【Linux Programming】Day19

Tags	
Date	@June 11, 2022
Summary	Logging and Resources and Limits

【Ch4】Work with Files

4.7 Logging

System programs often write messages to the console, or a log file. These messages might indicate errors, warnings, or information about the state of the system.

The log messages are recorded in system files in `/usr/adm` or `/var/log`.

The Unix specification provides an interface for all programs to produce logging messages using the `syslog` function.

```
#include <syslog.h>

void syslog(int priority, const char *message, arguments...);
```

Each message has a priority argument that is a bitwise OR of a severity level and a facility value.

Facility values(from `syslog.h`) include `LOG_USER`, used to indicate the message comes from a user application, and `LOG_LOCAL0`, `LOG_LOCAL1`, up to `LOG_LOCAL7`.

The severity levels in descending order of priority are shown below.

Priority Level	Description
LOG_EMERG	An emergency situation
LOG_ALERT	High-priority problem, such as database corruption
LOG_CRIT	Critical error, such as hardware failure
LOG_ERR	Errors
LOG_WARNING	Warning
LOG_NOTICE	Special conditions requiring attention
LOG_INFO	Informational messages
LOG_DEBUG	Debug messages

The log message created by `syslog` consists of a message header and a message body. The header is created from the facility indicator and the date and time.

4.8 Resources and Limits

Programs running on Linux are subject to resource limitations.

The header file `limits.h` defines many manifest constants that represent the constraints imposed by the OS.

Limit Constant	Purpose
NAME_MAX	The maximum number of characters in a filename
CHAR_BIT	The number of bits in a `char` value
CHAR_MAX	The maximum `char` value
INT_MAX	The maximum `int` value

The header file `sys/resource.h` provides definitions for resource operations.

```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int priority);
int getrlimit(int resource, struct rlimit *r_limit);
```

```
int setrlimit(int resource, const struct rlimit *r_limit);
int getrusage(int who, struct rusage *r_usage);
```

`id_t` is an integral type used for user and group identifiers. The `rusage` structure is used to determine how much CPU time has been used by the current program. It must contain at least the following two members:

rusage Member	Description
`struct timeval ru_utime`	The user time used
`struct timeval ru_stime`	The system time used

The `timeval` structure is defined in `sys/time.h` and contains fields `tv_sec` and `tv_usec`, representing seconds and microseconds, respectively.

CPU time consumed by a program is separated into user time and system time.

The `getrusage` function writes CPU time information to the `rusage` structure pointed by the parameter `r_usage`.

The `who` parameter can be one of the following:

who Constant	Description
RUSAGE_SELF	Returns usage information about current program only.
RUSAGE_CHILDREN	Includes usage information of child processes as well.

Applications can determine and alter their priority with the `getpriority` and `setptiority` functions.

Note: ordinary users can only reduce the priorities of their program.

The `which` parameter specifies how the `who` parameter is to be treated.

which Parameter	Description
PRIO_PROCESS	who is a process identifier.
PRIO_PGRP	who is a process group.
PRIO_USER	who is a user identifier.

So, to determine the priority of the current process, we might call:

```
priority = getpriority(PRIO_PROCESS, getpid());
```

The default priority is 0. Positive priorities are used for background tasks that run when no other higher priority task is ready to run.

Negative priorities cause a program to run more frequently, taking a larger shared of the available CPU time.

The range of valid priorities is -20 to +20. The higher the numerical value, the lower the execution precedence.

`getpriority` returns a valid priority if successful or a -1 with `errno` set on error. Because -1 is itself a valid priority, `errno` should be set to zero before calling `getpriority` and checked that it's still zero on return.

Limits on system resources can be read and set by `getrlimit` and `setrlimit`. Both of these functions make use of a general-purpose structure, `rlimit`, to describe resource limits.

It's defined in `sys/resource.h` and has the following members.

rlimit Member	Description
rlim_t rlim_cur	The current, soft limit
rlim_t rlim_max	The hard limit

A number of system resources can be limited:

resource Parameter	Description
RLIMIT_CORE	The core dump file size limit, in bytes
RLIMIT_CPU	The CPU time limit, in seconds
RLIMIT_DATA	The data () segment limit, in bytes
RLIMIT_FSIZE	The file size limit, in bytes
RLIMIT_NOFILE	The limit on the number of open files
RLIMIT_STACK	The limit on stack size, in bytes
RLIMIT_AS	The limit on address space (stack and data), in bytes