Systems Programming

Make, Stdio, Proc, Limits

Overview

Last Time

- Chapter 7 – Memory
- Chapter 8 – User and groups

Readings for today

- Make - implementation
- Chapter 13 – I/O buffering
- Shells:
  - Basics: read command into doubly linked list
  - Shell variables, set command
  - background,
  - Substitutions: variable substitutions, pseudo filename completion, history substitution,
  - Simple I/O redirection
- Shell version 2: signals, pipes, command substitution

MAKE(1) LOCAL USER COMMANDS MAKE(1)

NAME

make - GNU make utility to maintain groups of programs

SYNOPSIS

```
make [ -f makefile ] [ options ] ... [ targets ] ...
```

WARNING

This man page is an extract of the documentation of GNU make. It is updated only occasionally, because the GNU project does not use nroff. For complete, current documentation, refer to the Info file make.info which is made from the Texinfo source file make.texi.

DESCRIPTION - The purpose of the make utility is to:

1) determine automatically which pieces of a large program need to be recompiled, and

2) issue the commands to recompile them.

Make Advantages

- `make` command saves time - both typing and recompilation
- The Makefile documents the dependencies and how to build the software. (and its manual pages as well)
- For distribution of software one can `make' and `make install' a package without knowing anything about how to exactly compile the program.

Makefiles - Make Specification Files

- Consists of a set of dependencies (target and dependent) and rules (how to create target)

- Definitions/Macros of the form
  - name=value

- Target Groups of the form:

```
target_1 : dependency list_1
<TAB>          cmdlist_1
target_2 : dependencylist_2
<TAB>  cmdlist_2
```

A Simple Makefile

- **Makefile Example**

```
prog: main.o routines.o
        cc -o prog main.o routines.o
```

- **Each command line starts with a \tab**

```
main.o: main.c defs.h
   cc -c main.c
routines.o: routines.c defs.h
   cc -c routines.c
```

Another Makefile Example

```
FILES = Makefile defs.h main.c
routines.c
OBJS = main.o routines.o
LIBES = -lm
CFLAGS = -g
LP = /fac/matthews/bin/p2c
INSTALL_DIR = /fac/matthews/bin

prog: main.o routines.o
    $(CC) $(CFLAGS) $(OBJS)
$(LIBES) -o prog

$(OBJS): defs.h
```

```
cleanup:
        -rm *.o
        -du

install: prog
    mv prog
$(INSTALL_DIR)

print: $(FILES)
    pr $? >
/tmp/manton
    $(LP) /tmp/manton
    touch print
    -rm /tmp/manton
```

Make Options

- To determine if we make a target we check the modification time of all its dependencies (things/files it depends on); if any is newer rebuild the target

- `make -n` : doin' nothing (dry-run)
- `make -k` : keep going when an error is found

- By default, it builds the first target listed in the makefile (common : `all`)

Make Implementation Algorithm

```
Procedure newest(target)
    If target is not in the target tree then
          If file exists
             return(modification_time)
          Else return(FAIL)
   Else
          min = modification_time of target
          Foreach child in the dependency list Do
                child_time = newest(child)
                If child_time < min Then
                  min = child_time
          End
          If min < modification_time of target
          Then
            build(target)
            min = now
          EndIf
   End
End
```

```
Begin {Main}
   Parse Specification File
   Build Dependency Tree
   newest(target)
End
```

Build(target)?

GNU Make

- http://www.gnu.org/software/make/
- ``Makefile conventions'' (147 k characters) of the GNU Coding Standards (147 k characters).
- Downloading GNU ftp server:
  - http://ftp.gnu.org/gnu/make/
- Documentation for Make
  - http://www.gnu.org/software/make/manual/
- Make tutorial
  - http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Make builtin Macros

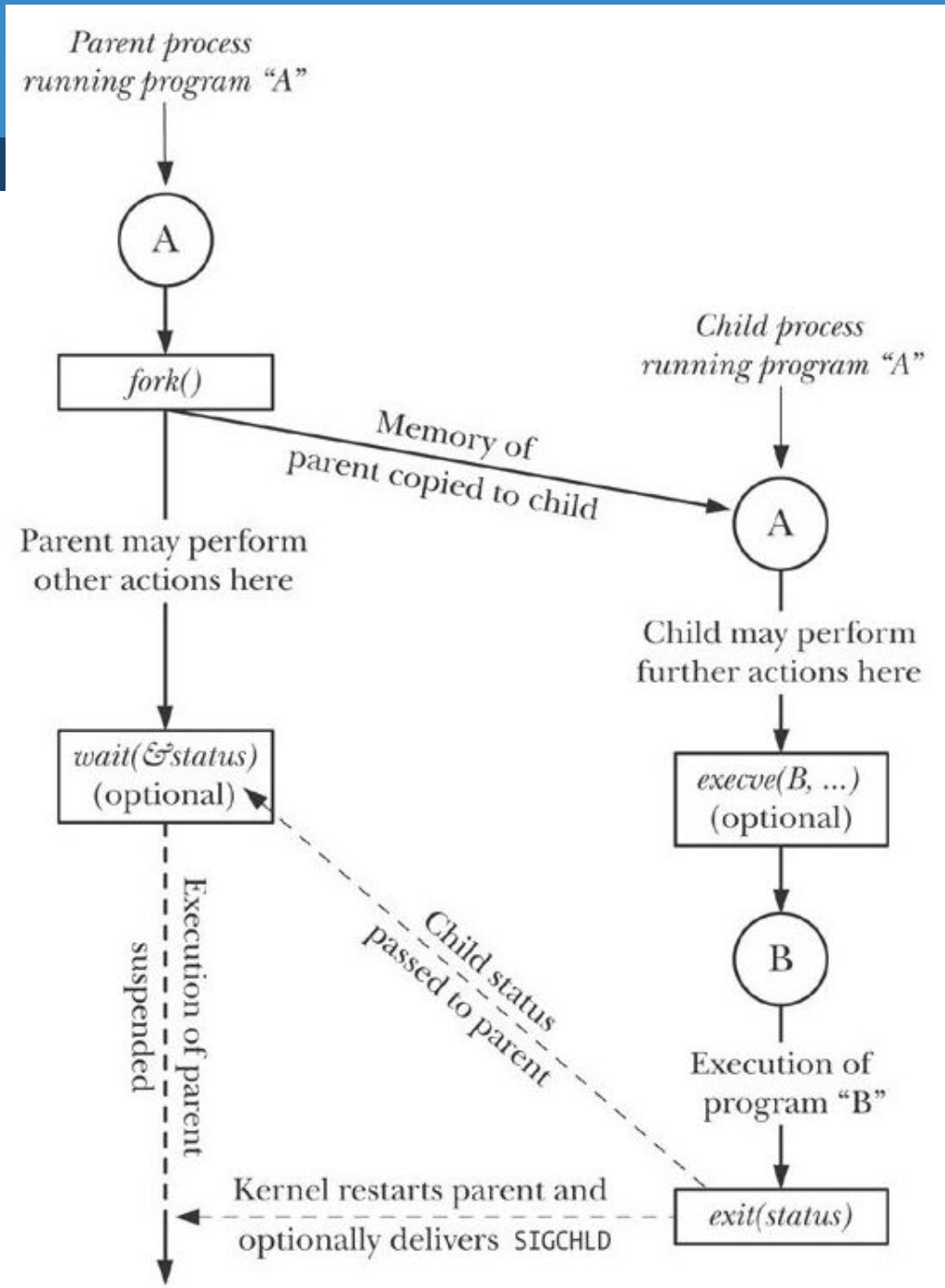
- $@ : Name of current target
- $? : the set of prerequisite that are younger than the target
- $< : the name of first prerequisite
- $* : the name of current prerequisite, without any suffix

Chapter 24 PROCESS CREATION

- 24.1 Overview of *fork()*, *exit()*, *wait()*, and *execve()*
- 24.2 Creating a New Process: *fork()*
- 24.2.1 File Sharing Between Parent and Child
- 24.2.2 Memory Semantics of *fork()*
- 24.3 The *vfork()* System Call
- 24.4 Race Conditions After *fork()*
- 24.5 Avoiding Race Conditions by Synchronizing with Signals
- 24.6 Summary

Fork/exec Fig24

- Fork()
- Exit(status)
- Wait(&status)
- Execve(B,…)



fork

```
#include <unistd.h>
pid_t fork(void);
```

- rv=fork()
- Returns pid of child in parent
  - Returns 0 in child
  - Returns -1 on error
- “the two processes are executing the same program text, but they have separate copies of the stack, data, and heap segments.”

Parent-child sharing

Other things that are the same

- Per process open files and File descriptors
- Some signals

Some are not the same

- Pid, ppid of course
- locks

Procexec/t_fork.c

```
/* t_fork.c - Demonstrate the use of fork(), showing that
parent and child get separate copies of stack and data
segments. */
#include "tlpi_hdr.h"

static int idata = 111; /* Allocated in data
segment */
int main(int argc, char *argv[]) {
    int istack = 222; /* Allocated in stack
segment */
    pid_t childPid;
```

```
 switch (childPid = fork()) {
    case -1:
        errExit("fork");
    case 0:
        idata *= 3;
        istack *= 3;
        break;
    default:
        sleep(3);  /* Give child a chance to execute */
        break;
    }
    /* Both parent and child come here */
 printf("PID=%ld %s idata=%d istack=%d\n", (long) getpid(),
            (childPid == 0) ? "(child) " : "(parent)",
idata, istack);
    exit(EXIT_SUCCESS);
}
```

t_execve.c

```
int
main(int argc, char *argv[])
{
    char *argVec[10]; /* Larger than
required */
    char *envVec[] = { "GREET=salut",
"BYE=adieu", NULL };

    if (argc != 2 || strcmp(argv[1],
"--help") == 0)
        usageErr("%s pathname\n",
argv[0]);

    /* Create argument list for the new
program */

    argVec[0] = strrchr(argv[1], '/');
/* Get basename from argv[1] */
    if (argVec[0] != NULL)
        argVec[0]++;
    else
        argVec[0] = argv[1];
    argVec[1] = "hello world";
    argVec[2] = "goodbye";
    argVec[3] = NULL;  /* List
must be NULL-terminated */


 /* Execute the program
specified in argv[1] */

    execve(argv[1], argVec,
envVec);
    errExit("execve");
/* If we get here, something
went wrong */
}
```

Chapter 11 - SYSTEM LIMITS AND OPTIONS

- 11.1 System Limits
- 11.2 Retrieving System Limits (and Options) at Run Time
- 11.3 Retrieving File-Related Limits (and Options) at Run Time
- 11.4 Indeterminate Limits
- 11.5 System Options
- 11.6 Summary
- 11.7 Exercises

System Limits

- How many files can a process hold open at one time?
- Does the system support realtime signals?
- What is the largest value that can be stored in a variable of type int?
- How big an argument list can a program have?
- What is the maximum length of a pathname?

Hey, you can hard-code the assumption into your program

Why worry?

- **But, let's not do that. How about :**
  - Across UNIX implementations?
  - At run time on a particular implementation?
  - From one file system to another?

- **SUSv3 defines three functions— sysconf(), pathconf(), and fpathconf()**

- **getconf() to retrieve the values (from shell)**

SYSCONF(3) Linux Programmer's Manual SYSCONF(3)

NAME

sysconf - Get configuration information at runtime

SYNOPSIS

```
#include <unistd.h>
 long sysconf(int name);
```

DESCRIPTION

POSIX allows an application to test at compile or run time whether certain options are supported, or what the value is of certain configurable constants or limits.

At compile time this is done by including <unistd.h> and/or <limits.h> and testing the value of certain macros.

At run time, one can ask for numerical values using the present function sysconf(). On can ask for numerical values that may depend on the file system a file is in using the calls fpathconf(3) and pathconf(3). One can ask for string values using confstr(3).

Table 11-1. Selected SUSv3 limits

Name of Limit <limits.h>	Min Value	Sysconf() Name	Description
ARG_MAX	4096	_SC_ARG_MAX	Maximum bytes for arguments (argv) plus environment (environ) that can be supplied to an exec() (Environment List and Passing the Caller's Environment to the New Program)
-	-	_SC_CLK_TCK	Unit of measurement for times()
LOGIN_NAME_MAX	9	_SC_LOGIN_NAME_MAX	Maximum size of a login name (including terminating null byte)
OPEN_MAX	20	_SC_OPEN_MAX	Maximum number of file descriptors that a process can have open at one time, and one greater than maximum usable descriptor number (Process Resource Lim.)
NGROUPS_MAX	8	_SC_NGROUPS_MAX	Maximum number of supplementary groups of process can have open at one time, and one greater than maximum usable descriptor number (Process Res.Lim)

none	1	_SC_PAGESIZE	Size of a virtual memory page (_SC_PAGE_SIZE is a synonym)
RTSIG_MAX	8	_SC_RTSIG_MAX	Maximum number of distinct realtime signals (Realtime Signals)
SIGQUEUE_MAX	32	_SC_SIGQUEUE_MAX	Maximum number of queued realtime signals (Realtime Signals)
STREAM_MAX	8	_SC_STREAM_MAX	Maximum number of *stdio* streams that can be open at one time
NAME_MAX	14	_PC_NAME_MAX	Maximum number of bytes in a filename, *excluding* terminating null byte
PATH_MAX	256	_PC_PATH_MAX	Maximum number of bytes in a pathname, *including* terminating null byte
PIPE_BUF	512	_PC_PIPE_BUF	Maximum number of bytes that can be written atomically to a pipe or FIFO (Overview)

Limits categories

- **Runtime invariants**
  - May be different per-runtime
  - May be indeterminate
- **Pathname variable values**
  - May be different per-path
- **Runtime increasable values**
  - Fixed minimum across implementation
  - May be increased

Example 11-1. Using sysconf() TLPI/syslim/t_sysconf.c

```
#include "tlpi_hdr.h"
static void  /* Print 'msg' plus sysconf() value for 'name' */
sysconfPrint(const char *msg, int name)
{
    long lim;
    errno = 0;
    lim = sysconf(name);
    if (lim != -1) {    /* Call succeeded, limit determinate */
        printf("%s %ld\n", msg, lim);
    } else {
        if (errno == 0)  /* Call succeeded, limit indeterminate */
            printf("%s (indeterminate)\n", msg);
        else                    /* Call failed */
            errExit("sysconf %s", msg);
    }
}
```

```
int
main(int argc, char *argv[])
{
    sysconfPrint("_SC_ARG_MAX:         ", _SC_ARG_MAX);
    sysconfPrint("_SC_LOGIN_NAME_MAX:
",_SC_LOGIN_NAME_MAX);
    sysconfPrint("_SC_OPEN_MAX:        ", _SC_OPEN_MAX);
    sysconfPrint("_SC_NGROUPS_MAX:     ",
_SC_NGROUPS_MAX);
    sysconfPrint("_SC_PAGESIZE:        ", _SC_PAGESIZE);
    sysconfPrint("_SC_RTSIG_MAX:       ",
_SC_RTSIG_MAX);
    exit(EXIT_SUCCESS);
}
```

FPATHCONF(3) Linux Programmer's Manual FPATHCONF(3)

NAME

fpathconf, pathconf - get configuration values for files

SYNOPSIS

```
#include <unistd.h>

long fpathconf(int fd, int name);
long pathconf(char *path, int name);
```

DESCRIPTION

fpathconf() gets a value for the configuration option name for the open file descriptor fd.

pathconf() gets a value for configuration option name for the filename path.

Table 11-2. Details of selected pathconf() _PC_* names

Constant	Notes
_PC_NAME_MAX	For a directory, this yields a value for files in the directory. Behavior for other file types is unspecified.
_PC_PATH_MAX	For a directory, this yields the maximum length for a relative pathname from this directory. Behavior for other file types is unspecified.
_PC_PIPE_BUF	For a FIFO or a pipe, this yields a value that applies to the referenced file. For a directory, the value applies to a FIFO created in that directory. Behavior for other file types is unspecified.

TLPI/syslim/t_fpathconf.c

```
static void   /* Print 'msg' plus value of fpathconf(fd, name) */
fpathconfPrint(const char *msg, int fd, int name)
{
    long lim;
    errno = 0;
    lim = fpathconf(fd, name);
    if (lim != -1) {        /* Call succeeded, limit determinate */
        printf("%s %ld\n", msg, lim);
    } else {
        if (errno == 0)  /* Call succeeded, limit indeterminate */
            printf("%s (indeterminate)\n", msg);
        else                    /* Call failed */
            errExit("fpathconf %s", msg);
    }
}
```

TLPI/syslim/t_fpathconf.c - main

```
int
main(int argc, char *argv[])
{
    fpathconfPrint("_PC_NAME_MAX: ", STDIN_FILENO,
_PC_NAME_MAX);
    fpathconfPrint("_PC_PATH_MAX: ", STDIN_FILENO,
_PC_PATH_MAX);
    fpathconfPrint("_PC_PIPE_BUF: ", STDIN_FILENO,
_PC_PIPE_BUF);

    exit(EXIT_SUCCESS);
}
```

System Options

_POSIX_ASYNCHRONOUS_IO (_SC_ASYNCHRONOUS_IO)	*Asynchronous I/O*	
_POSIX_CHOWN_RESTRICTED (_PC_CHOWN_RESTRICTED)	Only privileged processes can use *chown()* and *fchown()* to change the user ID and group ID of a file to arbitrary values (Changing File Ownership: chown(), fchown(), and lchown())	*
_POSIX_JOB_CONTROL (_SC_JOB_CONTROL)	*Job Control* (Job Control)	+
_POSIX_MESSAGE_PASSING (_SC_MESSAGE_PASSING)	*POSIX Message Queues* (Chapter 52)	
_POSIX_PRIORITY_SCHEDULING (_SC_PRIORITY_SCHEDULING)	*Process Scheduling* (Realtime Process Scheduling API)	
_POSIX_REALTIME_SIGNALS (_SC_REALTIME_SIGNALS)	*Realtime Signals Extension* (Realtime Signals)	
_POSIX_SAVED_IDS (none)	Processes have saved set-user-IDs and saved set-group-IDs (Saved Set-User-ID and Saved Set-Group-ID)	+
_POSIX_SEMAPHORES (_SC_SEMAPHORES)	*POSIX Semaphores* (Chapter 53)	

Chapter 12. System and Process Info

- 12.1 The /proc File System
- 12.1.1 Obtaining Information About a Process: /proc/*PID*
- 12.1.2 System Information Under /proc
- 12.1.3 Accessing /proc Files
- 12.2 System Identification: *uname()*
- 12.3 Summary
- 12.4 Exercises

Process info: ps -

- root> ps -e

```
PID  TTY          TIME CMD
  1  ?        00:00:01 init
  2  ?        00:00:00 kthreadd
  3  ?        00:00:00 migration/0
  4  ?        00:00:00 ksoftirqd/0
  5  ?        00:00:00 watchdog/0
  6  ?        00:00:00 migration/1
  7  ?        00:00:00 ksoftirqd/1
  8  ?        00:00:00 watchdog/1
  9  ?        00:00:00 events/0
 10  ?        00:00:00 events/1
 11  ?        00:00:00 cpuset
 12  ?        00:00:00 khelper
 13  ?        00:00:00 netns
 14  ?        00:00:00 async/mgr
 15  ?        00:00:00 pm
 17  ?        00:00:00 sync_supers
1786 pts/0    00:00:00 bash
1807 ?        00:00:00 oosplash
1820 ?        00:00:00 notification-da
1895 ?        00:00:44 soffice.bin
1978 ?        00:00:00 flush-8:0
2005 ?        00:00:00 flush-0:22
2010 ?        00:00:02 plugin-containe
2013 ?        00:00:05 acroread
2087 pts/0    00:00:00 vi
2092 ?        00:00:00 sshd
2181 ?        00:00:00 sshd
2182 pts/1    00:00:00 bash
2207 ?        00:00:00 evolution-data-
2211 ?        00:00:00 evolution-excha
2253 pts/1    00:00:00 ps
2254 pts/1    00:00:00 less
```

PS EXAMPLES

To see every process on the system using standard syntax:

```
ps -e
ps -ef
ps -eF
ps -ely
```

To see every process using BSD syntax:

```
ps ax
ps axu
```

To print a process tree:

```
ps -ejH
ps axjf
```

To get info about threads:

```
ps -eLf
ps axms
```

To get security info:

```
ps –eo euser, ruser, suser, fuser, f, comm, label
ps axZ
ps –eM
```

To see every process running as root (real & effective ID) in user format:

```
ps -U root -u root u
```

Print only the process IDs of syslogd:

```
ps -C syslogd -o pid=
```

/proc file system revisited

- How many processes are running on the system and who owns them?
- What files does a process have open?
- What files are currently locked, and which processes hold the locks?
- What sockets are being used on the system?

/proc/PID/status

- **root> cat /proc/2/status**

```
Name:   kthreadd
State:  S (sleeping)
Tgid:   2
Pid:    2
PPid:   0
TracerPid: 0
Uid:    0   0 0   0
Gid:    0   0 0   0
FDSize: 32
Groups:
Threads:    1
SigQ:   0/16023
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: ffffffffffffffff
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: ffffffffffffffff
CapEff: fffffffffffffeff
CapBnd: ffffffffffffffff
Cpus_allowed:   ff
Cpus_allowed_list:      0-7
Mems_allowed:   1
Mems_allowed_list:      0
voluntary_ctxt_switches:
107
nonvoluntary_ctxt_switches:
0
```

Table 12-1. Selected files in each /proc/ PID directory

File	Description (process attribute)
cmdline	Command-line arguments delimited by \0
Cmd	Symbolic link to current working directory
environ	Environment list NAME = value pairs, delimited by \0
exe	Symbolic link to file being executed
fd	Directory containing symbolic links to files opened by this process
maps	Memory mappings
mem	Process virtual memory (must lseek() to valid offset before I/ O)
mounts	Mount points for this process
root	Symbolic link to root directory

More /proc/PID

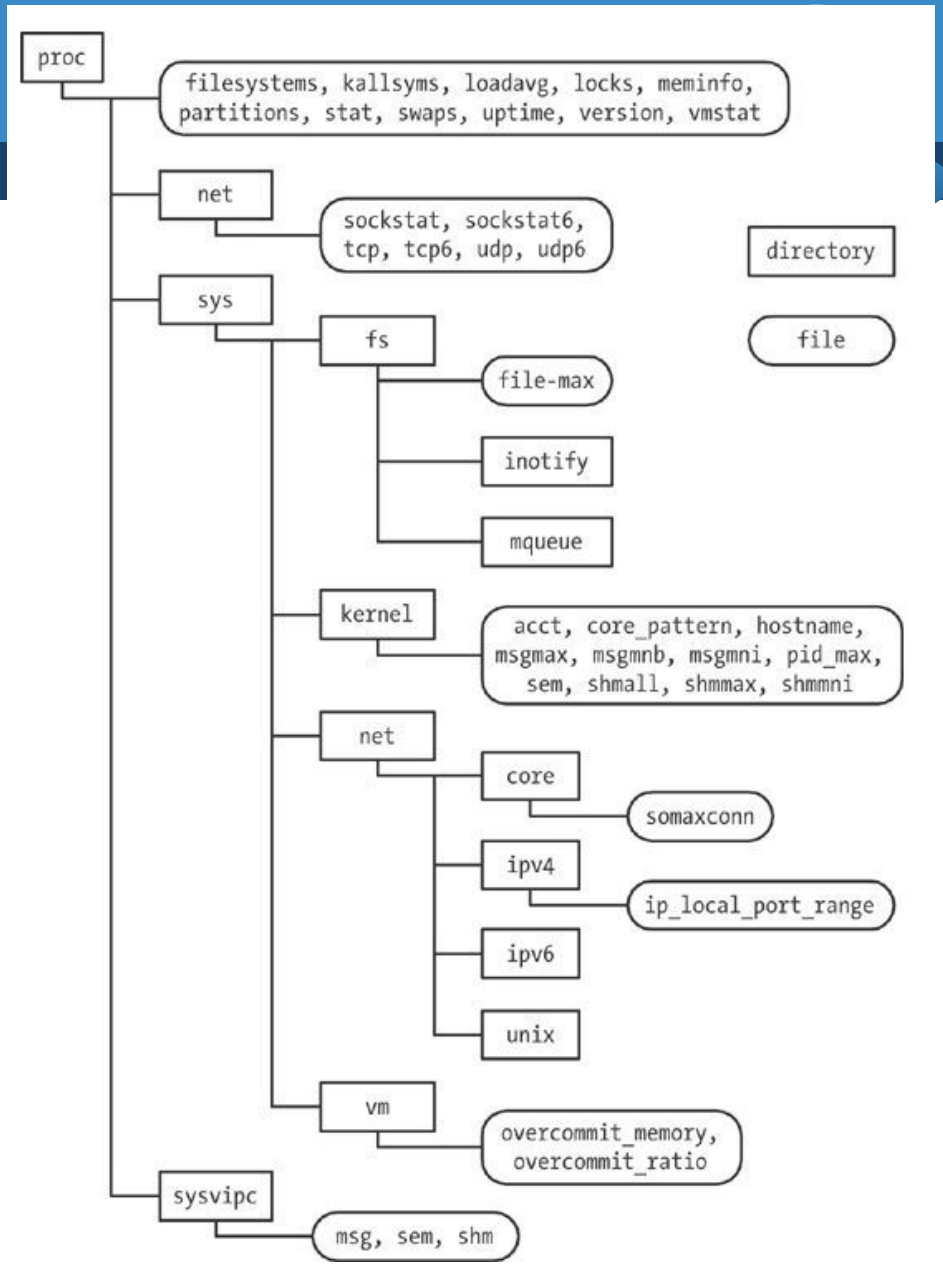
- /proc/PID/fd – symbolic link for every file descriptor that a process has open

- /proc/PID/task/TID – thread ID

Table 12-2 Selected /proc subdirectories

Directory	Information exposed by files in this directory
/proc	Various system information
/proc/net	Status information about networking and sockets
/proc/sys/fs	Settings related to file systems
/proc/sys/kernel	Various general kernel settings
/proc/sys/net	Networking and sockets settings
/proc/sys/vm	Memory-management settings
/proc/sysvipc	Information about System V IPC objects

Figure 12-1 /proc hierarchy



- root> cd sysinfo
- root> ls

```
Makefile
procfs_pidmax.c
procfs_user_exe.c
t_uname.c
procfs_pidmax
procfs_user_exe
t_uname
```

QA

Make -p shows Rules/Macro defs

```
Variables …
COMPILE.cpp = $(COMPILE.cc)
LINUX_LIBCRYPT = -lcrypt
# default
CC = cc
CPP = $(CC) -E
… hundreds of lines

%: %.cpp
#  commands to execute (built-in):
        $(LINK.cpp) $^ $(LOADLIBES) $(LDLIBS) -o $@

%.o: %.cpp
#  commands to execute (built-in):
        $(COMPILE.cpp) $(OUTPUT_OPTION) $<
```

Examples

- Date command
- TLPI/time/calendar_time.c

Strftime() - format date and time

Table 10-1. Selected conversion specifiers for strftime()

Specifier	Description	Example
%%	A % character	%
%a	Abbreviated weekday name	Tue
%A	Full weekday name	Tuesday
%b, %h	Abbreviated month name	Feb

none	1	_SC_PAGESIZE	Size of a virtual memory page (_SC_PAGE_SIZE is a synonym)
RTSIG_MAX	8	_SC_RTSIG_MAX	Maximum number of distinct realtime signals (Realtime Signals)
SIGQUEUE_MAX	32	_SC_SIGQUEUE_MAX	Maximum number of queued realtime signals (Realtime Signals)
STREAM_MAX	8	_SC_STREAM_MAX	Maximum number of *stdio* streams that can be open at one time

FILE I/O Buffering

- 13.1 Kernel Buffering of File I/O: The Buffer Cache
- 13.2 Buffering in the *stdio* Library
- 13.3 Controlling Kernel Buffering of File I/O
- 13.4 Summary of I/O Buffering
- 13.5 Giving the Kernel Hints About I/O Patterns: *posix_fadvise()*
- 13.6 Bypassing the Buffer Cache: Direct I/O
- 13.7 Mixing Library Functions and System Calls for File I/O
- 13.8 Summary
- 13.9 Exercises

Determining limits and options from the shell: getconf

- The constant FOPEN_MAX, defined in `<stdio.h>`, is synonymous with STREAM_MAX.
- NAME_MAX excludes the terminating null byte, while PATH_MAX includes it.