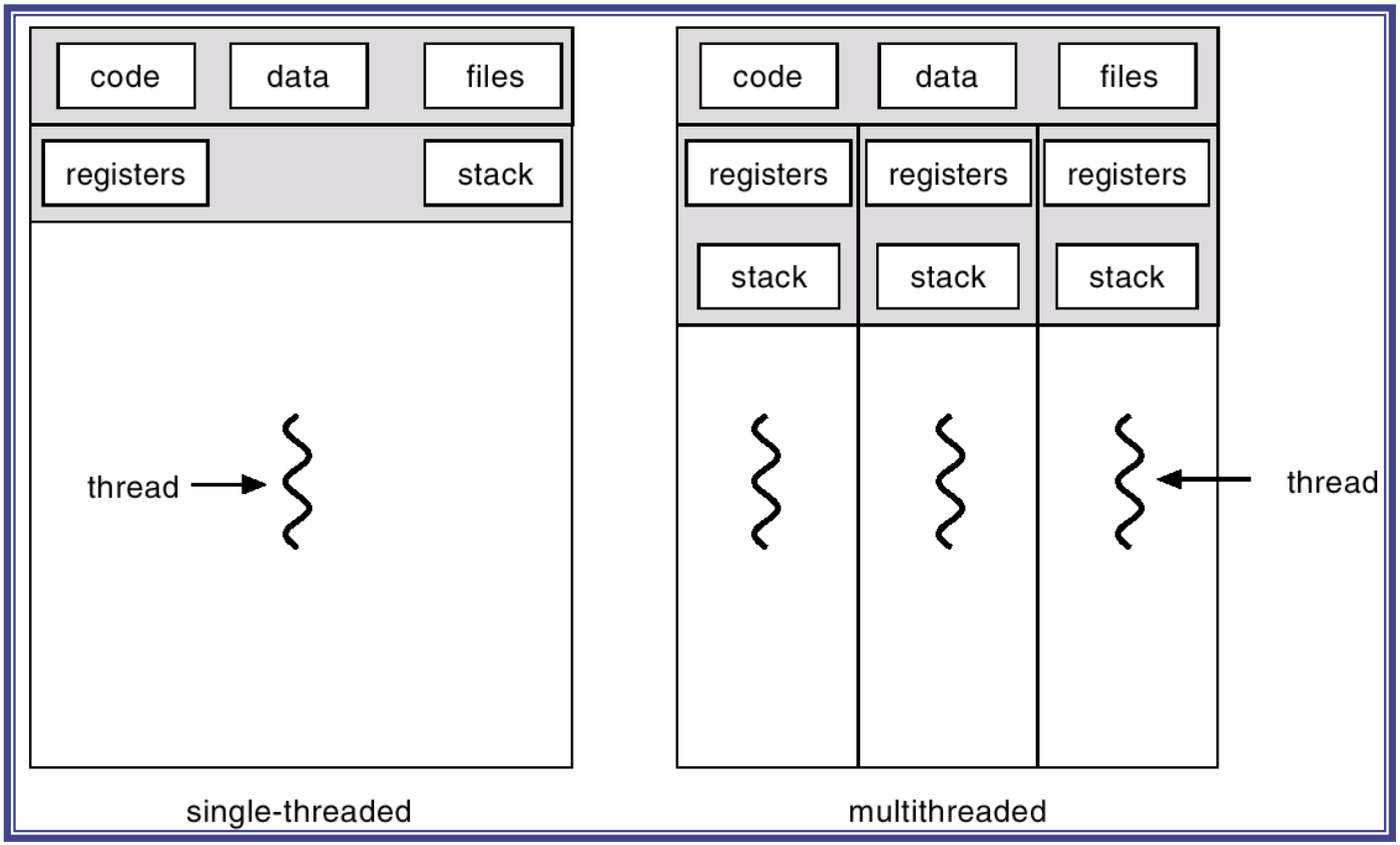
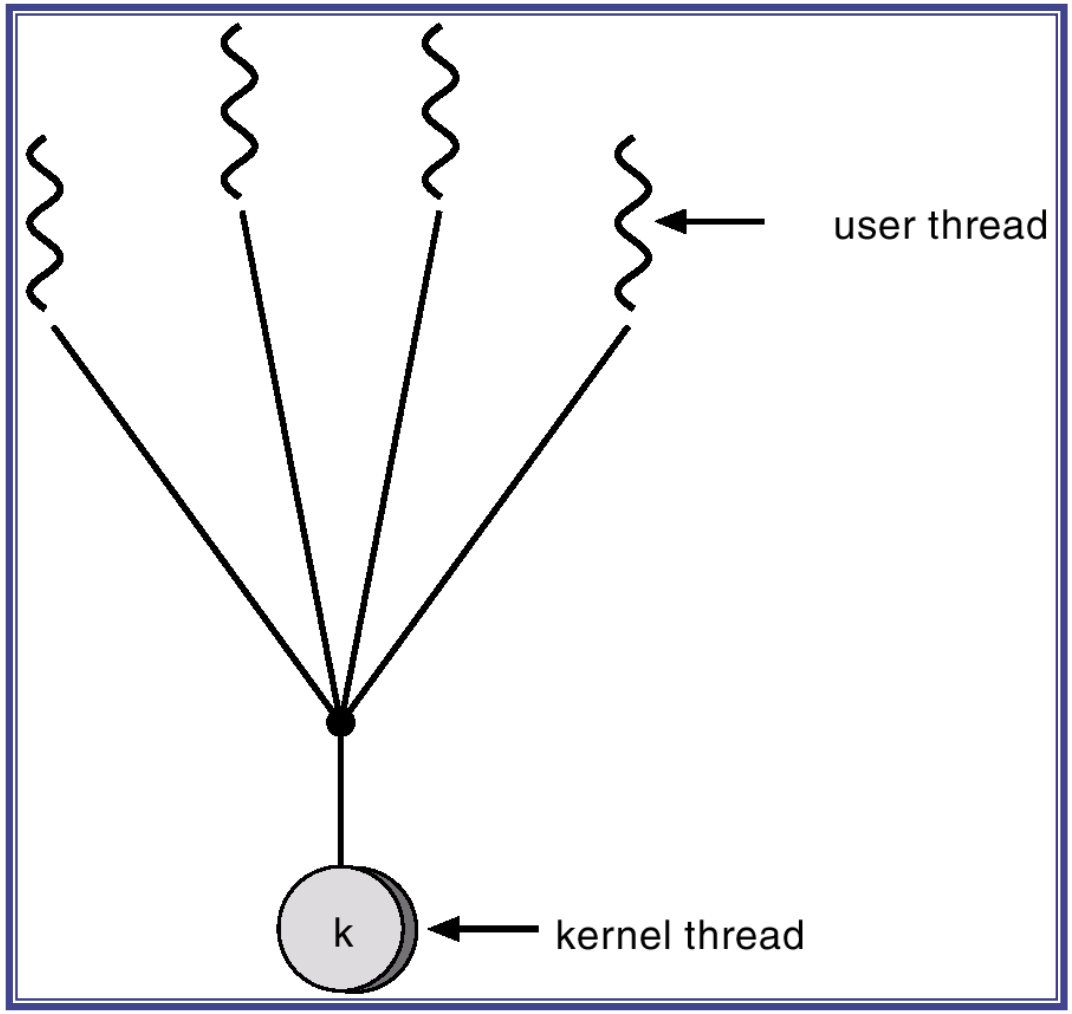
Threads

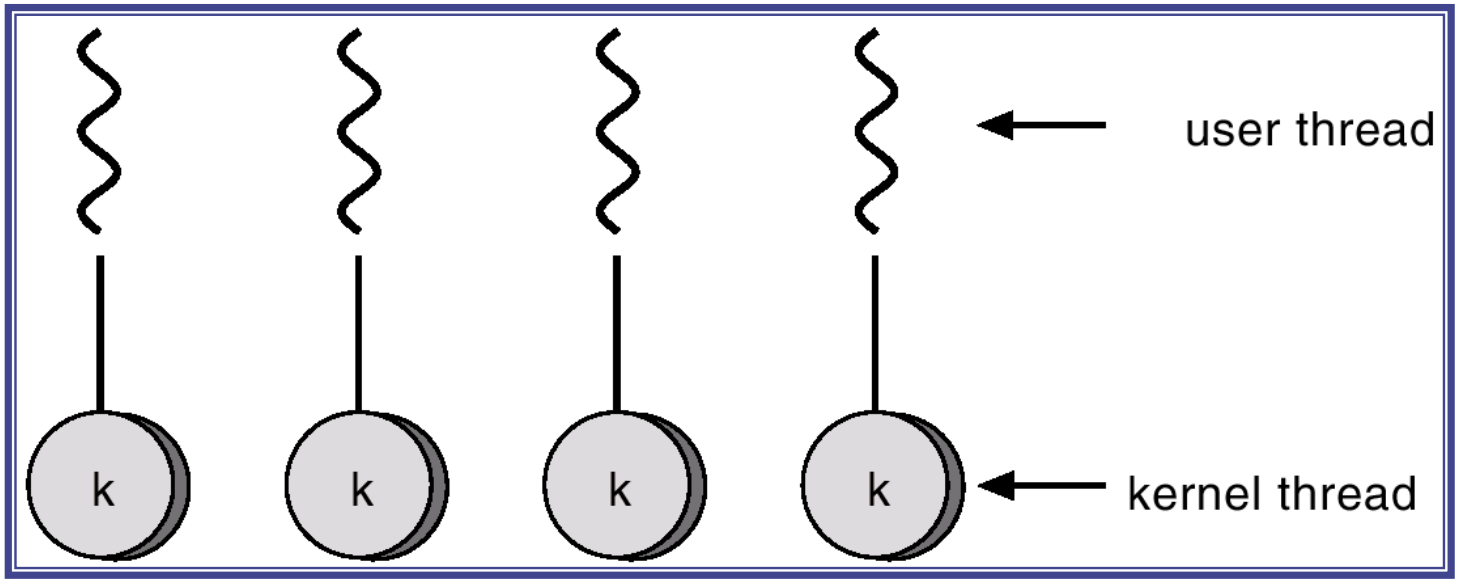
Single and Multithreaded Processes



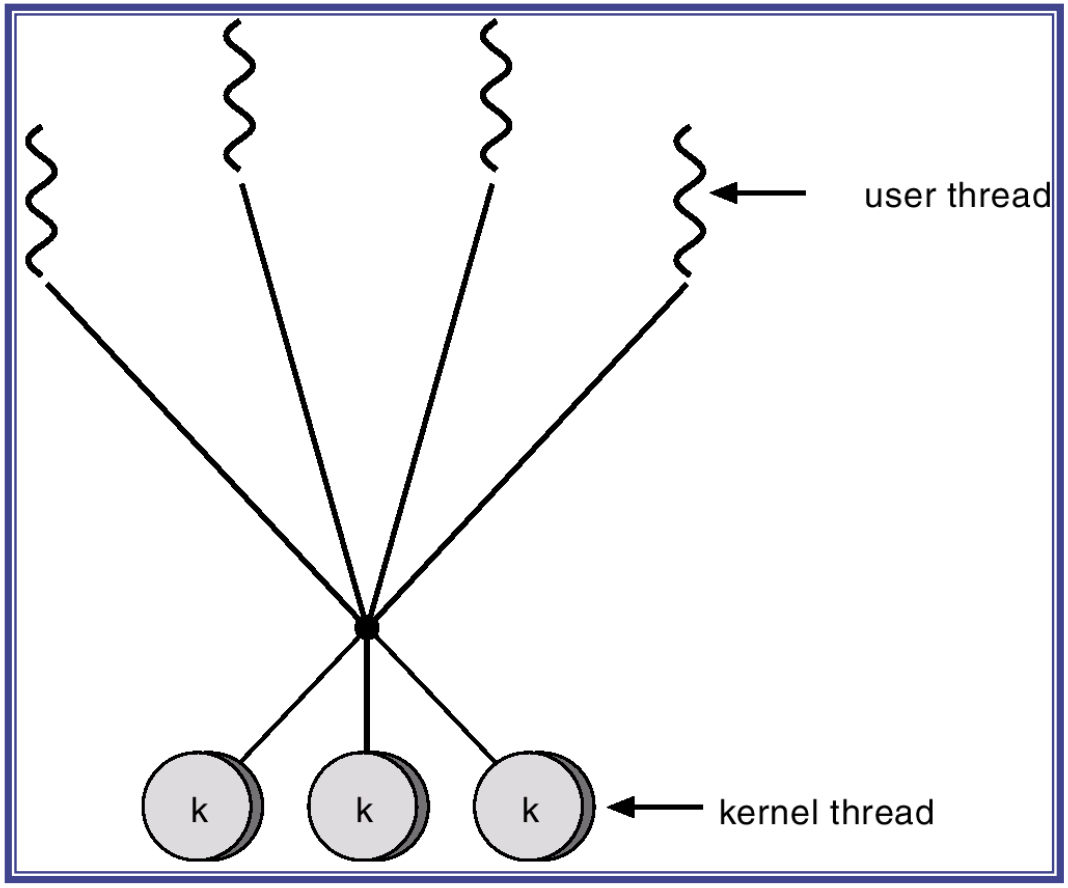
Many-to-One Model



One-to-one Model



Many-to-Many Model



Race Condition Example

```
int sum;

// This example demonstrates some of the trouble that you can
// get into with threads.  Since the example creates TWO threads
// and both threads access and change the global variable "sum,
// the output of the program varies from run to run.  This is
// called a RACE CONDITION.
int main(void)
{
  // Initalize random number generator
  srand(time(0));

  // Thread identifiers
  pthread_t tid0, tid1;

  // Create threads
  int theNum0 = 10;
  pthread_create(&tid0,NULL,summation, &theNum0);

  int theNum1 = 100;
  pthread_create(&tid1,NULL,summation, &theNum1);

  // Wait for thread to exit
  pthread_join(tid0,NULL);
  pthread_join(tid1,NULL);

  // Output sum
  cout << "Summation computed by thread is: " << sum << endl;
}
```

```
void* summation(void* param)
{
  // Delay for 0-1 seconds, simulating processing
  sleep(rand() % 2);

  // Convert (void *) parameter to (int *) parameter
  // then get the integer.
  int theNum = *((int *) param);
  cout << "Computing summation of " << theNum << endl;

  // Compute summation
  sum = 0;
  for (int i=1; i<=theNum; i++) {
    sum += i;
  }
  // Thread exit
  pthread_exit(0);
}
```

Singing pthread Example

```
#include <iostream.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <main.h>
#include <pthread.h>


void* doo(void* param);
// Precondition:  None
// Postcondition: Sings "Doo!" once a second for 60 seconds

int main(void)
{
  // Thread identifier
  pthread_t tid;

  cout << "Sing it thread!" << endl;

  // Create thread, no parameters to starting function
  pthread_create(&tid,NULL,doo,NULL);

  // Sing "Whop!" once every 5 seconds for 60 seconds
  for (int i=0; i<6; i++)
  {
    sleep(5);
    cout << "Whop!" << endl;
  }

  // Wait for thread to exit
  pthread_join(tid,NULL);

  cout << "That was mighty fine!!!" << endl;
}
```

```
void* doo(void* param)
{
  // Sing "Doo!" once a second for 30 seconds
  for (int i=0; i<30; i++)
  {
    cout << "Doo! "<<endl;
    sleep(1);
  }

  // Thread exit
  pthread_exit(0);
}
```

Summation pthread Example

```
// GLOBAL VARIABLE!!!
// This variable appears in the "static data" area of the
// process address space.  That means all threads can
// access and manipulate the variable.
int sum;

void* summation(void* param);
// Precondition:  param is declared as a (void*) pointer so that
//                it will be compatible with the pthread_create call.
//                In reality, though, param should be a the address
//                of an integer variable.]
//
// Postcondition: sets the global variable "sum" to the summation
//                of all integers up to the integer whose address
//                was passed in as param.
//
//                Example:
//                           sum=0;
//                           int theNum=10;
//                           summation(&theNum);
//                           cout << sum << endl;
//
//                           OUTPUTS: 55

int main(void)
{
  // Thread identifier
  pthread_t tid0;


  // Create thread
  int theNum0 = 10;
  pthread_create(&tid0,NULL,summation, &theNum0);

  // Wait for thread to exit
  pthread_join(tid0,NULL);

  // Output sum
  cout << "Summation computed by thread is: " << sum << endl;
}
```

```
void* summation(void* param)
{
  // Delay for 5 seconds
  sleep(5);

  // Convert (void *) parameter to (int *) parameter
  // then get the integer.
  int theNum = *((int *) param);
  cout << "Computing summation of " << theNum << endl;

  // Compute summation
  sum = 0;
  for (int i=1; i<=theNum; i++) {
    sum += i;
  }

  // Thread exit
  pthread_exit(0);
}
```