Threads

Concurrent Functions

류은경
ekryu@knu.ac.kr

Objectives

Ideas and Skills

- Threads of execution
- Multithreaded programs
- Creating and destroying threads
- Sharing data between threads safely using mutex locks
- Synchronizing data transfer using condition variables
- Passing multiple arguments to a thread

System Calls and Functions

- `pthread_create`, `pthread_join`
- `pthread_mutex_lock`, `pthread_mutex_unlock`
- `pthread_cond_wait`, `pthread_cond_signal`

Threads : Concurrent Functions

14.1 Doing Several Things at Once

14.2 Threads of Execution

14.3 Interthread Cooperation

14.4 Comparing Threads with Processes

- **Using `fork` and `exec`, we can run several programs at the same time.**

- **What if** we want to **run several functions at the same time** or several invocations of the same function?

Threads : Concurrent Functions

14.1 Doing Several Things at Once

14.2 Threads of Execution

14.3 Interthread Cooperation

14.4 comparing Threads with Processes

14.2.1 A Single-Threaded Program

```
/* hello_single.c - a single threaded hello world program */
 #include  <unistd.h>  // for sleep
#include  <stdio.h>
#define NUM     5

main()
{
        void    print_msg(char *);

        print_msg("hello");
        print_msg("world\n");
}
void print_msg(char *m)
{
        int i;
        for(i=0 ; i<NUM ; i++){
                printf("%s", m);
                fflush(stdout);
                sleep(1);
        }
}
```

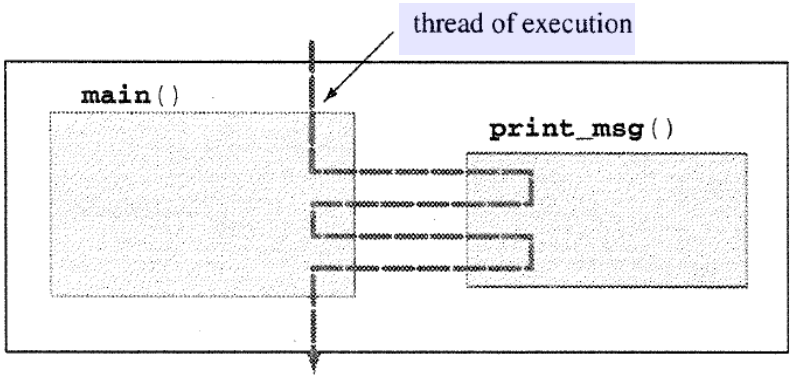


FIGURE 14.1

Single thread of execution.

14.2.2 A Multithreaded Program

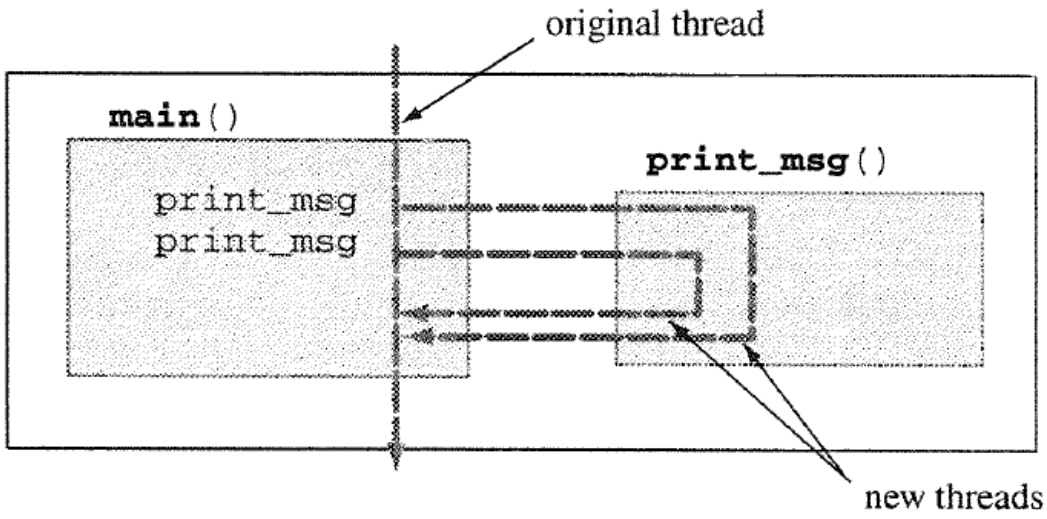
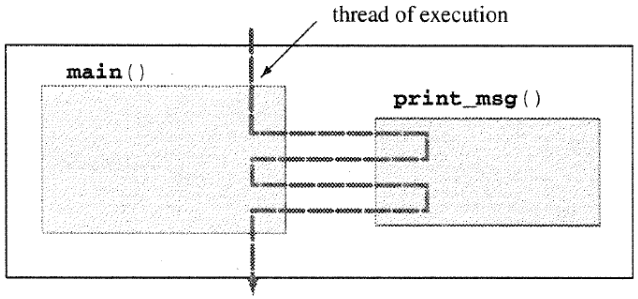


FIGURE 14.2

Multiple Threads of Execution.



one process,
two functions,
one thread

FIGURE 14.1

Single thread of execution.

```c
/* hello_multi.c - a multi-threaded hello world program */
#include  <stdio.h>
#include  <pthread.h>

#define NUM     5

main()
{
        pthread_t t1, t2;                       /* two threads */

        void    *print_msg(void *);

        pthread_create(&t1, NULL, print_msg, (void *)"hello");
        pthread_create(&t2, NULL, print_msg, (void *)"world\n");
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
}

void *print_msg(void *m)
{
        char *cp = (char *) m;
        int i;
        for(i=0 ; i<NUM ; i++){
                printf("%s", m);
                fflush(stdout);
                sleep(1);
        }
        return NULL;
}
```

```
$ cc hello_multi.c -lpthread -o hello_multi
$ ./hello_multi
helloworld
helloworld
helloworld
helloworld
helloworld
$
```

```
root@DESKTOP-K4MA2V5:~# ./hello_multi
helloworld
helloworld
helloworld
helloworld
helloworld
root@DESKTOP-K4MA2V5:~# ./hello_multi
helloworld
world
hellohelloworld
helloworld
helloworld
root@DESKTOP-K4MA2V5:~# ./hello_multi
helloworld
world
hellohelloworld
helloworld
helloworld
root@DESKTOP-K4MA2V5:~#
```

Threads : Concurrent Functions

14.1 Doing Several Things at Once

14.2 Threads of Execution

14.3 Interthread Cooperation

14.4 comparing Threads with Processes

14.3 Interthread Cooperation

- **Processes communicate with each other**
  using **pipes, sockets, signals, exit/wait, and the environment.**

- **Threads** in a single process **can communicate by** setting and reading these **global variables**

```c
/* incprint.c - one thread increments, the other prints */

#include  <stdio.h>
#include  <pthread.h>

#define NUM     5

int     counter = 0;

main()
{
        pthread_t t1;                   /* one thread */
        void      *print_count(void *); /* its function */
        int       i;

        pthread_create(&t1, NULL, print_count, NULL);
        for( i = 0 ; i<NUM ; i++ ){
                counter++;
                sleep(1);
        }

        pthread_join(t1, NULL);
}
void *print_count(void *m)
{
        int i;
        for(i=0 ; i<NUM ; i++){
                printf("count = %d\n", counter);
                sleep(1);
        }
        return NULL;
}
```

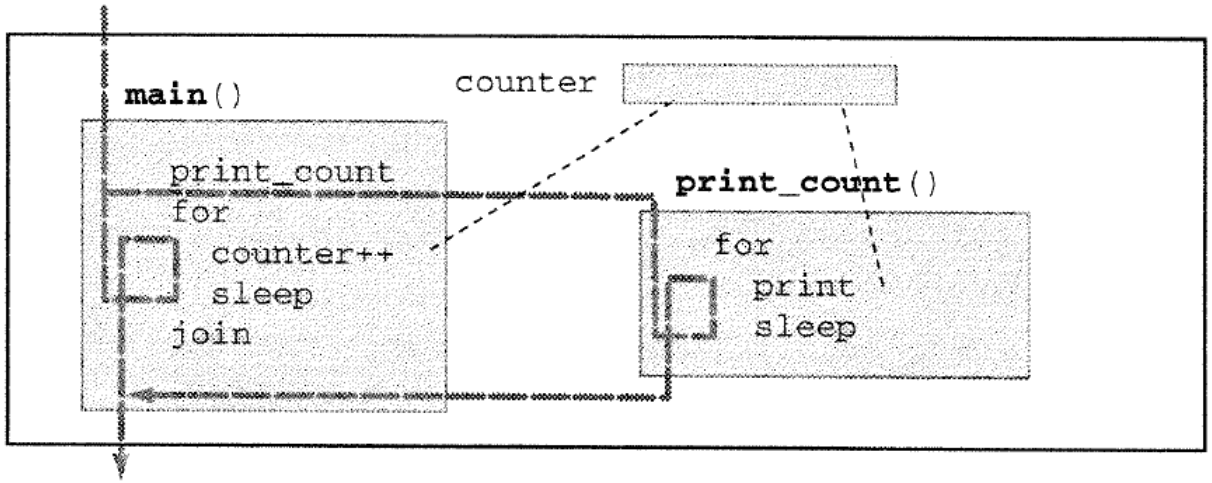


FIGURE 14.3

Two threads share a global variable.

```
$ cc incprint.c -lpthread -o incprint
$ ./incprint
count = 1
count = 2
count = 3
count = 4
count = 5
```

- **Unix `wc` program**
  - Typically, it is single threaded

```
root@DESKTOP-K4MA2V5:~# wc twordcount1.c incprint.c
  50  129  898 twordcount1.c
  38   81  528 incprint.c
  88  210 1426 합계
root@DESKTOP-K4MA2V5:~#
```

- **How can we design a multithreaded program to count and print the total number of words in two files?**

- **Version 1: Two Threads, One Counter**

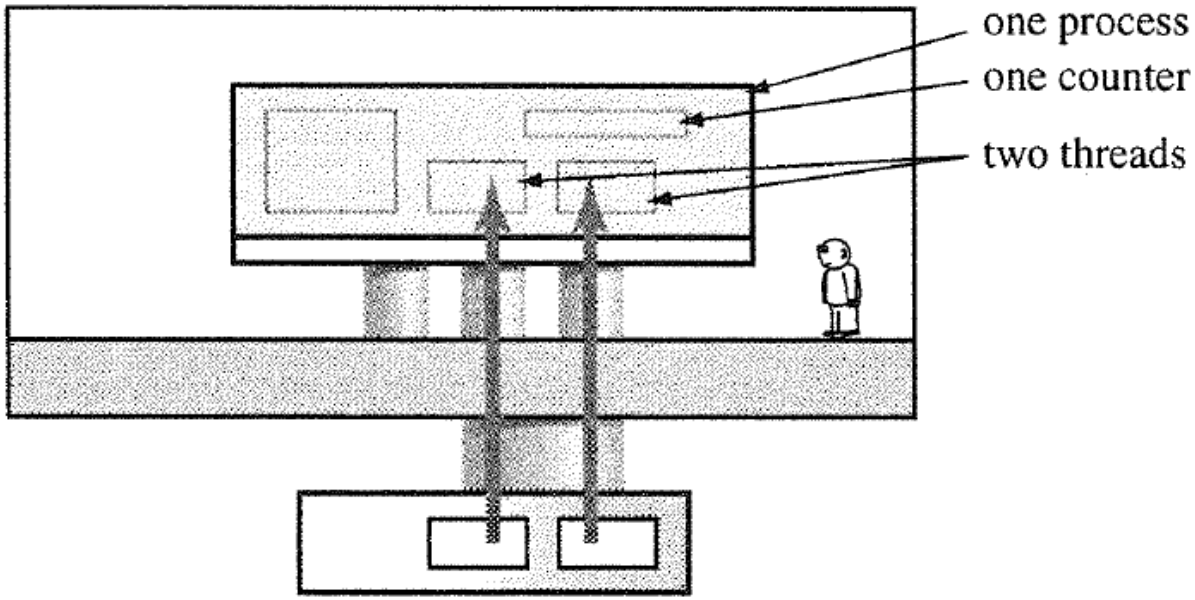


FIGURE 14.4

A common counter for two threads.

```
root@DESKTOP-K4MA2V5:~# ./twordcount1 twordcount1.c incprint.c
  217: total words
```

```
/* twordcount1.c - threaded word counter for two files. Version 1 */

#include  <stdio.h>
#include  <pthread.h>
#include  <ctype.h>

int       total_words ;

main(int ac, char *av[])
{
        pthread_t t1, t2;                   /* two threads */
        void      *count_words(void *);

        if ( ac != 3 ){
                printf("usage: %s file1 file2\n", av[0]);
                exit(1);
        }
        total_words = 0;
        pthread_create(&t1, NULL, count_words, (void *) av[1]);
        pthread_create(&t2, NULL, count_words, (void *) av[2]);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        printf("%5d: total words\n", total_words);
}
```

```
void *count_words(void *f)
{
        char *filename = (char *) f;
        FILE *fp;
        int  c, prevc = '\0';

        if ( (fp = fopen(filename, "r")) != NULL ){
                while( ( c = getc(fp)) != EOF ){
                        if ( !isalnum(c) && isalnum(prevc) )
                                total_words++;
                        prevc = c;
                }
                fclose(fp);
        } else
                perror(filename);
        return NULL;
}
```

```
$ cc twordcount1.c -lpthread -o twc1
$ ./twc1 /etc/group /usr/dict/words
45614: total words
$ wc -w /etc/group /usr/dict/words
      58 /etc/group
   45402 /usr/dict/words
   45460 total
```

Different results : ...

```
total_words++;
→ total_words = total_words + 1;
```

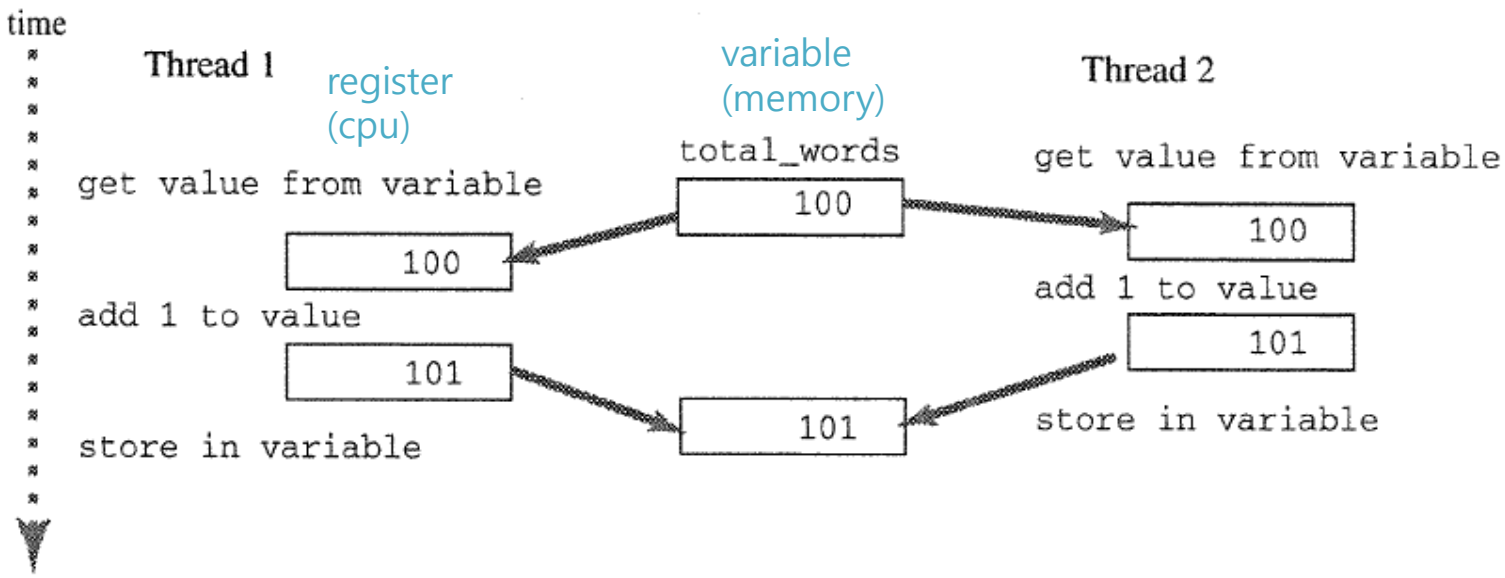


FIGURE 14.5

Two threads increment the same counter.

- **How can we prevent threads from interfering with each other?**

- **Two solutions :**
  - Version 2: Two Threads, One Counter, One **Mutex**
  - Version 3: Two Threads, **Two Counters**, Multiple Arguments to Threads

- **Version 2 :** Two Threads, One Counter, **One Mutex**
  - The threads system includes **variables** called **mutual exclusion locks.**

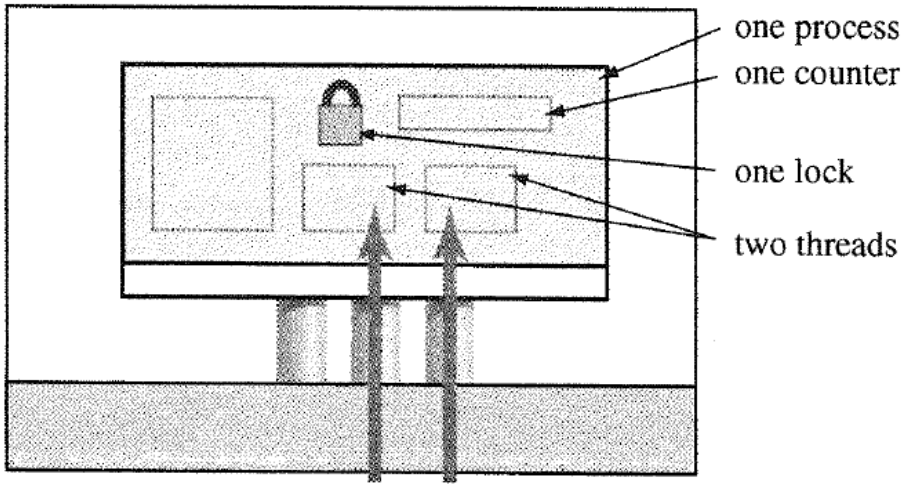


FIGURE 14.6

Two threads use a mutex to share a counter.

```
int  total_words ;
pthread_mutex_t counter_lock = PTHREAD_MUTEX_INITIALIZER;


pthread_mutex_lock(&counter_lock);
total_words++;
pthread_mutex_unlock(&counter_lock);
```

```
/* twordcount2.c - threaded word counter for two files.    */
/*                 version 2: uses mutex to lock counter   */

#include  <stdio.h>
#include  <pthread.h>
#include  <ctype.h>
int             total_words ;    /* the counter and its lock */
pthread_mutex_t counter_lock = PTHREAD_MUTEX_INITIALIZER;

main(int ac, char *av[])
{
        pthread_t t1, t2;                /* two threads */
        void      *count_words(void *);

        if ( ac != 3 ){
                printf("usage: %s file1 file2\n", av[0]);
                exit(1);
        }
        total_words = 0;
        pthread_create(&t1, NULL, count_words, (void *) av[1]);
        pthread_create(&t2, NULL, count_words, (void *) av[2]);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        printf("%5d: total words\n", total_words);
}
```

```
void *count_words(void *f)
{
        char *filename = (char *) f;
        FILE *fp;
        int  c, prevc = '\0';
        if ( (fp = fopen(filename, "r")) != NULL ){
                while( ( c = getc(fp)) != EOF ){
                        if ( !isalnum(c) && isalnum(prevc) ){
                                pthread_mutex_lock(&counter_lock);
                                total_words++;
                                pthread_mutex_unlock(&counter_lock);
                        }
                        prevc = c;
                }
                fclose(fp);
        } else
                perror(filename);
        return NULL;
}
```

- **Do We Need a Mutex? …**

- **Using a mutex makes the program run slower.**
  - Checking the lock, setting the lock, and releasing the lock for every word in both files adds up to a lot of operations

- **Version 3:** Two Threads, **Two Counters,** Multiple Arguments to Threads

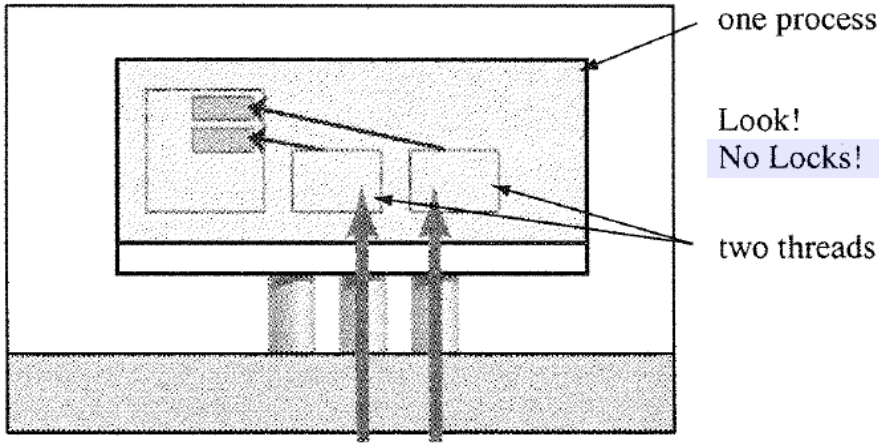


FIGURE 14.7

Each thread has a pointer to its own struct.

```c
/* twordcount3.c - threaded word counter for two files.
 *               - Version 3: one counter per file
 */

#include  <stdio.h>
#include  <pthread.h>
#include  <ctype.h>
#include  <stdlib.h>

struct arg_set {                /* two values in one arg */
               char *fname;     /* file to examine       */
               int  count;      /* number of words       */
};
```

※ `pthread_create` only lets us pass a single argument.

```
main(int ac, char *av[])
{
        pthread_t       t1, t2;          /* two threads */
        struct arg_set args1, args2;     /* two argsets */
        void            *count_words(void *);

        if ( ac != 3 ){
                printf("usage: %s file1 file2\n", av[0]);
                exit(1);
        }
        args1.fname = av[1];
        args1.count = 0;
        pthread_create(&t1, NULL, count_words, (void *) &args1);

        args2.fname = av[2];
        args2.count = 0;
        pthread_create(&t2, NULL, count_words, (void *) &args2);

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        printf("%5d: %s\n", args1.count, av[1]);
        printf("%5d: %s\n", args2.count, av[2]);
        printf("%5d: total words\n", args1.count+args2.count);
}
```

```
void *count_words(void *a)
{
        struct arg_set *args = a;       /* cast arg back to correct type */
        FILE *fp;
        int  c, prevc = '\0';

        if ( (fp = fopen(args->fname, "r")) != NULL ){
                while( ( c = getc(fp)) != EOF ){
                        if ( !isalnum(c) && isalnum(prevc) )
                                args->count++;
                        prevc = c;
                }
                fclose(fp);
        } else
                perror(args->fname);
        return NULL;
}
```

```
root@DESKTOP-K4MA2V5:~# ./twordcount1 twordcount1.c incprint.c
  217: total words
root@DESKTOP-K4MA2V5:~# ./twordcount1 twordcount1.c incprint.c
  194: total words
root@DESKTOP-K4MA2V5:~# ./twordcount1 twordcount1.c incprint.c
  204: total words
root@DESKTOP-K4MA2V5:~# ./twordcount1 twordcount1.c incprint.c
  200: total words
root@DESKTOP-K4MA2V5:~#
```

```
root@DESKTOP-K4MA2V5:~# ./twordcount3 twordcount1.c incprint.c
  134: twordcount1.c
   83: incprint.c
  217: total words
root@DESKTOP-K4MA2V5:~# ./twordcount3 twordcount1.c incprint.c
  134: twordcount1.c
   83: incprint.c
  217: total words
root@DESKTOP-K4MA2V5:~# ./twordcount3 twordcount1.c incprint.c
  134: twordcount1.c
   83: incprint.c
  217: total words
root@DESKTOP-K4MA2V5:~#
```

Threads : Concurrent Functions

14.1 Doing Several Things at Once

14.2 Threads of Execution

14.3 Interthread Cooperation

14.4 Comparing Threads with Processes

Processes	Threads
• Part of the Unix since the beginning	• Added later
• Model of the process is clear & uniform.	• There are different type of threads with different attributes. • The examples we have looked at use an interface called POSIX threads.
• **Each process has its own data space, file descriptors, and process ID number.**	• **Threads share one** data **space, set of file descriptors, and process ID number.** • But a thread, like process,has **its own PC, a register set, and a stack space.**

- **All threads share the same process;**
  - If one thread calls exec, ….
  - If one thread calls exit, …
  - What if a thread causes a segmenation violation or other system error and the thread crashes? …
  - …

Threads : Concurrent Functions

14.1 Doing Several Things at Once

14.2 **Threads** of Execution

14.3 **Interthread Cooperation**

14.4 Comparing Threads with Processes

Course Lesson Plan

Week	Course Goals and Objectives
Week 1	Unix System Programming:The Big Picture
Week 2	Users, Files, and the Manual: who Is First?
Week 3	Directories and File Properties: Looking through ls
Week 4	Focus on File Systems: Writing pwd
Week 5	Connection Control: Studying stty
Week 6	Programming for Humans:Terminal Control and Signals
Week 7	Event-Driven Programming
Week 8	Mid-term Exam.
Week 9	Processes and Programs:Studying sh
Week 10	A Programmable Shell: Shell Variables and the Environment (1)
Week 11	A Programmable Shell: Shell Variables and the Environment (2)
Week 12	I/O Redirection and Pipes
Week 13	Connecting to Processes & Threads: Concurrent Functions
Week 14	Final exam.
Week 15	Project Presentation

Files

Device I/O

Multitasking

IPC