SYSTEM INFORMATION

UNIX Programming 2014 Fall by Euiseong Seo

Host Information

- POSIX defines host information as follows
  - OS name (Linux)
  - OS release (3.13.0)
  - OS version (#60-Ubuntu SMP Web Aug 13)
  - Node name (csl)
  - Machine name (x86-64)
- uname(2) retrieves host information

uname(2)

- Prototype

```
#include <sys/utsname.h>
int uname(struct utsname *buf);
struct utsname {
    char sysname[];    /* Operating system name (e.g., "Linux") */
    char nodename[];   /* Name within "some implementation-define network" */
    char release[];    /* Operating system release (e.g., "2.6.28") */
    char version[];    /* Operating system version */
    char machine[];    /* Hardware identifier */
}
```

- Example

```
struct utsname uts;
if(uname(&uts) == -1) {
    perror("uname");
    exit(1);
}
printf("hostname: %s\n", uts.nodename);
```

System Statistics

- sysinfo(2) provides following system statistics info

```
struct sysinfo {
    long uptime;             /* Seconds since boot */
    unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
    unsigned long totalram;  /* Total usable main memory size */
    unsigned long freeram;   /* Available memory size */
    unsigned long sharedram; /* Amount of shared memory */
    unsigned long bufferram; /* Memory used by buffers */
    unsigned long totalswap; /* Total swap space size */
    unsigned long freeswap;  /* swap space still available */
    unsigned short procs;    /* Number of current processes */
    unsigned long totalhigh; /* Total high memory size */
    unsigned long freehigh;  /* Available high memory size */
    unsigned int mem_unit;   /* Memory unit size in bytes */
}
```

System Statistics

- Prototype

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

int sysinfo(struct sysinfo *info);
```

- Example

```
struct sysinfo sinfo;
if(sysinfo(&sinfo) == -1) {
    perror("sysinfo");
    exit(1);
}
printf("The system is up for %ld seconds\n", sinfo.uptime);
```

System Configuration

- Some system limits can be determined at run-time
  - Maximum length of arguments to a new process
  - Maximum number of simultaneous processes per UID
  - Number of clock ticks per second
  - Page size
  - Number of processors configured
  - Number of processors currently online
  - And so on…

System Configuration

- Prototype

```
#include <unistd.h>

long sysconf(int name);
```

- Example

```
printf("The clock ticks %d times per second.\n",sysconf(_SC_CLK_TCK));
printf("I can only open %d files.\n",sysconf(_SC_OPEN_MAX));
```

Pathname Dependent System Conf.

- Limits related to files, directories, etc. can be obtained by fpathconf(3) or pathconf(3)
- Prototype

```
#include <unistd.h>

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

  - Name examples
    - _PC_LINK_MAX : maximum number of links to the file
    - _PC_NAME_MAX : maximum length of file names in a directory

Who Am I?

- UID and EUID can be retrieved by getuid(2) and geteuid(2), respectively
- Prototype

```
#include <unistd.h>
#include <sys/types.h>

uid_t getuid(void);
uid_t geteuid(void);
```

- Example

```
if(getuid() == 0) {
        printf("Respect me! I can delete all files!\n")
}
```

Who Is Her?

- To get the entire password entry from /etc/passwd, use getpwuid(3) or getpwnam(3)
- Prototype

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid)
struct passwd *getpwnam(const char *name)

struct passwd {
    char   *pw_name;       /* username */
    char   *pw_passwd;     /* user password */
    uid_t   pw_uid;        /* user ID */
    gid_t   pw_gid;        /* group ID */
    char   *pw_gecos;      /* user information */
    char   *pw_dir;        /* home directory */
    char   *pw_shell;      /* shell program */
};
```

Who Is Her?

- Example

```
struct passwd *pw;
pw = getpwuid(getuid());
printf("You logged in as %s.\n", pw->pw_name);
```

Machine Time

- Time is kept as a number of seconds since epoch
  - Epoch (or UNIX time, or POSIX time) is 0:00:00, January 1st, 1970 in UTC (coordinated universal time, or Greenwich Mean Time)
- time(3) gets current time

```
#include <time.h>
time_t time(time_t *t);

main()
{
    time_t t;
    time(&t); // or t = time(NULL);
    printf("Current time is %d\n", t);
}
```

Machine Time

- gettimeofday(2) gives better resolution, returning seconds and microseconds

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

int gettimeofday(struct timeval *tv, struct timezone *tz);
int settimeofday(const struct timeval *tv, const struct timezone *tz);

struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

Human Readable Time

- ctime(3) returns a human readable time string
- localtime(3) breaks down to month, day, and year
  - timelocal(3) does reverse of localtime(3)
- Prototype

```
#include <time.h>
char *ctime(const time_t *timep);
struct tm *localtime(const time_t *timep);
struct tm {
    int tm_sec;         /* seconds */
    int tm_min;         /* minutes */
    int tm_hour;        /* hours */
    int tm_mday;        /* day of the month */
    int tm_mon;         /* month */
    int tm_year;        /* year */
    int tm_wday;        /* day of the week */
    int tm_yday;        /* day in the year */
    int tm_isdst;       /* daylight saving time */
};
```

Human Readable Time

- Example

```c
#include <sys/types.h>
#include <time.h>

main()
{
    time_t t;
    struct tm *tm;
    time(&t);
    printf("%s", ctime(&t));
    tm = localtime(&t);
    printf("The current year is %d.\n", 1900+tm->tm_year);
}
```

```
MacPro:Temp euiseong$ ./a.out
Mon Oct 27 14:09:09 2014
The current year is 2014.
MacPro:Temp euiseong$
```

Programming Tip

- Prevention of memory leak
  - System programming involves frequent allocation and deallocation of heap memory
  - Memory objects that are not freed even when they are no longer used are called memory leak
- Memory overrun
  - Writing beyond given memory area
  - Sometimes generates segmentation fault, sometimes not
  - Difficult to detect during development
- There are many tools to detect memory bugs

Programming Tip

- *Electric Fence*
  - Simple memory debugger written by Bruce Perens
  - Usually used to detect following bug types
    - Overrunning the end (or beginning) of a dynamically allocated buffer
    - Using a dynamically allocated buffer after returning it to the heap
- You can use *dmalloc* or *valgrind* to detect more serious memory leaks and bugs

Programming Tip

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    char *arr;    int i;
    arr = (char *)malloc(sizeof(char)*5);
    strcpy(arr,"amee is my name");
    return 0;
}
```

```
euiseong@accept:~/Temp$ ulimit -c unlimited
euiseong@accept:~/Temp$ gcc test2.c -lefence
euiseong@accept:~/Temp$ ./a.out

  Electric Fence 2.2 Copyright (C) 1987-1999 Bruce Perens
<bruce@perens.com>
Segmentation fault (core dumped)
euiseong@accept:~/Temp$ gdb ./a.out ./core
```

Programming Tip

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    char *arr;    int i;
    arr = (char *)malloc(sizeof(char)*5);
    return 0;
}
```

```
euiseong@accept:~/Temp$ dmalloc -l ./logfile -i 100 high
DMALLOC_OPTIONS=debug=0x4f4ed03,inter=100,log=./logfile
export DMALLOC_OPTIONS
euiseong@accept:~/Temp$ DMALLOC_OPTIONS=debug=0x4f4ed03,inter=100,log=./logfile
euiseong@accept:~/Temp$ export DMALLOC_OPTIONS
euiseong@accept:~/Temp$ gcc test2.c -ldmalloc
euiseong@accept:~/Temp$ ./a.out
euiseong@accept:~/Temp$ more logfile
```