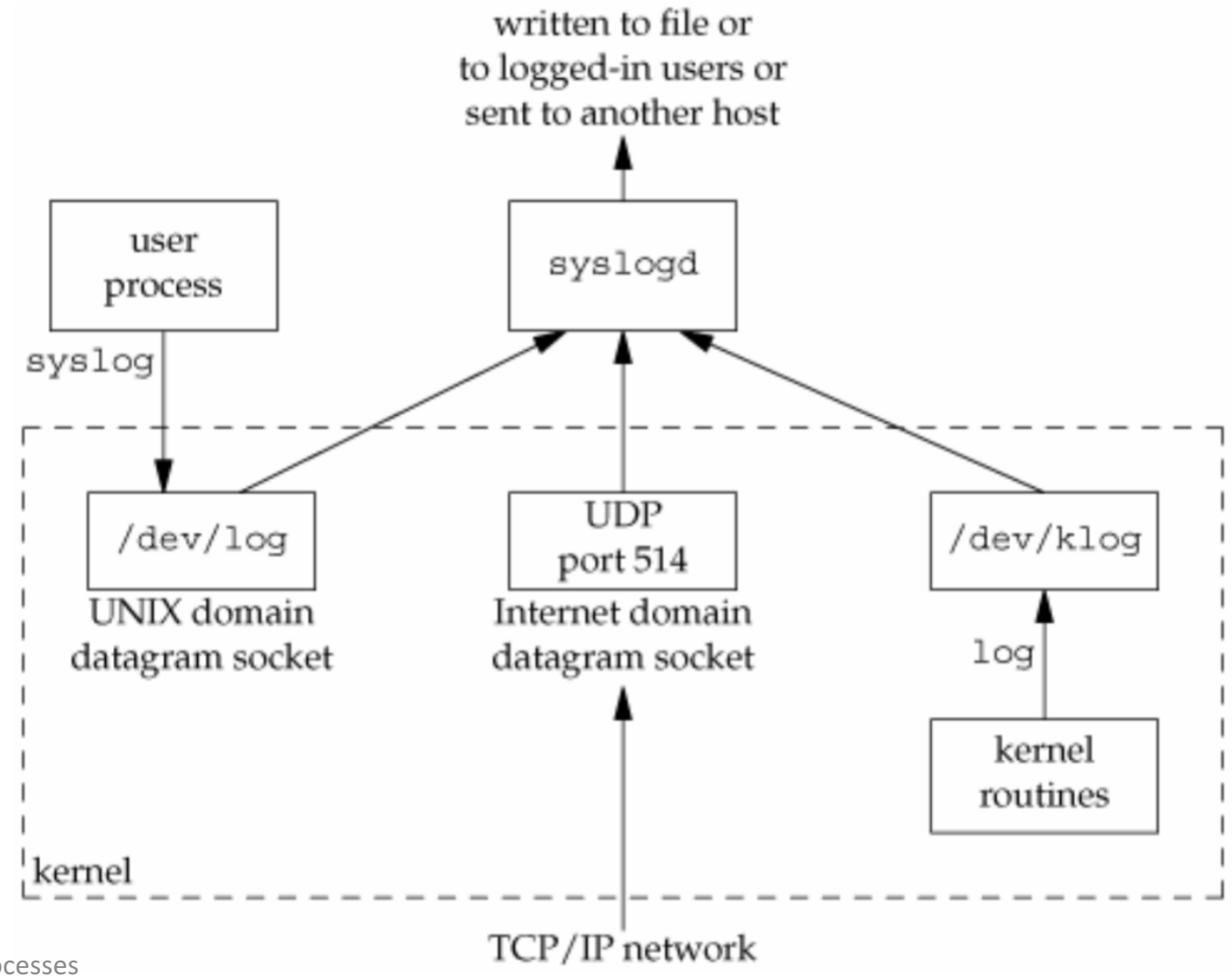
Coding Rules

- Use umask to reset permission masks
- Call fork at parent and then exit (the parent process)
  - Will not block the shell
  - Will not be a process group leader and can be a session leader
- Call setsid
  - Becomes a session leader and a process group leader
  - Has no controlling terminal
  - System V systems suggest to call fork again and terminate the parent – to prevent the child process from acquiring controlling terminal
- Change the current working directory, may be to the root directory
- Close unused file descriptors, and redirect descriptors {0, 1, 2} to /dev/null
- Setup log files or log systems to store output from the daemon

Error Logging

- A daemon process cannot write error message to standard error
  - It does not have a controlling terminal
- A daemon should not write messages to the console
- Administrators may prefer to have a centralized place to collect logs from all daemon process
- The syslog facility initially proposed in the BSD system
- Many systems derived the design from syslogd, e.g., Linux's rsyslogd (reliable and extended syslogd)

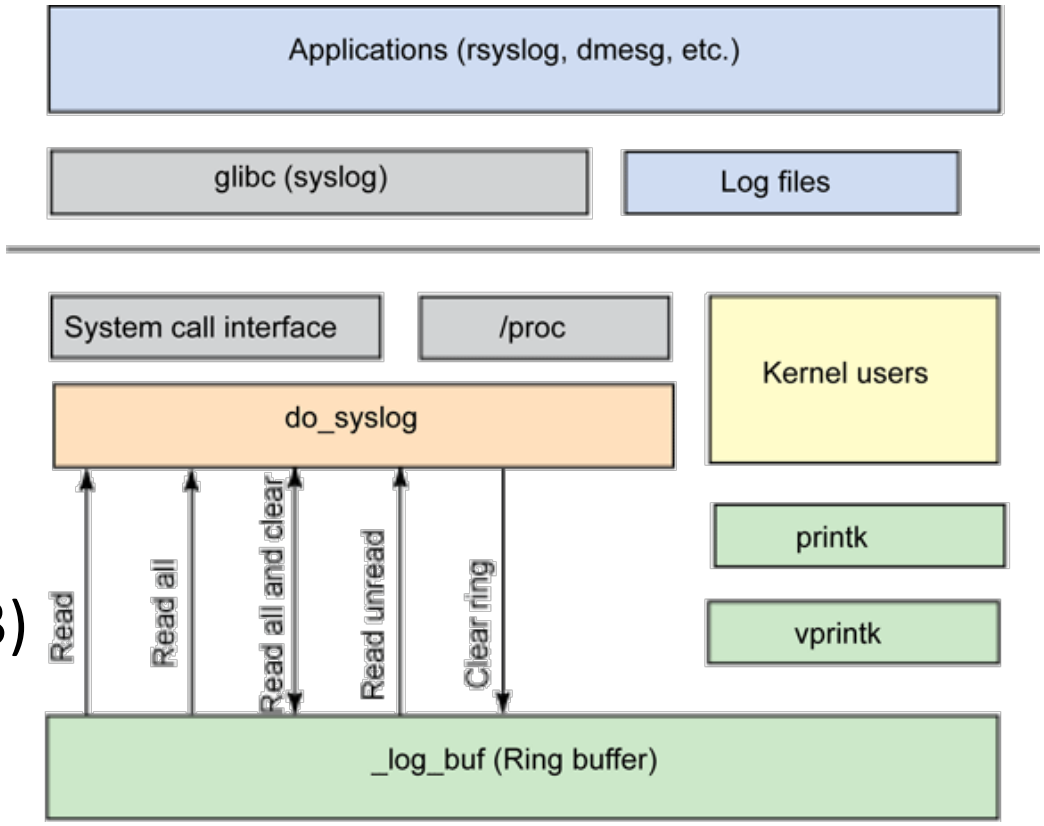
The BSD syslog Facility



Linux Kernel Logging Stack

Source: http://www.ibm.com/developerworks/library/l-kernel-logging-apis/

- Note: syslog(2) and syslog(3) are different
  - syslog(2): Read and clear the kernel log buffer
  - syslog(3): Write log to log system
- What we introduce later will be syslog(3) in glibc



The syslog Interface

- Open, write to, and close the log system

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

- The call to openlog is optional
  - It will be called automatically on the first call to syslog
  - You can specify an ident string to the log system
  - You can set option and facility: discuss later
- The call to closelog is also optional – It simply closes the descriptor ← why this is enough?

openlog Options

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.
LOG_PID	Log the process ID with each message. This is intended for daemons that fork a child process to handle different requests .

openlog Facility

- Can be used to distinguish the source of logs
- May be not available on all systems

Facility	Description	Facility	Description
LOG_AUTH	authorization programs, e.g., login, su, getty, …	LOG_LPR	line printer system: lpd, lpc
LOG_AUTHPRIV	same as LOG_AUTH, but logged to file with restricted permissions	LOG_MAIL	the mail system
LOG_CRON	cron and at	LOG_NEWS	The Usenet network news system
LOG_DAEMON	system daemons: inetd, routed, …	LOG_SYSLOG	the syslogd itself
LOG_FTP	FTP daemon (ftpd)	**LOG_USER**	message from other user processes (default)
LOG_KERN	messages generated by the kernel	LOG_UUCP	the UUCP system
		LOG_LOCAL0 ~ LOG_LOCAL7	reserved for local use

syslog Priority

- The priority argument is a combination of the openlog facility and the following log levels
- Ordered from highest to lowest

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 signification condition
LOG_INFO	informational message
LOG_DEBUG	debug message (lowest priority)

openlog/syslog Example

- An example for line printer spooler daemon

```
openlog("lpd", LOG_PID, LOG_LPR);
syslog(LOG_ERR, "open error for %s: %m", filename);
```

  - The %m is used to print the corresponding error string (of errno)
- A similar implementation

```
syslog(LOG_ERR | LOG_LPR, "open error for %s: %m", filename);
```

Single-Instance Daemons

- Some daemons are implemented so that only a single copy of the daemon should be running at a time for proper operation
- The daemon might need exclusive access to a device
  - For example: the cron daemon
- Sample codes to check the existence of a running daemon
  - Based on the pid file for the daemon

Single-Instance Daemons: Check Running Daemons

```
int already_running(void) {
    int fd;
    char buf[16];
    if((fd = open(LOCKFILE, O_RDWR|O_CREAT, LOCKMODE)) < 0) {
        syslog(LOG_ERR, "can't open %s: %s", LOCKFILE, strerror(errno));
        exit(1);
    }
    if (lockfile(fd) < 0) {  /* lockfile: introduce in the next chapter */
        if (errno == EACCES || errno == EAGAIN) {
            close(fd);
            return 1;
        }
        syslog(LOG_ERR, "can't lock %s: %s", LOCKFILE, strerror(errno));
        exit(1);
    }
    ftruncate(fd, 0);
    sprintf(buf, "%ld\n", (long) getpid());
    write(fd, buf, strlen(buf));
    return 0;
}
```

No Assignment this Time

The assignment will be combined with the next Chapter.