

# Ch14. Threads

## Concurrent Functions

Prof. Seokin Hong

Kyungpook National University

Fall 2019

# Doing Several Things at Once

---

- **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**?
- In this chapter, we study **threads**.

# What is in a process?

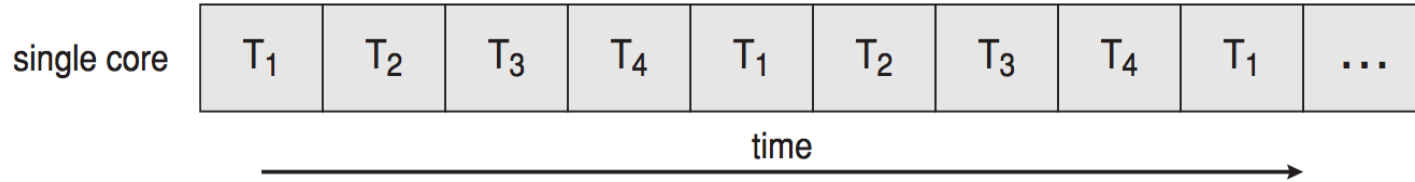
---

- A process consists of
  - **An address space**, containing
    - The code (instructions) for the running program
    - The data for the running program
  - **Thread state**, consisting of
    - The program counter (PC), indicating the next instruction
    - The stack pointer register
    - Other general purpose register values
  - **A set of OS resources**
    - Open files, network connections
- What if decompose ...
  - Address space
  - Thread state (stack, stack pointer, program counter, registers)
  - OS resources

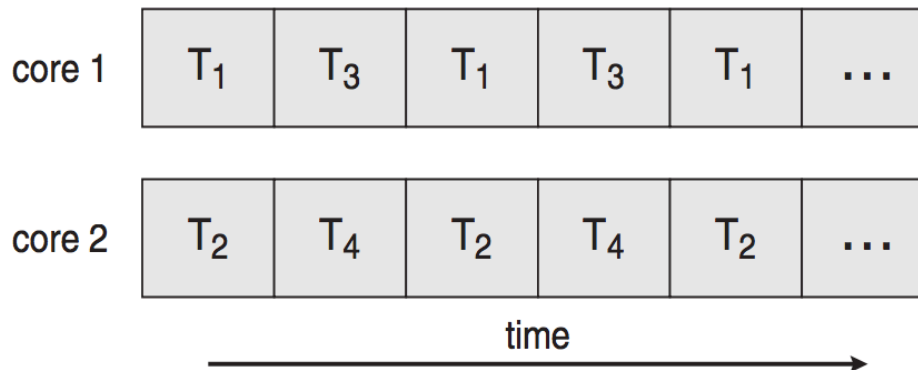
# Concurrency VS Parallelism

---

- Concurrent execution on single-core system:



- Parallelism on a multi-core systems:



# Thread: Motivation

---

- **In many cases,**

- Everybody wants to run the same code
- Everybody wants to access the same data
- Everybody has the same privileges
- Everybody uses the same resources (open files, network connections, etc)

- **But, everybody want to have **multiple hardware execution states** for concurrency and parallelism**

- An execution stack and stack pointer (SP)
- The program counter (PC), indicating the next instruction
- A set of general-purpose processor registers and their values

# Thread: Key Idea

---

- **Separation of the concept of a process**

- **Threads share a process**

- Same program code
    - Same address space
    - Same OS resources

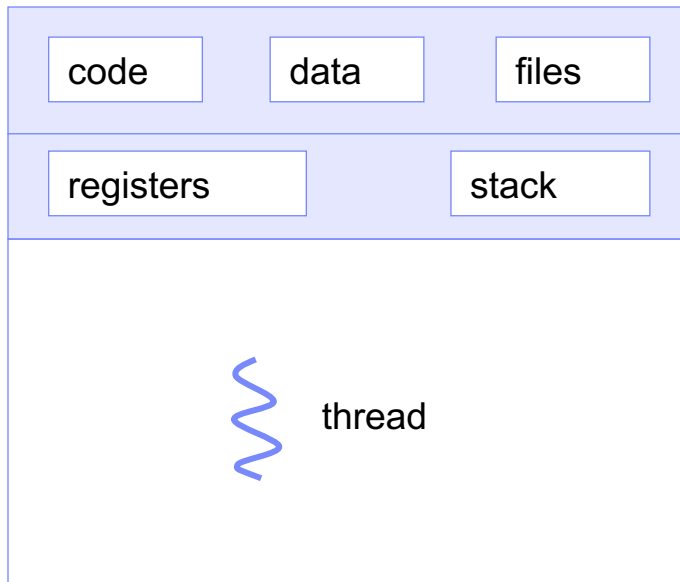
- **Each thread has its own execution state**

- Stack, stack pointer, program counter, registers
    - So, threads are schedulable

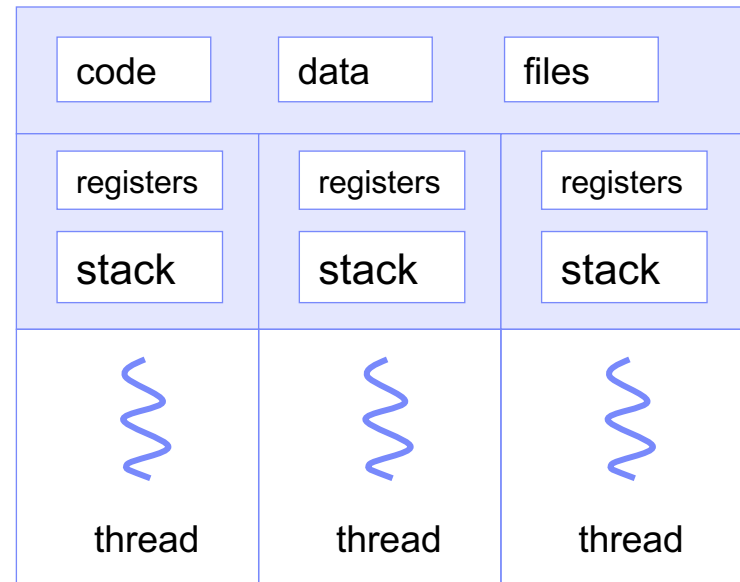
- **This execution state is usually called a thread, or a lightweight process**

# Thread: Key Idea

---



single-threaded process



multithreaded process

# Thread vs Process

---

- **Most modern operating system support two entities**
  - **Process:** defines the address space and general process attributes (such as open files, etc)
  - **Thread :** defines a sequential execution stream within a process
  
- **A thread is bound to a single process (address space)**
  - Address spaces can have multiple threads executing within them
  - Sharing data between threads is cheap: all thread see the same address space.
  - Creating thread is cheap too!
  
- **Threads become the unit of scheduling**
  - Processes / address spaces are just containers in which threads execute



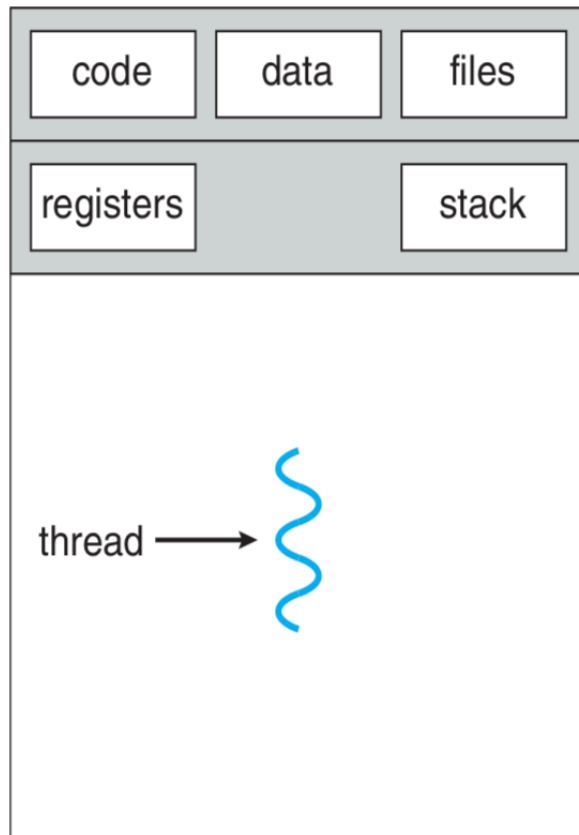
# Thread vs Process

---

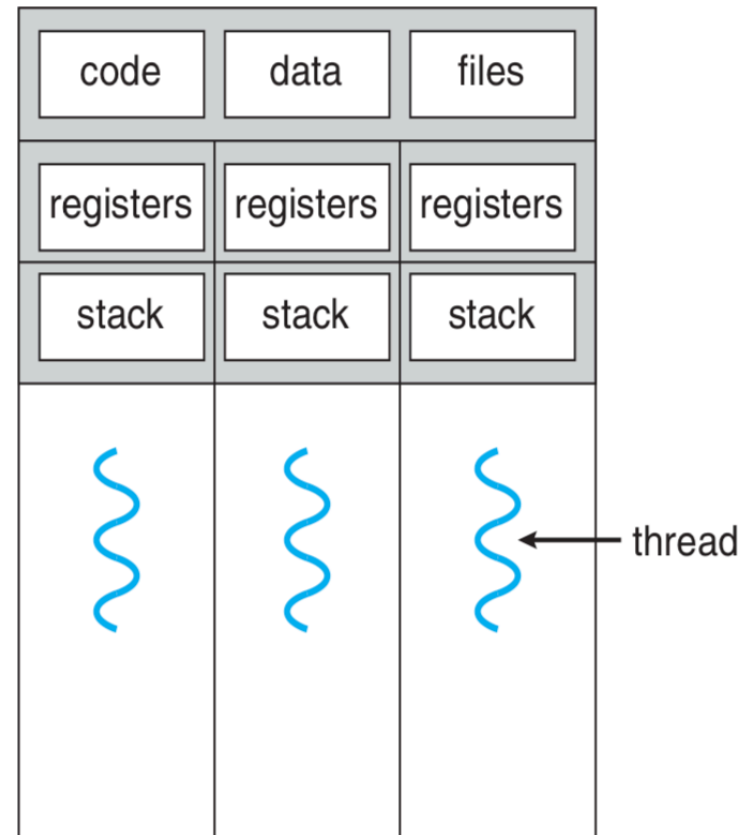
Process	Thread
Processes are heavyweight operations	Threads are lighter weight operations
Each process has its own memory space	Threads use the memory of the process they belong to
Inter-process communication is slow as processes have different memory addresses	Inter-thread communication can be faster than inter-process communication because threads of the same process share memory with the process they belong to
Context switching between processes is more expensive	Context switching between threads of the same process is less expensive
Processes don't share memory with other processes	Threads share memory with other threads of the same process

# Single and multi-threaded processes

---



single-threaded process



multithreaded process

# A Single-Threaded Program

---

```
/* hello_single.c -- a single threaded hello
world program*/

#include <unistd.h>
#include <stdio.h>
#define NUM 5

void print_msg(char*);

main()
{
    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);
    }
}
```

# Thread Creation

---

pthread_create		
purpose	Create a new thread	
include	#include <pthread.h>	
Usage	int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*func)(void*), void *arg);	
Args	thread	a pointer to a variable of type pthread_t
	attr	a pointer to a variable of type pthread_attr_t
	func	the function this new thread will run
	arg	the argument to be passed to func
Returns	0	if successful
	errcode	if not successful

# Thread Join

---

pthread_join		
purpose	Wait for termination of a thread	
include	#include <pthread.h>	
Usage	int pthread_join(pthread_t thread, void **retval);	
Args	thread	the thread to wait for
	retval	points to a variable to receive the return value from the thread
Returns	0	if successful
	errcode	if not successful

※ **pthread\_join** blocks the calling thread until the specified thread terminates.

# Ex1: A Multi-Threaded Program

```
/* hello_multi.c -- a multi-threaded hello world program*/

#include <unistd.h>
#include <stdio.h>
#include <pthread.h>
#define NUM 5

void *print_msg(void*);

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

    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)
{
    int i;
    for(i=0; i<NUM; i++)
    {
        printf("%s", m);
        fflush(stdout);
        sleep(1);
    }
}
```

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

# Inter-thread Cooperation

---

- Processes communicate with each other using pipes, sockets, signals, exit/wait, and the environment.
- **Threads execute functions in a single process, so threads share global variables.**
- **Threads can communicate by setting and reading these global variables.**
- **Simultaneous access to memory is a powerful, but dangerous.**

# Inter-thread Cooperation, Ex2 : incprint.c

---

```
// incprint.c - one thread increments, the other prints
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define NUM 5

int counter = 0;
void *print_count(void*); // its function

void main(){
    pthread_t t1;          // one thread
    int i;

    // create a thread
    pthread_create(&t1, NULL, print_count, NULL);
    for(i = 0; i < NUM; i++){
        counter++;
        sleep(1);
    }
    // wait for a thread to be completed
    pthread_join(t1, NULL);
    return 0;
}

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



# Inter-thread Cooperation, Ex3 : twordcount1.c

---

## ■ Unix wc program : ...

```
seokin@compasslab1 $ wc twordcount1.c incprint.c
```

```
45 132 992 twordcount1.c
33 80 572 incprint.c
78 212 1564 total
```

Number of line

Number of words

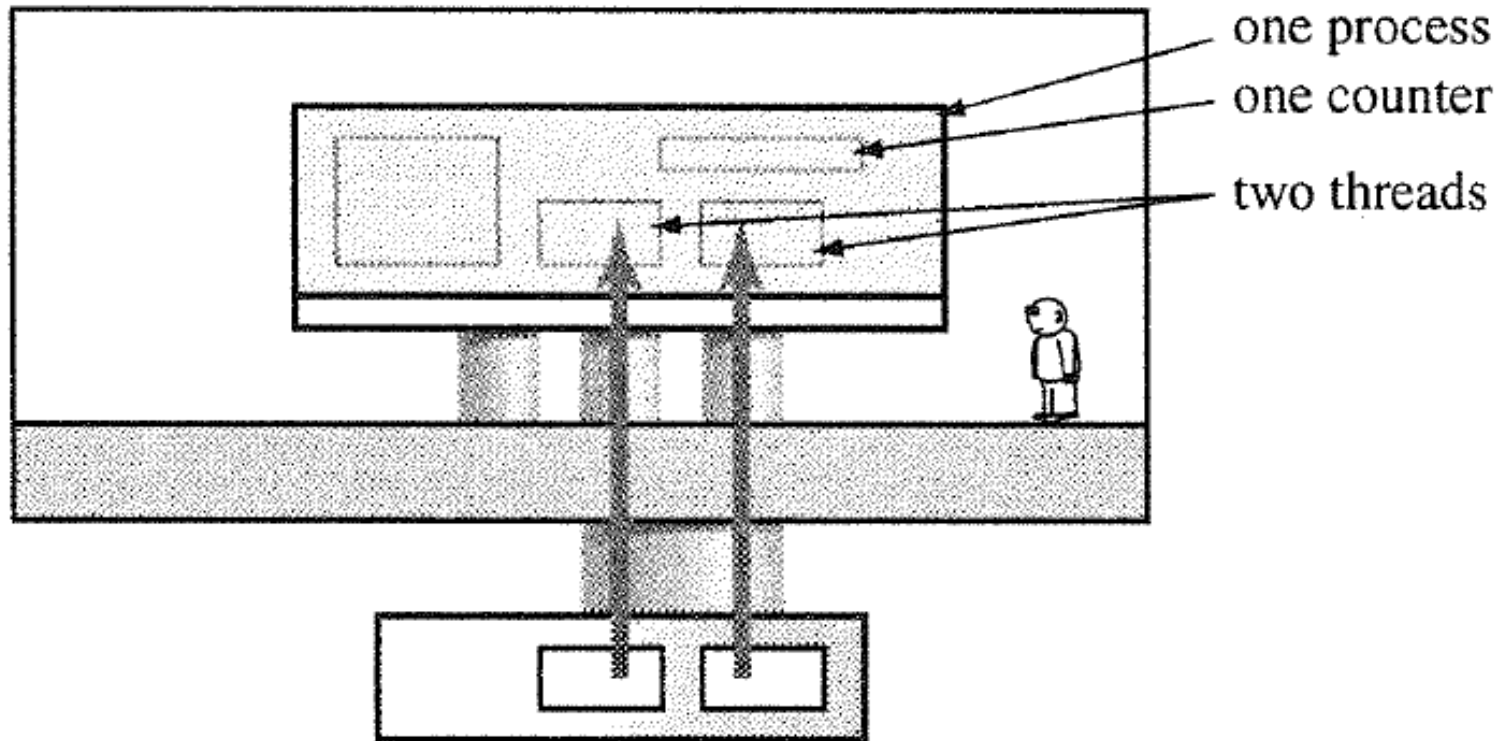
Number of bytes

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

# Inter-thread Cooperation, Ex3 : twordcount1.c

---

- Version 1: Two Threads, One Counter



# Inter-thread Cooperation, Ex3 : twordcount1.c

---

```
/* twordcount1.c - threaded word counter for two files. Ver1.0 */

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

int total_words;

main(int ac, char *av[])
{
    pthread_t t1, t2;
    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);
}
```

# Inter-thread Cooperation, Ex3 : twordcount1.c

---

```
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;
}
```

**isalnum():** returns non-zero value if c is a digit or a letter, else it returns 0

# Inter-thread Cooperation, Ex3 : twordcount1.c

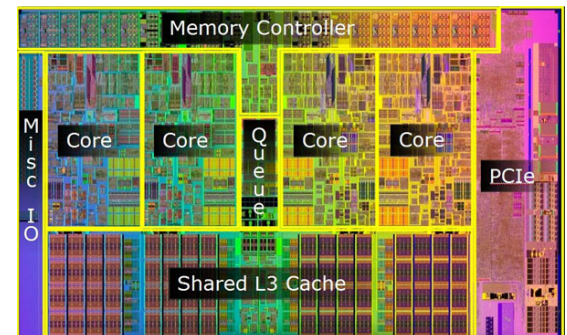
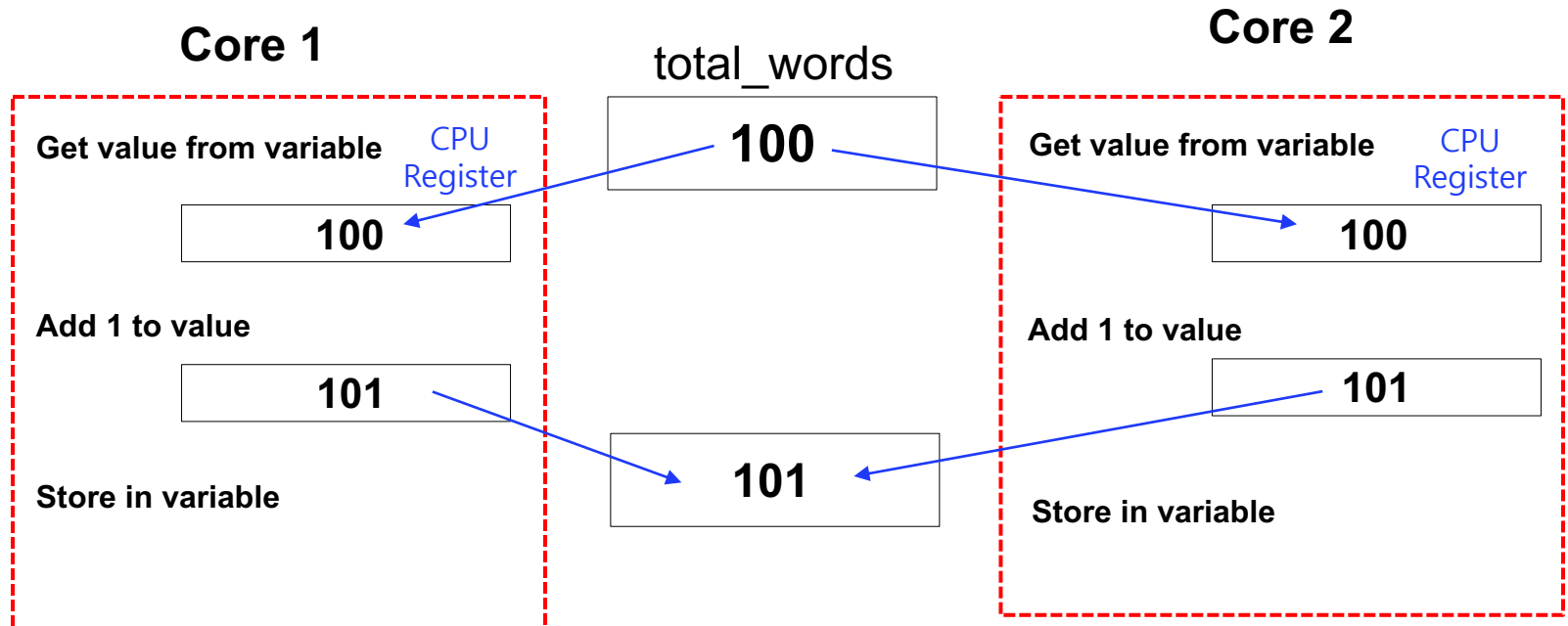
---

```
seokin@compasslab1 $ ./twordcount singthr.c twordcount.c
160: total words
seokin@compasslab1$ ./twordcount singthr.c twordcount.c
158: total words
seokin@compasslab1$ ./twordcount singthr.c twordcount.c
161: total words
seokin@compasslab1$ ./twordcount singthr.c twordcount.c
161: total words
seokin@compasslab1$ ./twordcount singthr.c twordcount.c
159: total words
seokin@compasslab1$ ./twordcount singthr.c twordcount.c
156: total words
```

**Different results ! Why?**

# Inter-thread Cooperation, Ex3 : twordcount1.c

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



# Inter-thread Cooperation, Ex3 : twordcount1.c

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

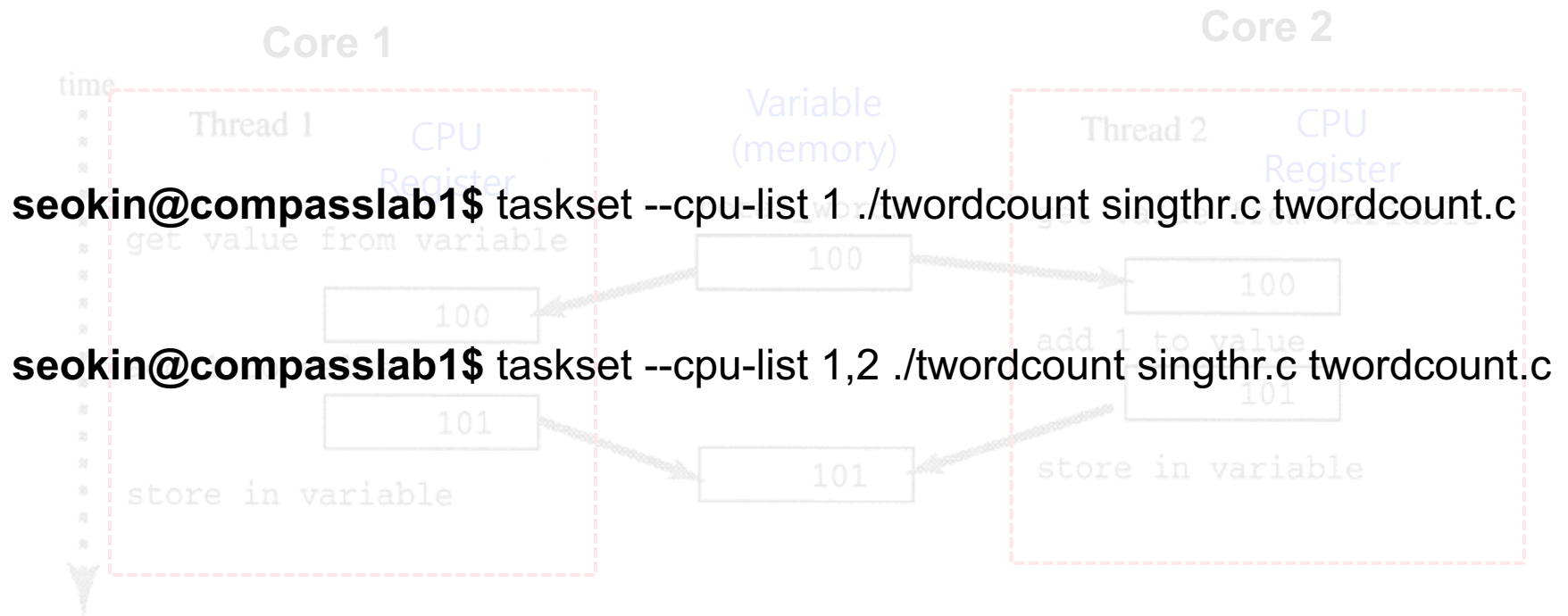


FIGURE 14.5

Two threads increment the same counter.

# Preventing thread interference

---

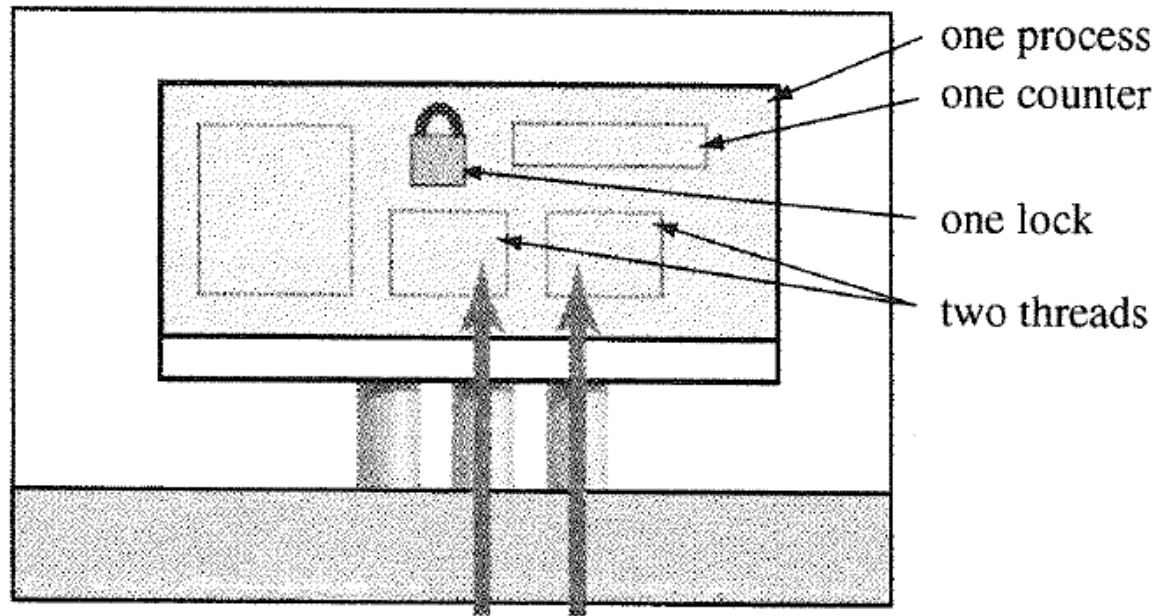
- How can we prevent threads from interfering with each other?
- **Two solutions :**
  - Version 2: Two Threads, One Counter, One Mutex
  - Version 3: Two Threads, Two Counters, Multiple Arguments to Threads



# Inter-thread Cooperation, Ex4 : twordcount2.c

---

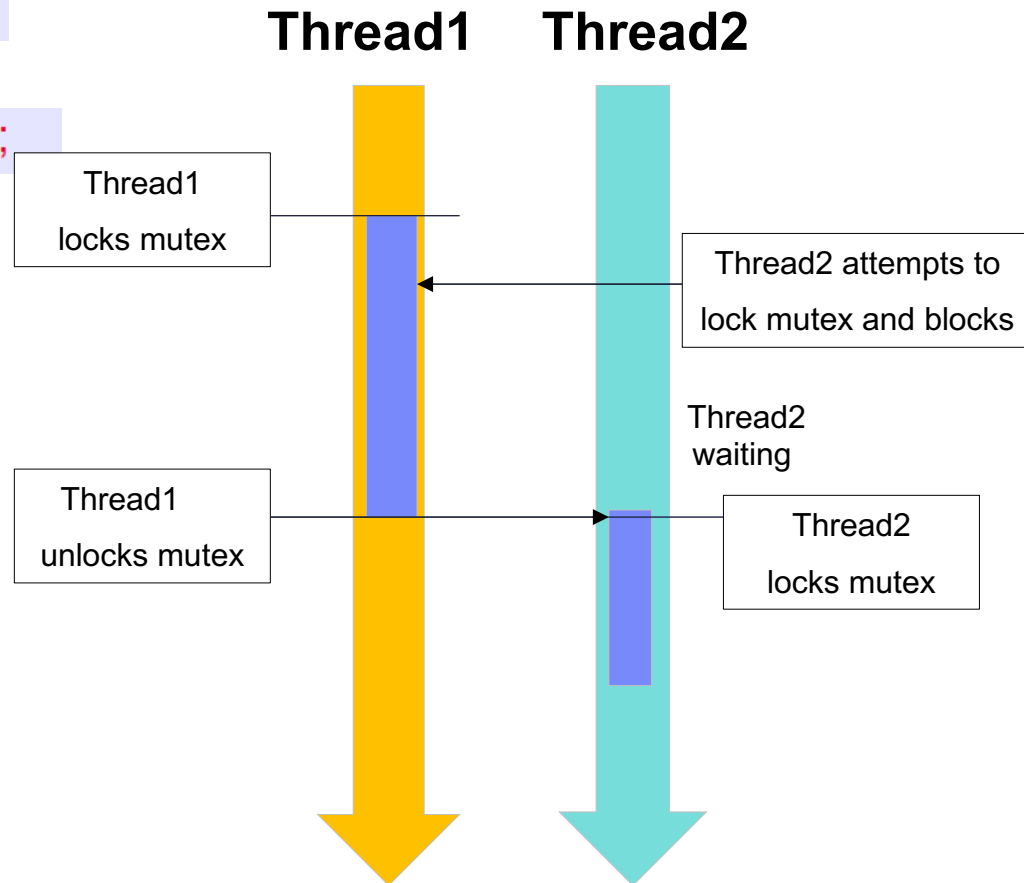
- Version 2 : Two Threads, One Counter, One Mutex
  - The threads system uses variables called **mutual exclusion lock** to prevent simultaneous access to any variable, function, or other resource



# Inter-thread Cooperation, Ex4 : twordcount2.c

```
int total_words ;  
pthread_mutex_t counter_lock = PTHREAD_MUTEX_INITIALIZER;
```

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



# pthread\_mutex\_lock

---

pthread_mutex_lock	
purpose	Wait for and lock a mutex
include	#include <pthread.h>
Usage	int pthread_mutex_lock(pthread_mutex_t *mutex);
Args	mutex      a pointer to a mutual exclusion object
Returns	0            for success errcode    for errors

# pthread\_mutex\_unlock

---

pthread_mutex_unlock	
purpose	Unlock a mutex
include	#include <pthread.h>
Usage	int pthread_mutex_unlock(pthread_mutex_t *mutex);
Args	mutex      a pointer to a mutual exclusion object
Returns	0            for success errcode    for errors

# Inter-thread Cooperation, Ex4 : twordcount2.c

---

```
/*twordcounter2.c -- threaded word counter for two files */
/*      version 2: uses mutex to lock counter      */

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

int total_words; /* the counter */
pthread_mutex_t counter_lock = PTHREAD_MUTEX_INITIALIZER; /* lock*/

void * count_words(void*);

main(int ac, char *av[])
{
    pthread_t t1, t2;      /* two threads */
    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("%d: total words\n", total_words);
}
```

# Inter-thread Cooperation, Ex4 : twordcount2.c

---

```
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?

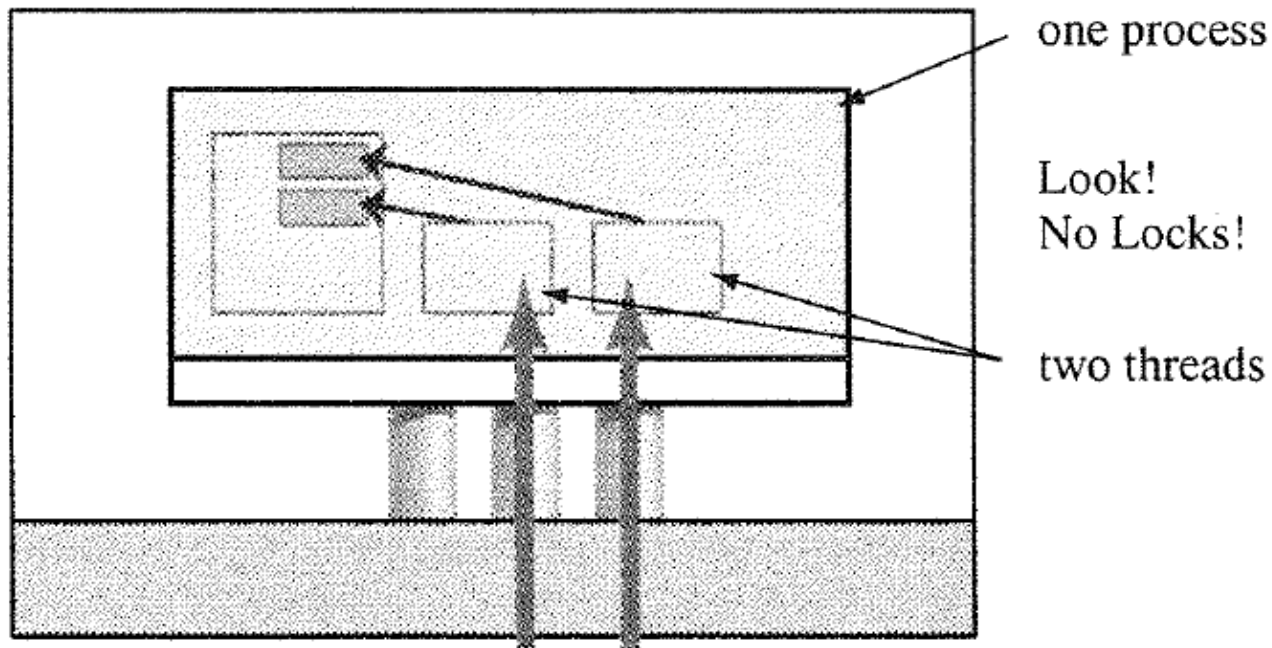
---

- If both threads might try to modify the same variable at the same time, they have to use a mutex to prevent interference.
  - Mutex ensures that the both threads have a proper view of the memory.
- 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

# Inter-thread Cooperation, Ex5 : twordcount3.c

---

- Version 3: Two Threads, Two Counters, Multiple Arguments to Threads
  - Give each thread its own counter





# Inter-thread Cooperation, Ex5 : twordcount3.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 {
    char *fname; /* file to examine */
    int count;   /* number of words */
};
```

\* **pthread\_create** only lets us pass a single argument. Thus, we need to use a structure data type to pass multiple argument to the thread.

# Inter-thread Cooperation, Ex5 : twordcount3.c

※ Passing pointers to local structs

not only eliminates the need for a mutex,  
but also gets rid of global variables.

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

    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);
}
```

# Inter-thread Cooperation, Ex5 : twordcount3.c

---

```
void *count_words(void *a)
{
    struct arg_set *args = a;
    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;
}
```

# Contents

---

- Doing Several Things at Once
- Threads of Execution
- Interthread Cooperation
- Comparing Threads with Processes
- **Inter-thread Notification**

# Inter-thread Notification

---

- How can one thread notify another thread?
  - When a counting thread finishes its work, how can it notify the original thread that its results are ready?
  - Ex)
    - \$ twordcount really-big-file tiny-file

# Functions for Condition Variables

---

pthread_cond_wait	
purpose	Blocks a thread on a condition variable
include	#include <pthread.h>
Usage	int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
Args	cond      pointer to a condition variable mutex     pointer to a mutex
Returns	0            for success errcode    for errors

# Functions for Condition Variables

---

## ■ *pthread\_cond\_wait()*

- This function is used to block on a condition variable
  - It allows a set of threads to sleep until tickled!
  - This makes processor time available to the other threads!
- called with *mutex* locked by the calling thread
- atomically release *mutex* and cause the calling thread to block on the condition variable *cond*
- upon successful return, the mutex has been locked and is owned by the calling thread.
- The mutex is used to protect *the condition variable itself*

# Functions for Condition Variables

---

pthread_cond_signal	
purpose	Unblocks a thread waiting on a condition variable
include	#include <pthread.h>
Usage	int pthread_cond_signal(pthread_cond_t *cond);
Args	cond      pointer to a condition variable
Returns	0            for success errcode    for errors



# Inter-thread Cooperation, Ex6 : twordcount4.c

---

```
/*twordcount4.c – threaded word counter for two files.
 *           – Version4: condition variable allows counter
 *           functions to report results early
 */

#include <stdio.h>
#include <pthread.h>
#include <ctype.h>
#include <stdlib.h>
struct arg_set{
    char *fname;
    int count;
};

struct arg_set *mailbox = NULL;
pthread_mutex_t lock    = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  flag     = PTHREAD_COND_INITIALIZER;
```

# Inter-thread Cooperation, Ex6 : twordcount4.c

---

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

    int reports_in = 0;
    int total_words = 0;

    if(ac!=3)
    {
        printf("usage: %s file1 file2\n",av[0]);
        exit(1);
    }

    pthread_mutex_lock(&lock); /*lock the mail box now*/
    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);
```

# Inter-thread Cooperation, Ex6 : twordcount4.c

---

```
while(reports_in<2){
    printf("MAIN: waiting for flag to go up\n");
    pthread_cond_wait(&flag, &lock);          /*wait for notification*/
    printf("MAIN: Wow! flag was raised, I have the lock\n");
    printf("%7d: %s\n", mailbox->count, mailbox->fname);
    total_words += mailbox->count;

    if(mailbox == &args1)
        pthread_join(t1,NULL);
    if(mailbox == &args2)
        pthread_join(t2,NULL);

    mailbox = NULL;
    pthread_cond_signal(&flag);
    reports_in++;
}
printf("%7d: total words\n", total_words);
}
```

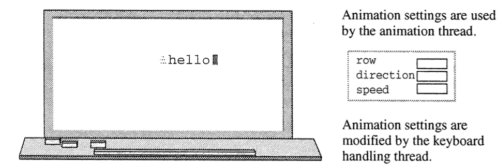
```
void *count_words(void *a)
{
    struct arg_set *args = a;
    FILE *fp;
    int c, prevc = '\0';

    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);

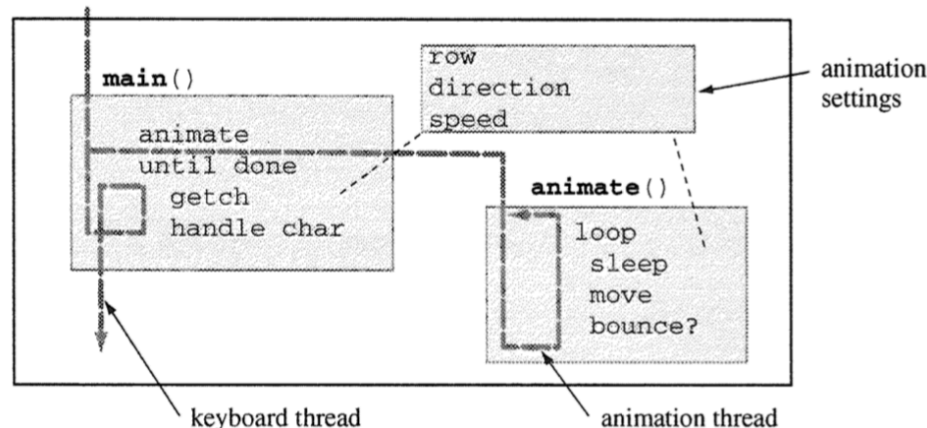
    printf("COUNT: waiting to get lock\n");
    pthread_mutex_lock(&lock);      /*get the mailbox*/
    printf("COUNT: have lock, storing data\n");
    if (mailbox !=NULL)
        pthread_cond_wait(&flag, &lock);

    mailbox = args;                  /*put ptr to our args there */
    printf("COUNT: raising flag\n");
    pthread_cond_signal(&flag);      /*raise the flag*/
    printf("COUNT: unlocking box\n");
    pthread_mutex_unlock(&lock);     /*release the mailbox */
    return NULL;
}
```

# Thread and Animation



- Externally, users see two independent flows of activity:
  - Animation and Keyboard control
- Threads let us separate the animation code from the keyboard-input code
  - If animation code requires a high computing power to produces complicated animation effects,
    - Thread on its own processor produces faster and smoother action



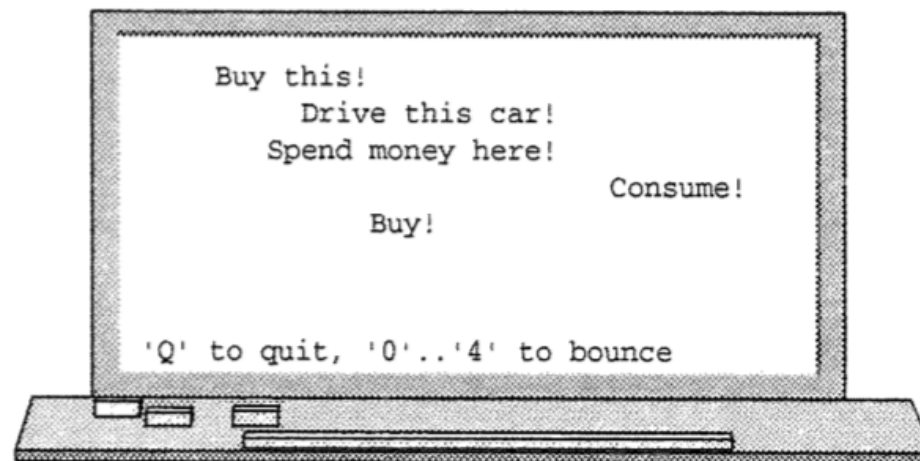
# Multiple Animation, Ex7: tanimate.c

---

## ■ “tanimate.c”

- Accepts up to ten command-line strings and animates each of arguments on a separate line.
- Runs an animation function in several threads.
  - Each instance of the function is passed a different set of arguments

\$ tanimate 'Buy this' 'Drive this car' 'Spend Money here' 'Consume' 'buy!'



# Multiple Animation, Ex7: tanimate.c

---

```
#include <stdio.h>
#include <curses.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <termios.h>

#define MAXMSG 10 /* limit to number of strings */
#define TUNIT 20000 /* timeunits in microseconds */

struct propset
{
    char *str; /* the message */
    int row; /* the row */
    int delay; /* delay in time units */
    int dir; /* +1 or -1 */
};

pthread_mutex_t mx = PTHREAD_MUTEX_INITIALIZER;

void set_cr_noecho_mode(void);
int setup( int, char *[], struct propset []);
```

# Multiple Animation, Ex7: tanimate.c

---

```
int main(int ac, char *av[])
{
    int c;          /* user input */
    pthread_t  thrds[MAXMSG]; /* the threads */
    struct propset props[MAXMSG]; /* properties of string */
    int num_msg;    /* number of strings */
    int i;

    if( ac == 1)
    {
        printf("usage: tanimate string..\n");
        exit(1);
    }

    num_msg = setup(ac-1, av+1, props);

    /* create all the threads */
    for ( i = 0; i < num_msg; i++ )
        if( pthread_create( &thrds[i], NULL, animate, &props[i]))
        {
            fprintf(stderr, "error creating thread");
            endwin();
            exit(0);
        }
}
```



# Multiple Animation, Ex7: tanimate.c

---

main()

```
/* process user input */
while(1)
{
    c = getchar();
    if ( c == 'Q' ) break;
    if ( c == ' ' )
        for( i = 0; i < num_msg; i++ )
            props[i].dir = -props[i].dir;

    if ( c >= '0' && c <= '9' )
    {
        i = c - '0';
        if( i < num_msg )
            props[i].dir = -props[i].dir;
    }
}

/* cancel all the threads */
pthread_mutex_lock(&mx);
for ( i = 0; i < num_msg; i++)
    pthread_cancel(thrds[i]);

endwin();
return 0;
}
```

# Multiple Animation, Ex7: tanimate.c

---

```
int setup( int nstrings, char * strings[], struct propset props[])
{
    int num_msg = ( nstrings > MAXMSG ? MAXMSG : nstrings );
    int i;

    /* assign rows and velocities to each string */
    srand( getpid() );
    for ( i=0; i < num_msg; i++)
    {
        props[i].str = strings[i]; /* the message */
        props[i].row = i;          /* the row */
        props[i].delay = 1 + (rand()%15); /* a speed */
        props[i].dir = ((rand()%2)?1:-1); /* +1 or -1 */
    }

    /* set up curses */
    initscr();
    set_cr_noecho_mode();

    clear();
    mvprintw(LINES-1, 0, "'Q' to quit, '0'..'%.d' to bounce", num_msg-1);

    return num_msg;
}
```

# Multiple Animation, Ex7: tanimate.c

---

```
/* the code that runs in each thread */
void *animate(void *arg)
{
    struct propset *info = arg;      /* point to info block */
    int len = strlen(info->str)+2;    /* +2 for padding */
    int col = rand()%(COLS-len-3);    /* space for padding */

    while( 1 )
    {
        usleep(info->delay*TUNIT);
        pthread_mutex_lock(&mx);      /* only one thread */
        move( info->row, col );        /* can call curses */
        addch(' ');                   /* at the same time */
        addstr( info->str );           /* Since I doubt it is */
        addch(' ');                   /* reentrant */
        move( LINES-1, COLS-1 );      /* park cursor */
        refresh();                    /* and show it */
        pthread_mutex_unlock(&mx);    /* done with curses */

        /* move item to next column and check for bouncing */

        col += info->dir;

        if ( col <= 0 && info->dir == -1 )
            info->dir = 1;
        else if ( col+len >= COLS && info->dir == 1 )
            info->dir = -1;
    }
}
```

# Multiple Animation, Ex7: tanimate.c

---

```
void set_cr_noecho_mode(void)
/*
 * purpose: put file descriptor 0 into chr-by-chr mode and noecho mode
 * method: use bits in termios
 */
{
    struct termios ttystate;

    tcgetattr( 0, &ttystate);          /* read curr. setting */
    ttystate.c_lflag    &= ~ICANON;    /* no buffering */
    ttystate.c_lflag    &= ~ECHO;      /* no echo either */
    ttystate.c_cc[VMIN]  = 1;           /* get 1 char at a time */
    tcsetattr( 0, TCSANOW, &ttystate); /* install settings */
}
```

```
$gcc tanimate.c -lcurses -lpthread -o tanimate
```