UNIX® System Interface Programming

SI-220

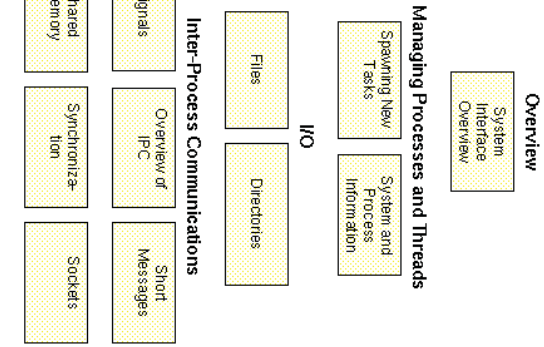
About This Course

Course Goal

This course provides you with knowledge and skills to:

- Become familiar with the UNIX® Application Programming Interface
- Learn to build multi-tasking solutions to solve problems
- Learn how processes and threads interact with one another

Course Map



Topics Not Covered

- Network Programming – Covered in SI-240: *Networking Programming*
- Multi-threaded Programming – Covered in SI-260: *Multi-Threaded Applications Programming*

How Prepared Are You?

- Write correct C programs that use command-line arguments, pointers, and structures
- Create and edit text files using `vi` or the Common Desktop Environment text editor
- Use basic Solaris Operating Environment commands

Introductions

- Name
- Company affiliation
- Title, function, and job responsibility
- Distributed computing experience
- Component development experience
- Application builder tool experience
- Reasons for enrolling in this course
- Expectations for this course

How to Use the Icons


Additional resources


Demonstration


Discussion


Note


Caution

Typographical Conventions and Symbols

- `Courier` is used for the names of commands, files, directories, programming code, programming constructs, and on-screen computer output.
- **`Courier bold`** is used for characters and numbers that you type, and for each line of programming code that is referenced in a textual description.
- *`Courier italics`* is used for variables and command-line placeholders that are replaced with a real name or value.
- ***`Courier italics bold`*** is used to represent variables whose values are to be entered by the student as part of an activity.

Typographical Conventions and Symbols

- *Palatino italics* is used for book titles, new words or terms, or words that are emphasized.

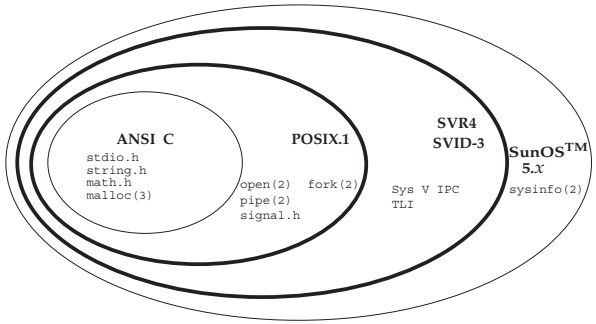
Module 1

System Interface Overview

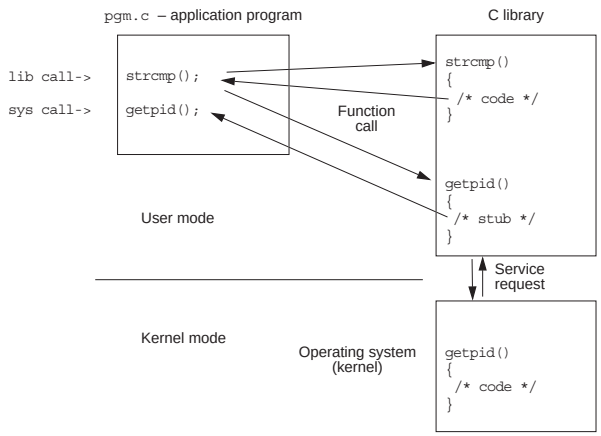
Overview

- Objectives
- Relevance

Supported Programming Standards



System Calls and Library Calls



Making a System Call

mytime.c

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


Making a System Call

badtime.c

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


Making a Library Call

mylibcall.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4   #include <string.h>
5   #include <unistd.h>
6   main() {
7
8     time_t t;
9     char then[30];
10    char *now;
11
12    time(&t);
13
14    /* ctime() put the current time into static space */
15    now = ctime(&t);
16
17  /* Copy the data into a new space */
```



```
18    strcpy(then, now);
19
20  /* Let time pass for one minute */
21    sleep(60);
22
23    time(&t);
24
25    /* ctime() puts new time into old static space */
26    now = ctime(&t);
27
28    printf("%s%s", then, now);
29  }
```

Making a Library Call

badlibcall.c

```
1   #include <sys/types.h>
2   #include <time.h>
3   #include <stdio.h>
4   main() {
5
6     time_t t;
7     char *then;
8     char *now;
9
10    time(&t);
11
12    /* ctime() put the current time into static space */
13    then = ctime(&t);
14
15  /* Let time pass for one minute */
16    sleep(60);
17
```



```
18    time(&t);
19
20    /* ctime() puts new time into same space */
21    now = ctime(&t);
22
23    /* then and now point to the same space */
24    printf("%s%s", then, now);
25  }
```


Error Handling

Example

```
1   #include <sys/types.h>
2   #include <unistd.h>
3   #include <stdio.h>
4   void perror();
5
6   main() {
7
8     if (setuid(23) == -1) {
9       perror("Setuid failure");
10    }
11  }
```


Example

```
1   #include <errno.h>
2   #include <string.h>
3   #include <stdio.h>
4   #include <fcntl.h>
5
6   main() {
7
8     char filename[NAMESIZE];
9     int fd;
10
11    gets(filename)
12
13    if ((fd = open(filename, O_RDWR)) == -1) {
14      fprintf (stderr, "Error opening %s: %s\n",
15          filename, strerror(errno));
16      exit (errno);
17  #define NAMESIZZE 20
18  }
```

System Calls Compared With Library Calls

System Call	Library Call
Described in Section 2 of the man pages.	Described in sections 3 of the man pages.
Never allocates space for parameters.	Can allocate space for parameters (see man pages). If allocates space, it can be static or dynamic.
Executes in system mode (kernel mode).	Executes in user mode.
When a failure occurs:	When a failure occurs:
Returns -1.	Often returns NULL (see man pages).
Sets `errno` (so you can use `perror(3C)`).	Can set `errno` (see man pages).

Dynamic Memory Allocation

- `malloc()`— Allocates memory
- `free()`— Frees memory
- `realloc()`— Changes the memory allocated

Dynamic Memory Allocation

```
1   #include <time.h>
2   #include <sys/time.h>
3   #include <string.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   main() {
8
9   struct timeval *tvp;
10    char *date;
11
12    tvp = (struct timeval *)
13        malloc(sizeof(struct timeval));
14    if (tvp == NULL) {
15      fprintf(stderr, "Out of virtual memory.\n");
16      exit(1);
17    }
18
19    if (gettimeofday(tvp, NULL) == -1) {
```



```
20      perror("gettimeofday failed");
21      exit(1);
22    }
23
24      date = strdup(ctime(&tvp->tv_sec));
25
26    if (date == NULL) {
27      fprintf(stderr, "Out of virtual memory.\n");
28      exit(1);
29    }
30
31    printf("%s", date);
32
33    free(tvp);
34    free(date);
35
36    return 0;
37  }
```

Automatic Storage Class

```
1   #include <stdio.h>
2
3   main() {
4     int *foo(int);
5       int *yp;
6
7       yp = foo(3);
8     another_function_call();
9     printf(" %d \n", *yp );
10  }
11
12  int *
13  foo( int p )
14  {
15    int a;
16
17      a = p + 2;
18    return( &a );
19  }
20
21  another_function_call() {
22      ;
23  }
```


String Functions

mystring.c

```
1   #include <string.h>
2
3   main() {
4
5     char str[80];
6     char *cp;
7
8     /* Copies 3 bytes into str; 'a', 'b', and '\0' */
9     strcpy(str, "ab");
10
11    /* Appends "de" to str */
12    strcat(str, "de");
13
14    /* Returns the length of string "ab", which is 2 */
15    strlen("ab");
16
17    /* Allocates 3 bytes of space, and copies "ab"
```



```
18    into it */
19    cp = strdup("ab");
20
21    /* Free the address held by cp */
22    free(cp);
23
24    /* Return the pointer to the character 'e' in the string
25  "hello" */
26    cp = strchr("hello", 'e');
27  }
```


myparse.c

```
1   #include <string.h>
2
3   main() {
4
5     /* Returns a pointer to the first occurrence of any
6        character in the string "def" that is found in the
7        string "hello"; in this case the character 'e' */
8     strpbrk("hello", "def");
9
10    /* Returns a pointer to the first occurrence of "cd" found
11       in string "abcdef". Returns NULL if none is found */
12    strstr("abcdef", "cd");
13
14    /* Returns the length of the initial segment of a given
15       string. In this case, 3 is returned for the length
16       of "123", common in both strings. */
17    strspn("123def4", "0123456789");
18
19  }
```

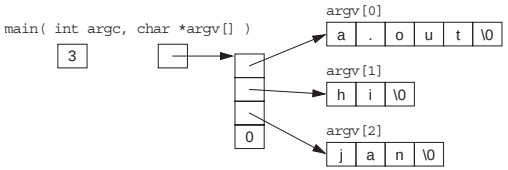
mystrtok.c

```
1   #include <string.h>
2
3   main() {
4
5     char *cp;
6     char buf[]="Hello World";
7
8     /* Make cp point to Hello\0 */
9     cp = strtok(buf, " ");
10
11    /* Now make cp point to World\0 */
12    cp = strtok(NULL, " ");
13
14    /* Now make cp point NULL */
15    cp = strtok(NULL, " ");
16  }
```


mystrtok2.c

```
1   #include <string.h>
2   #include <stdio.h>
3
4   #define DELIMITERS " \t\n"
5
6   main() {
7
8     char *arg;
9     char line[] = "ls -a /";
10
11    arg = strtok(line, DELIMITERS);
12    while( arg ) {
13      printf("%s\n", arg);
14      arg = strtok( (char*)NULL, DELIMITERS);
15    }
16  }
```


Using Command Line Arguments



myarray.c

```
1   #include <stdio.h>
2   int main( int argc, char *argv[]) {
3
4     int m;
5
6     for( m = 0; m < argc; m++ ) {
7       printf("Argument %d = %s\n", m, argv[m]);
8     }
9     return 0;
10  }
```

mypointer.c

```
1   #include <stdio.h>
2
3   int main( int argc, char **argv) {
4
5     char **tmp;
6
7     for( tmp = argv; *tmp != NULL; tmp++ ) {
8       printf("Argument %d = %s\n", tmp - argv, *tmp);
9     }
10    return 0;
11  }
```


Tracing Functions

`truss` – Traces system calls made by a process

- How to run:
  - `$truss <args to truss> <pgm> <pgm args>`
  - `$truss -p <pid_of_running_pgm> <args to truss>`
- `args` to `truss` include:
  - Trace output redirection
  - Specification of certain system calls
  - Address argument dereferencing (>= 2.7)
  - Library calls (>= 2.7)

Tracing Functions

```
$ truss pwd
execve("/usr/bin/pwd", 0xFFBEF70C, 0xFFBEF714)  argc = 1
open("/dev/zero", O_RDONLY)                     = 3
mmap(0x00000000, 8192, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE, 3, 0) = 0xFF3A0000
open("/usr/openwin/lib/libc.so.1", O_RDONLY)    Err#2 ENOENT
open("/usr/dt/lib/libc.so.1", O_RDONLY)         Err#2 ENOENT
open("/usr/lib/libc.so.1", O_RDONLY)            = 4
fstat(4, 0xFFBEF2A4)                            = 0
mmap(0x00000000, 8192, PROT_READ|PROT_EXEC, MAP_PRIVATE, 4, 0) =
0xFF390000
mmap(0x00000000, 761856, PROT_READ|PROT_EXEC, MAP_PRIVATE, 4, 0)
= 0xFF280000
munmap(0xFF322000, 57344)                       = 0
mmap(0xFF330000, 33348, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED, 4, 655360) = 0xFF330000
close(4)                                        = 0
open("/usr/openwin/lib/libdl.so.1", O_RDONLY)   Err#2 ENOENT
.........................
```



```
.........................
lstat64("/home/hotchkis", 0xFFBEE798)           = 0
lstat64("/home/hotchkis/..", 0xFFBEE798)        = 0
llseek(4, 0xFFFFFFFFFFFFFFBA, SEEK_CUR)         = 1342
close(4)                                        = 0
close(3)                                        = 0
/home/hotchkis/220/LF/overview
write(1, " / h o m e / h o t c h k"., 31)       = 31
llseek(0, 0, SEEK_CUR)                          = 11536
_exit(0)
```

Exercise: Overview

- Objectives
- Tasks
- Discussion
- Solutions