Daemons

- Daemon Processes
- Some Traditional System Daemons
- Creating a Daemon
- Starting Up a Daemon
- Server Organization
- Concurrent Server
- Error Logging
- BSD `syslog` Facility
- References

Daemon Processes

- A long running autonomous process which provides some service is called a *daemon* process.
- Daemon processes typically do not have a controlling terminal and run in their own process group and session.
- Since daemons do not have a controlling terminal, SIGHUP can never be generated for them by their controlling terminal. Instead, SIGHUP is traditionally overloaded for daemons: when a daemon receives a SIGHUP it rereads its configuration files.

Some Traditional System Daemons

```
$ ps -ax -o ppid,pid,pgid,sid,tt,tpgid,uid,comm
 PPID   PID  PGID   SID TT       TPGID   UID COMMAND
    0     1     0     0 ?           -1     0 init
    1     2     1     1 ?           -1     0 kflushd
    1     3     1     1 ?           -1     0 kpiod
    1     4     1     1 ?           -1     0 kswapd
    1   269   269   269 ?           -1     0 dhcpcd
    1   296   296   296 ?           -1     0 syslogd
    1   307   307   307 ?           -1     0 klogd
    1   321   114   114 ?           -1     2 atd
    1   335   335   335 ?           -1     0 crond
    1   349   349   349 ?           -1     0 inetd
    1   363   363   363 ?           -1     0 lpd
    1   399   399   399 ?           -1     0 sendmail
  349  4508   349   349 ?           -1     0 in.telnetd
    1 23434 23434 23434 ?           -1     0 fetchmail
  349 14766   349   349 ?           -1     0 in.telnetd
    1   433   433   433 ?           -1     0 httpd
  433 26117   433   433 ?           -1    99 httpd
  433 26118   433   433 ?           -1    99 httpd
  433 26119   433   433 ?           -1    99 httpd
  433 26120   433   433 ?           -1    99 httpd
$
```

Creating a Daemon

1. `fork()` with parent exiting. Child will become daemon. Child isn't a process group leader.
2. Call `setsid()`: no controlling terminal, session leader of a new session and group leader of a new group.
3. `chdir()` to either '/' or a particular working directory.
4. Optionally do `umask()`.
5. Close off unneeded file descriptors.

Starting Up a Daemon

```
int
daemon_init(void)
{
  pid_t pid;
  if ((pid= fork()) < 0) {
    return -1;
  }
  else if (pid != 0) { /* parent */
    exit(0);
  }
  else { /* child becomes daemon */
    setsid();
    chdir("/");
    umask(0);
  }
  return 0;
}
```

Typically, a daemon will first gather its command arguments, do general initialization and then call the above routine. Once it returns, the daemon is ready to do its stuff.

Server Organization

Simple server algorithm:

1. Server opens a well-known IPC or network facility.
2. Server enters a loop in which it reads requests and responds to them.

This organization is what is used by a *iterative* server.

Deficiencies in above algorithm:

- If a client requests a 200 MB file from a file server, then other clients are locked out while the first request is being processed.
- If the server crashes while processing a request, then all future clients are denied service, until the server is (manually?) restarted.

Concurrent Server

- Multiple requests are processed concurrently, usually by using concurrent processes.
- No serialization of requests.
- If multiple processes are used, then possibility of greater reliability.
- With multiple processes, it is possible to `exec()` another program to do the actual work (strategy used by `inetd` Internet super-server).

Error Logging

- Since a daemon doesn’t have a terminal, it can’t write error messages to the terminal.
- It could write errors, to its own log file, but such a situation can lead to chaos with a system administrator having to examine multiple log files.
- Use a centralized error logging daemon.

BSD `syslog` Facility

- Different sources of log messages: kernel routines; user processes; network (via UDP port 514).
- Supports different priorities of log messages: `LOG_EMERG`, `LOG_ALERT`, `LOG_CRIT`, `LOG_ERR`, `LOG_WARNING`, `LOG_NOTICE`, `LOG_INFO`, `LOG_DEBUG`.
- On startup, `syslogd` reads `/etc/syslogd.conf` which can redirect different priorities of log messages.
- Basic API:

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

- Example: (`%m` replaced by `strerror(errno)`):

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

References

Text, Ch. 37.

APUE, Ch. 13.