한양대학교 HANYANG UNIVERSITY

Unix System Overview

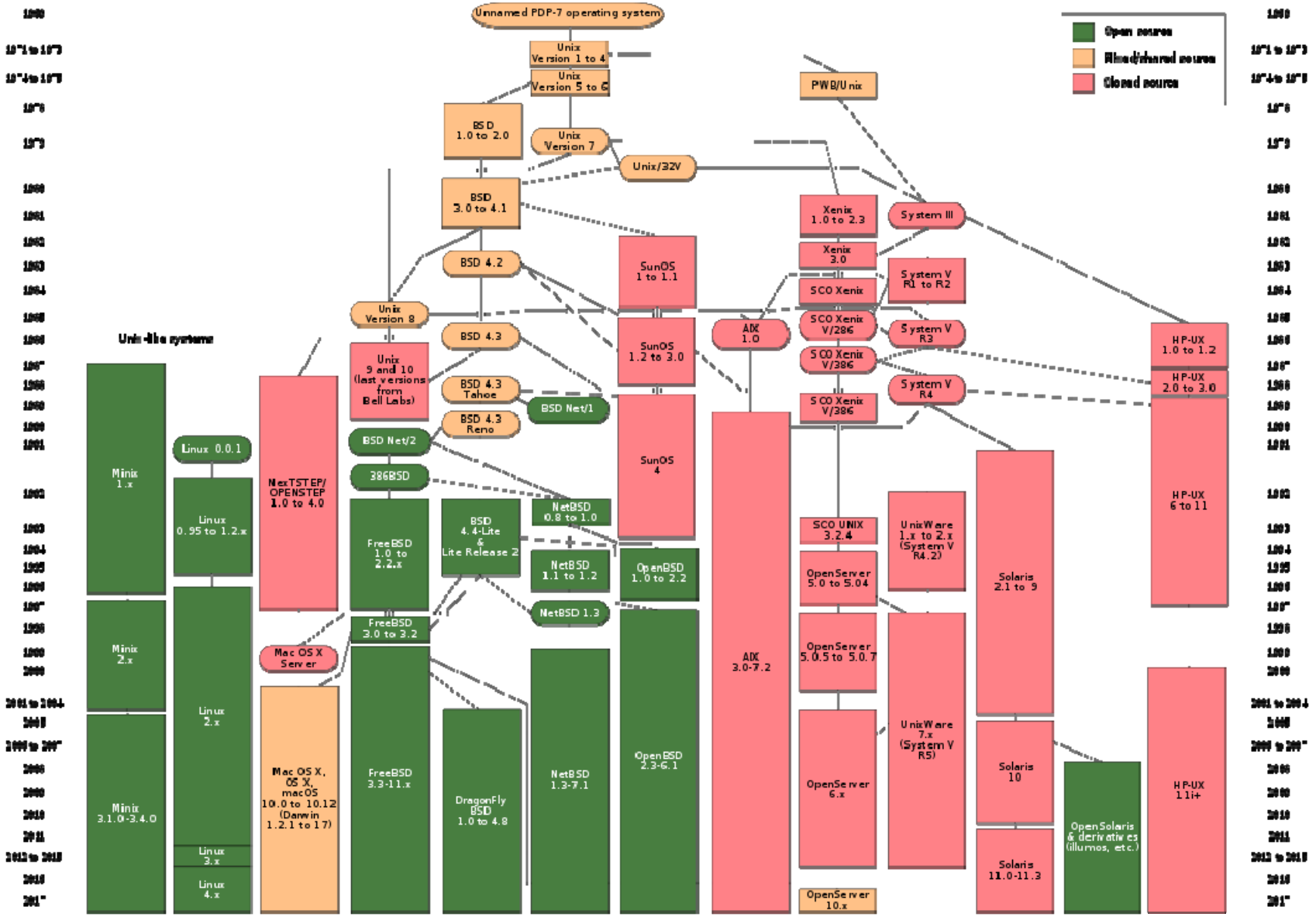
System Programming

2019 여름 계절학기

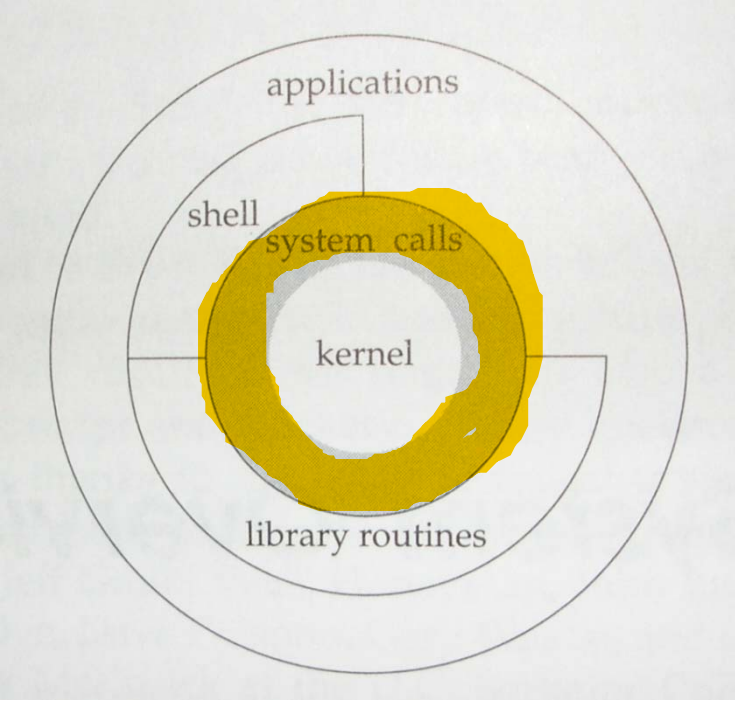
한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

- ❑ Unix System Overview
- ❑ Unix Standardization and Implementations
- ❑ File I/O
- ❑ Files and Directories
- ❑ Standard I/O Library
- ❑ System Data Files and Information
- ❑ Process Environment
- ❑ Process Control
- ❑ Signals
- ❑ Threads
- ❑ Advanced I/O
- ❑ Interprocess communication(IPC)
- ❑ Network IPC: Sockets

History of UNIX

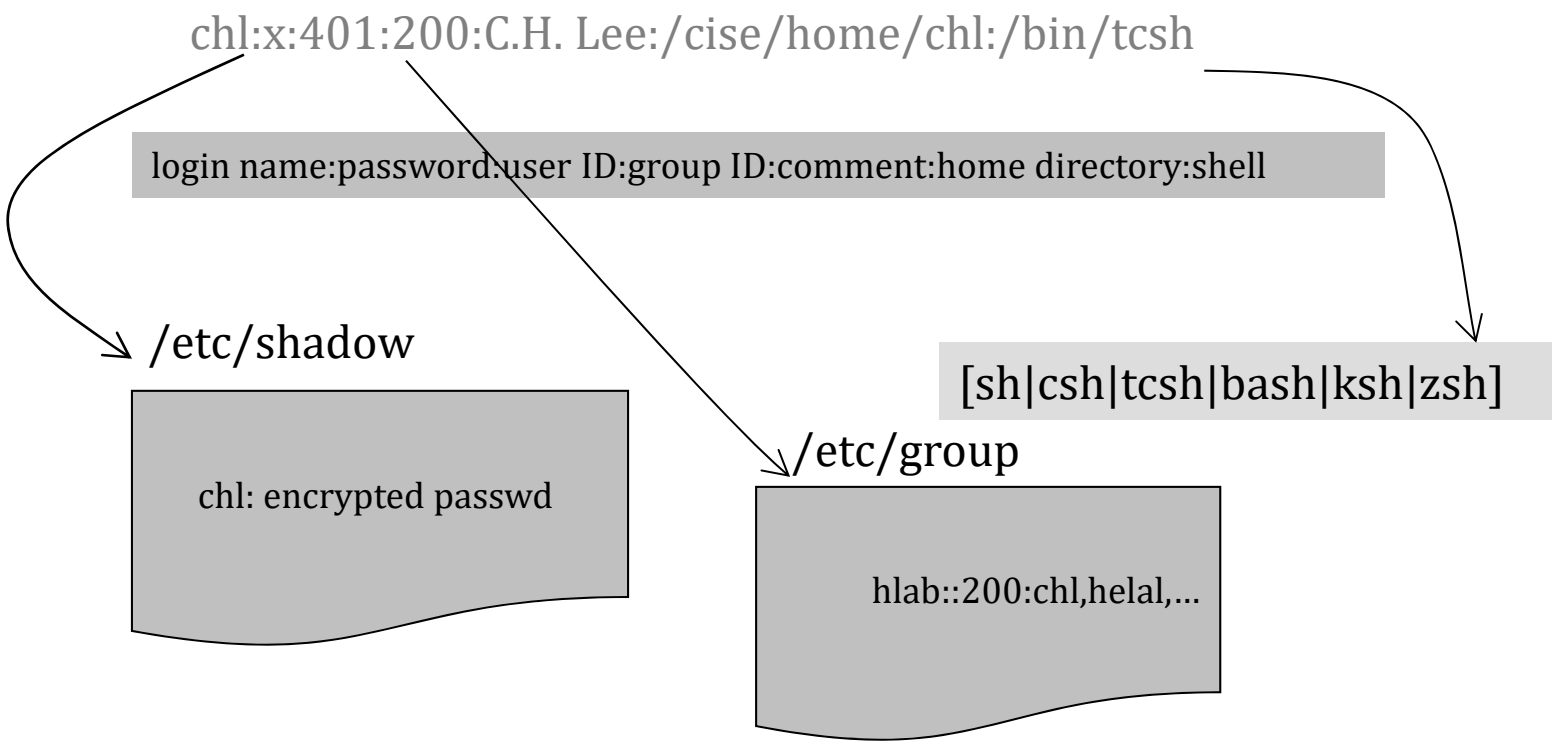


UNIX Architecture



- **Programming interface to the *kernel***
  - System call interfaces and functions in the standard C library

❑ `/etc/passwd` composed of seven colon-separated fields:



❑ Shell

- a command-line interpreter that reads user input and executes commands.
- Bourn Shell : UNIX version 7~
- Bourne-again shell : The GNU shell provided with all Linux systems

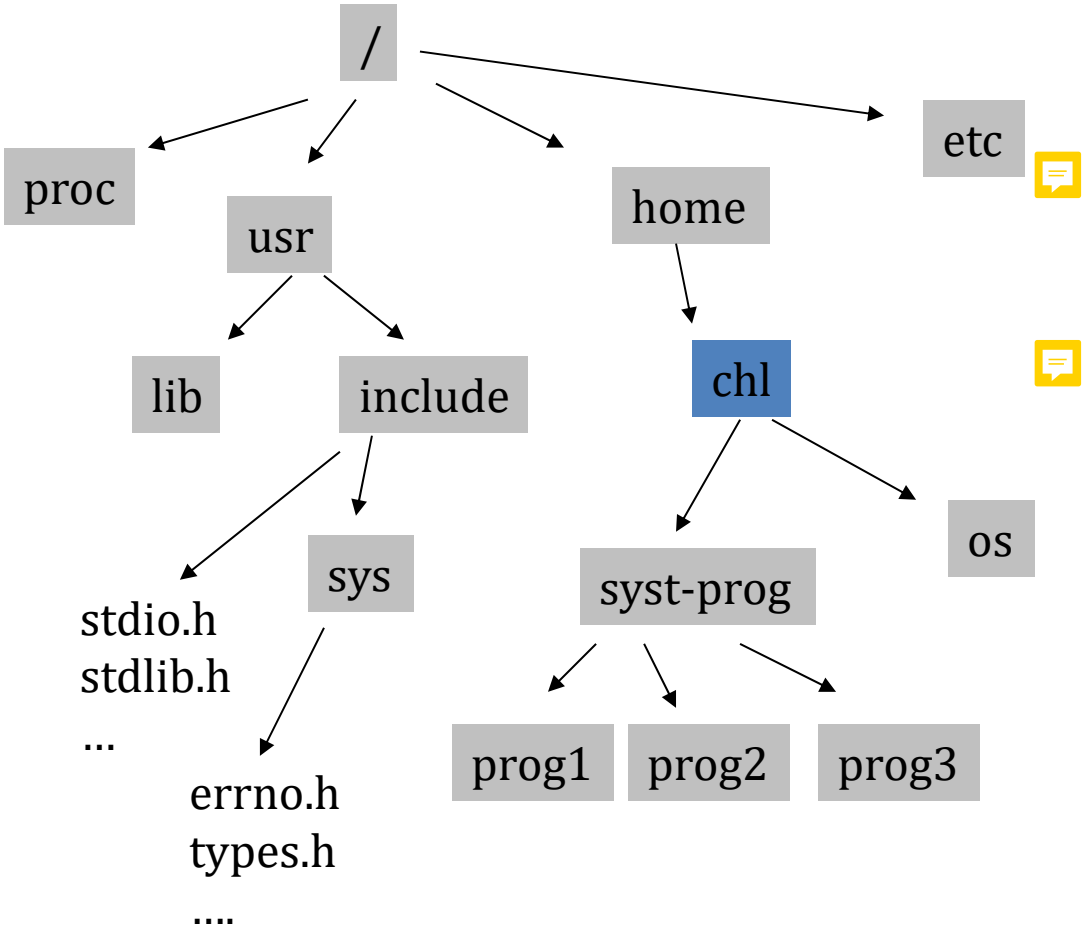
Name	Path	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
Bourne shell	`/bin/sh`	•	•	copy of `bash`	•
Bourne-again shell	`/bin/bash`	optional	•	•	•
C shell	`/bin/csh`	link to `tcsh`	optional	link to `tcsh`	•
Korn shell	`/bin/ksh`	optional	optional	•	•
TENEX C shell	`/bin/tcsh`	•	optional	•	•

Figure 1.2 Common shells used on UNIX systems

Files and Directories

- **UNIX file system: a hierarchical arrangement of directories and files**
  - File
  - A directory is a **file** that contains directory entries.
- **Filename**
  - At least 255 character filenames (Most commercial UNIX)
  - Special names: "/", ".", and ".."
- **Pathname**
  - A sequence of zero or more file names, separated by slash, and optionally starting with a slash
  - Absolute pathname vs. relative pathname
    - A pathname that begins with a slash is called an absolute pathname
    - otherwise, it's called a relative pathname. Relative pathnames refer to files relative to the current directory
- **Current Working Directory**
  - "."
  - pwd

UNIX File System



$ **pwd**
/home/chl

$ **ls syst-prog**
prog1 prog2 prog3

$ **ls ../../chl/syst-prog**
prog1 prog2 prog3

$ **ls /home/chl/syst-prog**
prog1 prog2 prog3

- **File descriptors: small non-negative integers to identify the files**
- **Standard input, output, and error**
  - stdin, stdout, and stderr
  - Redirections: “<“, “>”, and “2>” in the case of csh
- **Unbuffered I/O vs. Standard I/O**
  - (open, read, write, …) vs. (fopen, fgetc, printf, …)
  - Program 1.4 & Program 1.5

Program 1.3

```
#include "apue.h“
#include <dirent.h>

int main(int argc, char *argv[])
{
        DIR     *dp;
        struct  dirent  *dirp;

        if(argc != 2)
                err_quit("usage:  ls directory_name");

        if((dp = opendir(argv[1])) == NULL)
                err_sys("can't open %s", argv[1]);

        while((dirp = readdir(dp)) != NULL)
                printf("%s\n", dirp->d_name);

        closedir(dp);
        exit(0);
}
```

Program 1.4

```
#include "apue.h“
#define BUFFSIZE    4096
int main(void)
{
    int     n;
    char    buf[BUFFSIZE];
    while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
        if (write(STDOUT_FILENO, buf, n) != n)
            err_sys("write error");

    if (n < 0)
        err_sys("read error");

    exit(0);
}
```

Program 1.5

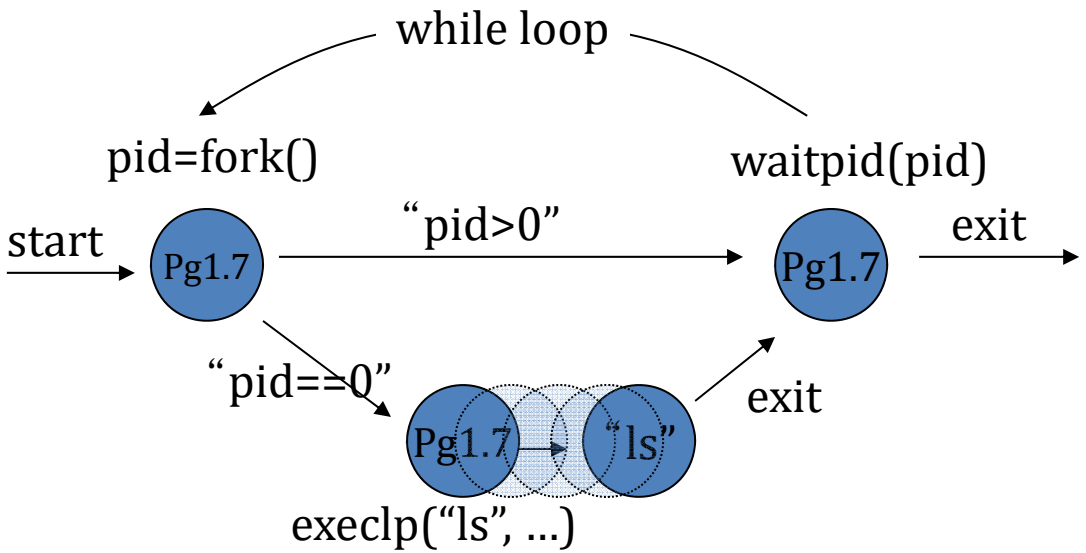
```
#include "apue.h“
int main(void)
{
   int     c;

   while ((c = getc(stdin)) != EOF)
      if (putc(c, stdout) == EOF)
         err_sys("output error");

   if (ferror(stdin))
      err_sys("input error");

   exit(0);
}
```

- Program: an executable file residing on disk
- Process: an executing instance of a program
- Process ID: a nonnegative integer
- Process control: *fork*, *exec*, and *waitpid* in Program 1.7



Program 1.7

```
#include "apue.h“
#include <sys/wait.h>
int main(void)
{
   char    buf[MAXLINE];   /* from apue.h */
   pid_t   pid;
   int     status;

   printf("%% ");
   while (fgets(buf, MAXLINE, stdin)
          != NULL) {
      if (buf[strlen(buf) - 1] == '\n')
         buf[strlen(buf) - 1] = 0;
```

```
if ((pid = fork()) < 0) {
         err_sys("fork error");
      } else if (pid == 0) {      /* child */
         execlp(buf, buf, (char *)0);
         err_ret("couldn't execute: %s", buf);
         exit(127);
      }

      /* parent */
      if ((pid = waitpid(pid, &status, 0)) < 0)
         err_sys("waitpid error");
      printf("%% ");
   }
   exit(0);
}
```

- **A thread of control**
- **All the thread within a process share**
  - Address space
  - File descriptors
  - Stacks
  - process-related attributes
- **Accesses to shared data should be synchronized**

Error Handling

- **When an error occurs in one of the UNIX System functions, a negative value (or a null pointer) is returned is often returned.**
- **In case of an error, the `errno` is set to a value to give additional info.**
- **/usr/include/(sys/)errno.h**

```
extern int errno;

#define EAGAIN  11 /* Resource temporarily unavailable */
#define ENOMEM   12 /* Not enough core       */
#define EACCES   13 /* Permission denied        */
```

  - Never cleared if no error occurs

- **/usr/include/string.h**

```
char *strerror(int errnum);
ex) strerror(EACCES)
```

- **/usr/include/stdio.h**

```
void perror(const char *msg);
ex) perror(argv[0])
```

Signals

- **A technique to notify a process that some condition has occurred.**
- **On receipt of a signal, a process can**
  - ignore the signal (not recommended.)
  - let the default action occur.
  - call your own handler (catch the signal.)
- **To generate a signal, [Ctrl-C], [Ctrl-backslash], or kill().**
- **Program 1.10**

Program 1.10

```
#include "apue.h"
#include <sys/wait.h>
static void sig_int(int); /* our signal-catching function */
int main(void)
{
  char buf[MAXLINE]; /* from apue.h */
  pid_t pid;
  int status;
  if (signal(SIGINT, sig_int) == SIG_ERR)
    err_sys("signal error");
  printf("%% ");/* print prompt (printf requires %% to print %) */
  while (fgets(buf, MAXLINE, stdin) != NULL) {
    if (buf[strlen(buf) - 1] == '\n')
    buf[strlen(buf) - 1] = 0; /* replace newline with null */
```

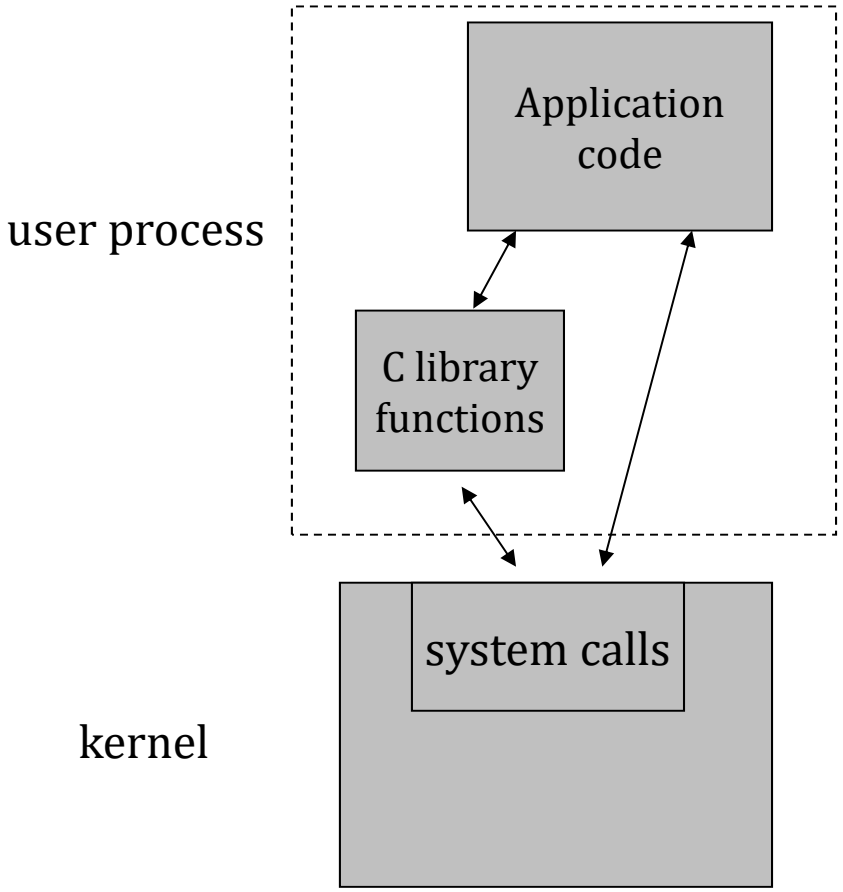
Program 1.10 (cont'd)

```
    if ((pid = fork()) < 0) {
      err_sys("fork error");
    } else if (pid == 0) {/* child */
      execlp(buf, buf, (char *)0);
      err_ret("couldn't execute: %s", buf);
      exit(127);
    }
    /* parent */
    if ((pid = waitpid(pid, &status, 0)) < 0)
      err_sys("waitpid error");
    printf("%% ");
  }
  exit(0);
}
void sig_int(int signo)
{
 printf("interrupt\n%% ");
}
```

UNIX Time Values

- Calendar time vs. Process time
- Calendar time in the number of seconds since 00:00:00 Jan. 1, 1970, UTC (`time_t`)
- Process time in clock ticks (`clock_t`)
- Clock time (*wall clock time*), user CPU time, and system CPU time
  - $ time ls

Sys Calls and Library Functions



- 50 system calls for Unix Version 7, 110 for 4.4BSD, 120 for SVR4, 240-260 for Linux, and 320 for FreeBSD.

- System call – manual section 2
- Library – manual section 3C, 3m, …

- More elaborate functionality
  - Atop of `sbrk()`, `malloc()` enables better memory allocation management.
  - Atop of `time()`, `gmtime()` provides broken-down time.
  - Atop of `read()`, `getc()` supports buffered I/O.

Unix man pages

❑ Sections

- 1 – commands, e.g. ls(1)
- 2 – system calls and error numbers, e.g. read(2)
- 3 – functions and libraries such as 3C, 3M, etc
- 4 – file formats, e.g. passwd(4)
- 5 – miscellany, e.g. environ(5)
- 6 – games and demos
- 7 – special files, e.g. hme(7D)
- 8
- 9 – device driver interface

Unix man pages

$ man man

$ man –s 1 time
$ man –s 2 time

$ man –s 1 intro
$ man –s 3 intro

Q and A