

# Chapter 4

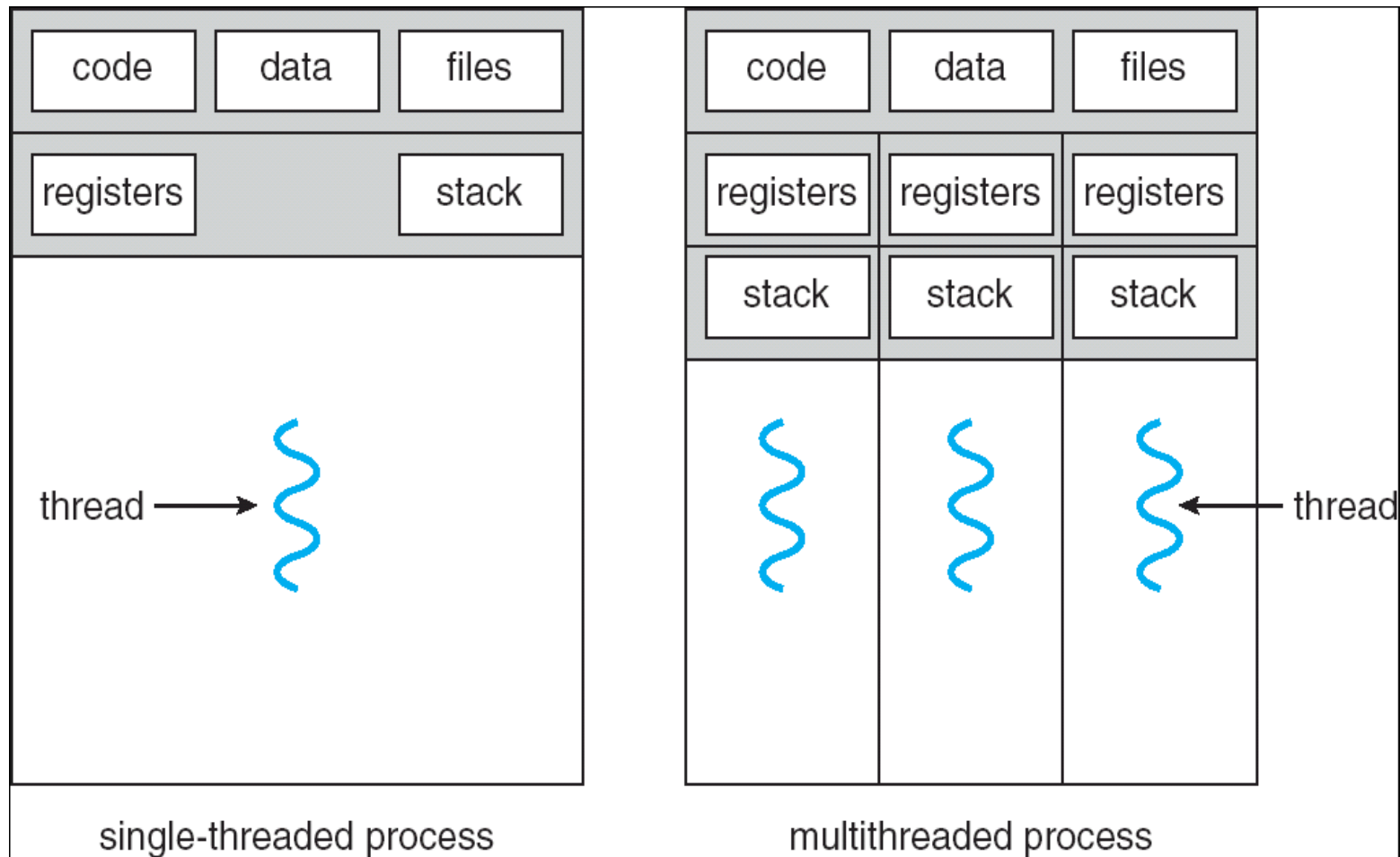
## Threads

Images from Silberschatz

# Threads

- Multiple lines of control *inside one process*
- What is shared?

- How many PCBs?





# Benefits

- Why multithread?

# Risks

- Scheduling
- Synchronization
- Hazards

# Safety

- An entity (class, method, function, ...) is thread-safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling/interleaving of those threads, with no additional synchronization on the part of the calling code.<sup>1</sup>

- Synchronization:

Atomic  
Race Condition

<sup>1</sup> Adapted from Goetz, et al, page 18.

# thread safe function calls

*name\_r()*

- `strtok_r()`
  
- `strtok()` - non-thread safe
  - exactly, why is this?
  - what data is shared?
  - who has access to it?

# Multicore

- More people have multicores
  - pressure on developers to write multithreaded code
- Multithreaded coding challenges:



# User vs Kernel Threads

- User:

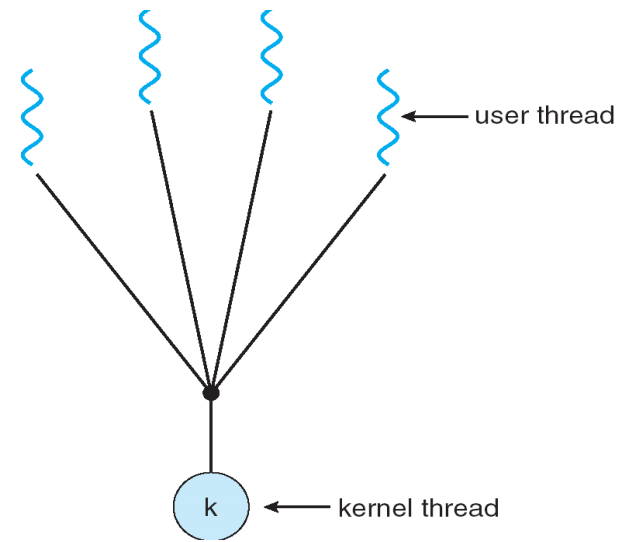
- Kernel:

	User Level Threads	Kernel Level Threads	Processes
Create Time	34	948	11,300

(chart from Stallings, W., *Operating Systems Internals and Design Principles*, 5<sup>th</sup> Edition, page 169.)

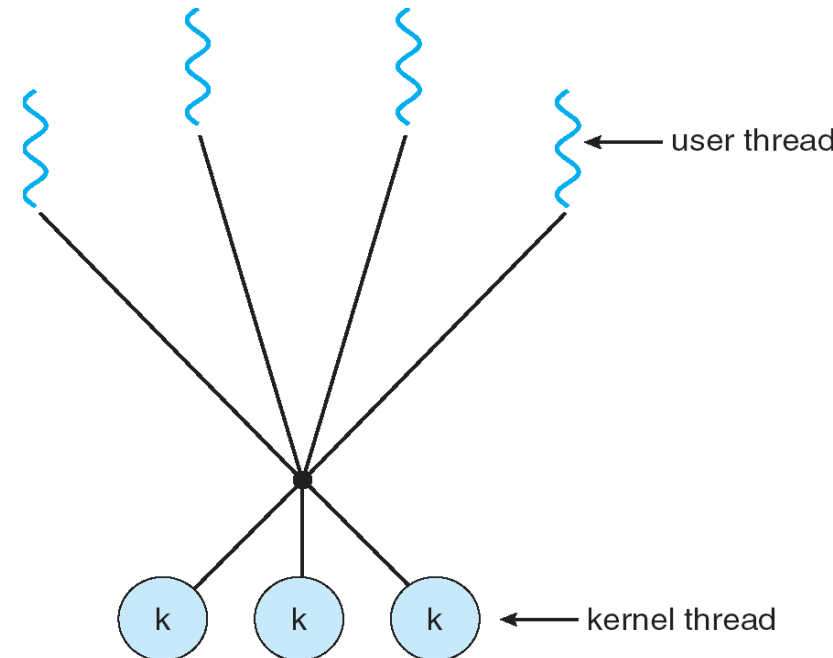
# Models

- Many-to-One



- One-to-One

- Many-to-Many



blocking system call?

# Thread Libraries

- User vs Kernel
- POSIX Pthreads
- Win32
- Java

# Pthreads

- Linux, cygwin, Solaris, etc.
  - `gcc -g -o appName appname.c -lpthread`
  - what shared library contains the PThread implementation?

```

/* This code works on Zeus!
 * link with -lpthread
 * gcc -o app -g app.o -lpthread
 */
#include <pthread.h>
#include <stdio.h>

int gSum; /* this data is shared by the thread(s) */

void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of attributes for the thread */
    int *pResult;

    if (argc != 2)
    {
        fprintf(stderr, "usage: a.out <integer value>\n");
        /*exit(1);*/
        return -1;
    }

    /* page 133 of Silberschatz */

```

```

if (atoi(argv[1]) < 0)
{
    fprintf(stderr, "Argument %d must be nonneg\n", atoi(argv[1]));
    /*exit(1);*/
    return -1;
}
gSum = 0;

/* get the default attributes */
pthread_attr_init(&attr);

/* create the thread */
pthread_create(&tid, &attr, runner, argv[1]);

/* now wait for the thread to exit */
pthread_join(tid, &pResult); // why a handle?

printf("sum = %d OR %d\n", gSum, *pResult);
}
/* page 133 of Silberschatz */

```

```
/**
 * The thread will begin control
 * in this function
 */
void *runner(void *param)
{
    int i, upper = atoi(param);

    if (upper > 0)
    {
        for (i = 1; i <= upper; i++)
        {
            gSum += i;
        }
    }

    pthread_exit((void*) &gSum);
}
/* page 133 of Silberschatz */
```

# Mutex!

```
/* This code works on Zeus!  
 * link with -lpthread  
 * gcc -o app -g app.o -lpthread  
 */  
  
#include <pthread.h>  
#include <stdio.h>  
  
int gSum; /* this data is shared by the thread(s) */  
pthread_mutex_t gMutex;  
  
void *runner(void *param); /* the thread */  
  
int main(int argc, char *argv[])  
{  
    pthread_t tid1, tid2; /* the thread identifier */  
    pthread_attr_t attr; /* set of attributes for the thread */  
    const int THREAD_PARAM_ONE = 5;  
    const int THREAD_PARAM_TWO = 6;  
  
    gSum = 0;  
  
    /* adapted from page 133 of Silberschatz */  
}
```



```

/* init the mutex */
pthread_mutex_init(&gMutex, NULL);

/* get the default attributes */
pthread_attr_init(&attr);

/* create the threads */
pthread_create(&tid1, &attr, runner, &THREAD_PARAM_ONE);
pthread_create(&tid2, &attr, runner, &THREAD_PARAM_TWO);

/* now wait for the threads to exit */
pthread_join(tid1, NULL);
pthread_join(tid2, NULL);

pthread_mutex_destroy(&gMutex);
pthread_attr_destroy(&attr);

printf("sum = %d\n", gSum);
}

/* adapted from page 133 of Silberschatz */

```

```

/**
 * The thread will begin control in this function
 */
void *runner(void *param)
{
    int i;
    // gSum = 0; // ???

    for (i = 1; i <= *(int*)param; i++)
    {
        pthread_mutex_lock(&gMutex);
        gSum += i;
        //nanosleep(100);
        pthread_mutex_unlock(&gMutex);
    }

    pthread_exit((void*) &gSum);
}
/* page 133 of Silberschatz */

```

# Pthread Functions

- pthread\_create
- pthread\_cond\_init
- pthread\_mutex\_init
- pthread\_attr\_init
- pthread\_mutex\_lock / unlock
- pthread\_cond\_wait / pthread\_cond\_timedwait
- pthread\_cond\_signal / pthread\_cond\_broadcast
- pthread\_mutex\_destroy
- pthread\_attr\_destroy
- pthread\_cond\_destroy
- pthread\_exit/pthread\_join
- pthread\_kill / pthread\_detach
- pthread\_setaffinity\_np
- sched\_setaffinity

# pthread

- `pthread_cond_wait()` / `pthread_cond_timedwait()`
- `pthread_cond_signal()` / `pthread_cond_broadcast()`
- `pthread_cond_t`

# pthread

- `pthread_rwlock_t`
- `pthread_rwlock_rdlock` / `pthread_rwlock_wrlock`
- `pthread_relock_unlock`