Module 3

System and Process Information

Overview

- Objectives
- Relevance

Retrieving Host Information

- `uname()`— Retrieves host name and other related information
- `sysinfo()`— Reports and sets information about the operating system

`myuname.c`

```
1   #include <sys/utsname.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   main() {
6
7     struct utsname uts;
8
9     if( uname(&uts) == -1 ) {
10      perror("myuname.c:main:uname");
11      exit(1);
12    }
13
14    printf("operating system: %s\n", uts.sysname);
15    printf("hostname: %s\n", uts.nodename);
16    printf("release: %s\n", uts.release);
17    printf("version: %s\n", uts.version);
18    printf("machine: %s\n", uts.machine);
19  }
```

mysysinfo.c

```
1   #include <sys/systeminfo.h>
2   #include <stdio.h>
3
4   #define BUFSIZE 1024
5
6   main() {
7     char buf[BUFSIZE];
8     int num;
9
10    num = sysinfo( SI_HW_SERIAL, buf,
          BUFSIZE);
11    if (num == -1) {
12      perror("sysinfo");
13    exit(1);
14    }
15
16    printf("hostid: %s\n", buf);
17    printf("hostid: %x\n", atoi(buf));
18  }
```


Retrieving System Variables

```
1   #include <sys/unistd.h>
2   #include <stdio.h>
3
4   main() {
5
6     printf("Number of processors: %d\n",
7        sysconf(_SC_NPROCESSORS_CONF));
8     printf("Memory page size: %d\n",
9        sysconf(_SC_PAGESIZE));
10    printf("Clock ticks/second: %d\n",
11       sysconf(_SC_CLK_TCK));
12    printf("Number of files that can be
13       opened: %d\n", sysconf(_SC_OPEN_MAX));
14  }
```


Determining File and Directory Limits

```
1   #include <unistd.h>
2   #include <stdio.h>
3
4   main() {
5
6     printf("Maximum filename length: %d\n",
7        pathconf(".", _PC_NAME_MAX));
8     printf("Maximum path length: %d\n",
9        pathconf("/", _PC_PATH_MAX));
10    printf("Pipe buffer size: %d\n",
11       pathconf("/var/spool/cron/FIFO",
12       _PC_PIPE_BUF));
13  }
```


Retrieving User Information

- `getuid()` – Retrieves user identification numbers
- `getpwuid()` – Retrieves the user identification number from the password database
- `getpwnam()` – Retrieves the user name from the password database

Retrieving User Information

```
1   #include <sys/types.h>
2   #include <unistd.h>
3   #include <pwd.h>
4   #include <stdio.h>
5
6   main() {
7
8     struct passwd *pw;
9
10    pw = getpwuid( getuid() );
11    printf("Logged in as %s\n", pw->pw_name);
12  }
```


Retrieving Machine Time

- `time()` – Returns number of seconds since 0:00:00, January 1, 1970
- `gettimeofday()` – Returns time in seconds and microseconds
- `ctime()` – Returns time in a human readable format
- `gmtime()` – Breaks time from `time()` into fields from seconds to years, Greenwich mean time
- `localtime()` – Same as `gmtime()` except local time
- `strftime()` – Returns time in customized string format

mytime.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4
5   main() {
6
7     time_t t;
8
9     time(&t);
10    printf("Time in seconds = %d\n", t);
11  }
```


mygettimeofday.c

```
1   #include <sys/time.h>
2   #include <stdio.h>
3
4   main() {
5
6     struct timeval tv;
7
8     gettimeofday(&tv, NULL);
9     printf("Time: seconds = %d microseconds = %d\n",
10            tv.tv_sec, tv.tv_usec);
11  }
```

Retrieving Machine Time

myctime.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4
5   main() {
6
7     time_t t;
8
9     time(&t);
10    printf("%s\n", ctime(&t))
11  }
```


Converting Time

mylocaltime.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4
5   main() {
6     time_t t;
7     struct tm *tmptr;
8     time(&t);
9     tmptr = localtime(&t);
10    printf("The year is %d\n", 1900 + tmptr->tm_year);
11    printf("The day of the year is %d\n",
12      tmptr->tm_yday);
13  }
14  /* Yesterday: 24hrs*60min/hr*60sec/min */
15    t2 = t1 - (24*60*60);
16    printf("Yesterday: %s", ctime(&t2));
17
```


Manipulating Time Data

mymaniptime.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4
5   main() {
6
7     time_t t1;
8     time_t t2;
9     struct tm *tmp1;
10    struct tm *tmp2;
11    char timestr[40];
12
13    time(&t1);
14    printf("Today : %s", ctime(&t1));
15
```



```
16    /* Yesterday: 24 hrs*60min/hr*60sec/min */
17    t2 = t1 - (24*60*60);
18    printf("Yesterday: %s, ctime(&t2));
19
20    /* A week ago */
21    t2 = t1 - 7*(24*60*60);
22    printf("Last Week: %s", ctime(&t2));
23
24    /* Two weeks from now */
25    t2 = t1 + 2*7*(24*60*60);
26    printf("Two Weeks: %s", ctime(&t2));
27
28    tmp2 = localtime(&t2);
29    strftime(timestr, 40,
30      "%D, Julian Date: %j ", tmp2);
31    puts( timestr );
32  }
```

Retrieving Machine Time

mytime.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4
5   main() {
6
7     time_t t;
8
9     time(&t);
10    printf("Time in seconds = %d\n", t);
11  }
```


mygettimeofday.c

```
1   #include <sys/time.h>
2   #include <stdio.h>
3
4   main() {
5
6     struct timeval tv;
7
8     gettimeofday(&tv, NULL);
9     printf("Time: seconds = %d microseconds = %d\n",
10        tv.tv_sec, tv.tv_usec);
11  }
```


Using Environment Variables

- `getenv()`— Retrieves the value of an environment variable
- `putenv()`— Adds variables to the environment
- `unsetenv()`— Removes an environment variable

Using Environment Variables

mygetenv.c

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   main() {
5
6     char *value;
7
8     value = getenv("HOME");
9     if( value == NULL ) {
10      printf("HOME is not defined\n");
11    } else if( *value == '\0') {
12      printf("HOME defined but has no value\n");
13    } else {
14      printf("HOME = %s\n", value);
15    }
16  }
```

myputenv.c

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   main() {
5
6     char *value;
7
8     putenv("HOME=/tmp");
9
10    value = getenv("HOME");
11    if( value == NULL ) {
12      printf("HOME is not defined\n");
13    } else if( *value == '\0') {
14      printf("HOME defined but has no value\n");
15    } else {
16      printf("HOME = %s\n", value);
17    }
18  }
```


myunsetenv.c

```
1   #include <stdlib.h>
2   #include <stdio.h>
3   #include <string.h>
4
5   /* Function to remove an environment variable */
6   void unsetenv( char *var) {
7
8     extern char **environ;
9     char **env;
10    int  len;
11
12    env = environ;
13    len = strlen(var);
14    while (*env) {
15      if((strncmp(var, *env, len) == 0) &&
16          ((*env)[len] == '=')) {
17        break; /* match */
18      }
19      env++;
20    }
```



```
21
22    /**********************
23    if (*env) is non-NULL, then we had a match, and will
24    enter the loop. If it is NULL, no match, and we will
25    not enter the loop. The loop now moves all entries
26    up one, to delete the unset variable.
27    **********************/
28    while (*env) {
29      *env = *(env + 1);
30      env++;
31    }
32    return;
33  }
34
35  /* Function to display an environment variable */
36  int checkvar(char *var) {
37
38    char *getenv();
39    char *value;
40
41    value = getenv(var);
42    if (value == NULL)
```



```
43      printf("%s not defined\n", var);
44    else
45      printf("%s = %s\n", var, value);
46  }
47
48  int main() {
49
50    unsetenv("MYVAR0");
51    putenv("MYVAR=somevalue");
52    checkvar("MYVAR");
53    putenv("MYVAR2=anothervalue");
54    checkvar("MYVAR2");
55    unsetenv("MYVAR");
56    checkvar("MYVAR");
57    checkvar("MYVAR2");
58  }
```

Using Process IDs and Process Groups IDs

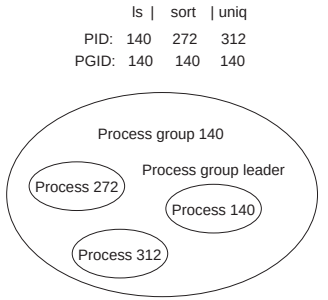
Function Calls

- `getpid()`– Retrieves process ID number
- `getppid()`– Retrieves parent PID
- `getpgrp()`– Retrieves process group ID number
- `getpgid()`– Retrieves process group GID

Identification Numbers

- Process ID
- Process group ID
- User group ID and group ID
- Effective user ID and effective group ID

Using Process IDs and Process Groups IDs



Using Real and Effective Group IDs

- `getuid()`– Retrieves real user ID
- `getgid()`– Retrieves real group user ID
- `geteuid()`– Retrieves effective user ID
- `geteguid()`– Retrieves effective group user ID

Resource Limits

- `limit()`– Displays and sets resources limits
- `getrlimit()`– Displays resource limits
- `setrlimit()`– Sets resources limits

Resource Limits

Resource Macro	Meaning	Signal	errno
RLIMIT_CORE	The maximum size of a core file in bytes that a process can create.		
RLIMIT_CPU	The maximum amount of CPU time in seconds used by a process. Soft only.	SIGXCPU	
RLIMIT_DATA	The maximum size of a process's heap in bytes.		ENOMEM
RLIMIT_FSIZE	The maximum size of a file in bytes that a process may create.	SIGXFSZ	EFBIG
RLIMIT_NOFILE	The maximum number of file descriptors that a process may create.		EMFILE
RLIMIT_STACK	The maximum size of a process's stack in bytes.	SIGSEGV	
RLIMIT_VMEM	The maximum size of a process's mapped address space in bytes.		ENOMEM

Resource Limits

```
1   #include <sys/resource.h>
2   #include <unistd.h>
3   #include <stdio.h>
4   main() {
5
6     struct rlimit myrlim;
7
8     getrlimit(RLIMIT_NOFILE, &myrlim);
9     printf("I can only open %d files\n",
          myrlim.rlim_cur);
10
11    myrlim.rlim_cur = 512;
12
13    if (setrlimit(RLIMIT_NOFILE, &myrlim) == -1) {
14       perror("setrlimit");
15    }
16
17    getrlimit(RLIMIT_NOFILE, &myrlim);
```



```
18    printf("I can now open %d files\n",
          myrlim.rlim_cur);
19    printf("sysconf() says %d files.\n",
20        sysconf(_SC_OPEN_MAX));
21  }
```


Time Usage

```
1   #include <sys/types.h>
2   #include <sys/times.h>
3   #include <unistd.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   main() {
8
9     int m;
10    time_t t;
11    struct tms tms1, tms2;
12    clock_t time1, time2;
13
14    /* Num ticks per sec */
15    double tick = sysconf(_SC_CLK_TCK);
16
17    if( (time1 = times( &tms1 )) == -1 ) {
18      perror("times");
19      exit(1);
```

```
20  }
21
22    for( m = 0; m < 999999; m++ ) {
23      getpid();
24    }
25
26    if( (time2 = times( &tms2 )) == -1 ) {
27      perror("times");
28      exit(1);
29    }
30
31    printf("My Real time is: %f sec \n",
32      ( time2 - time1 ) / tick );
33    printf("My User time is: %f sec \n",
34      (( tms2.tms_utime - tms1.tms_utime ) / tick ));
35    printf("My Sys  time is: %f sec \n",
36      (( tms2.tms_stime - tms1.tms_stime ) / tick ));
37  }
```


Current Directory

- `getcwd()`— Retrieves current working directory
- `chdir()`— Changes current directory

Current Directory

```
1   #include <sys/param.h>
2   #include <unistd.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   main() {
6
7     char  *dir;
8     long  pathmaxlen = pathconf(".", _PC_PATH_MAX);
9
10    dir = getcwd((char *)NULL, pathmaxlen + 1);
11
12    if(dir == NULL) {
13      perror("getcwd");
14      exit(1);
15    }
16
17    printf("CWD: %s\n", dir);
18    free(dir);
19
```



```
20    if(chdir("/tmp") == -1) {
21      perror("chdir");
22    }
23
24  pathmaxlen = pathconf(".", _PC_PATH_MAX);
25  dir = getcwd((char *)NULL, pathmaxlen + 1);
26
27    if (dir == NULL) {
28      perror("getcwd");
29    }
30
31    printf("CWD: %s\n", dir);
32
33    free(dir);
34  }
```

Exercise: System and Process Information

- Objectives
- Tasks
- Discussion
- Solutions
- List various attributes of a process
- Describe the differences between real and effective user IDs
- Retrieve time resource usage information about a process and its children