Threads

COMP 3361: Operating Systems I
Winter 2015
http://www.cs.du.edu/3361

1 Single Thread of Control

```
...
int numbers[1000000];
long even_sum;
long odd_sum;
...
even_sum = add_even(numbers,1000000);
odd_sum = add_odds(numbers, 1000000);
...

long add_evens(int *numbers, int size) {
      int i;
      long sum = 0;
      for (i=0; i<size; i+=2) {
            sum += numbers[i];
      }
      return sum;
}

long add_odds(int *numbers, int size) {
      int i;
      long sum = 0;
      for (i=1; i<size; i+=2) {
            sum += numbers[i];
      }
      return sum;
}
```

control flow (a single thread of control)

Two Threads of Control

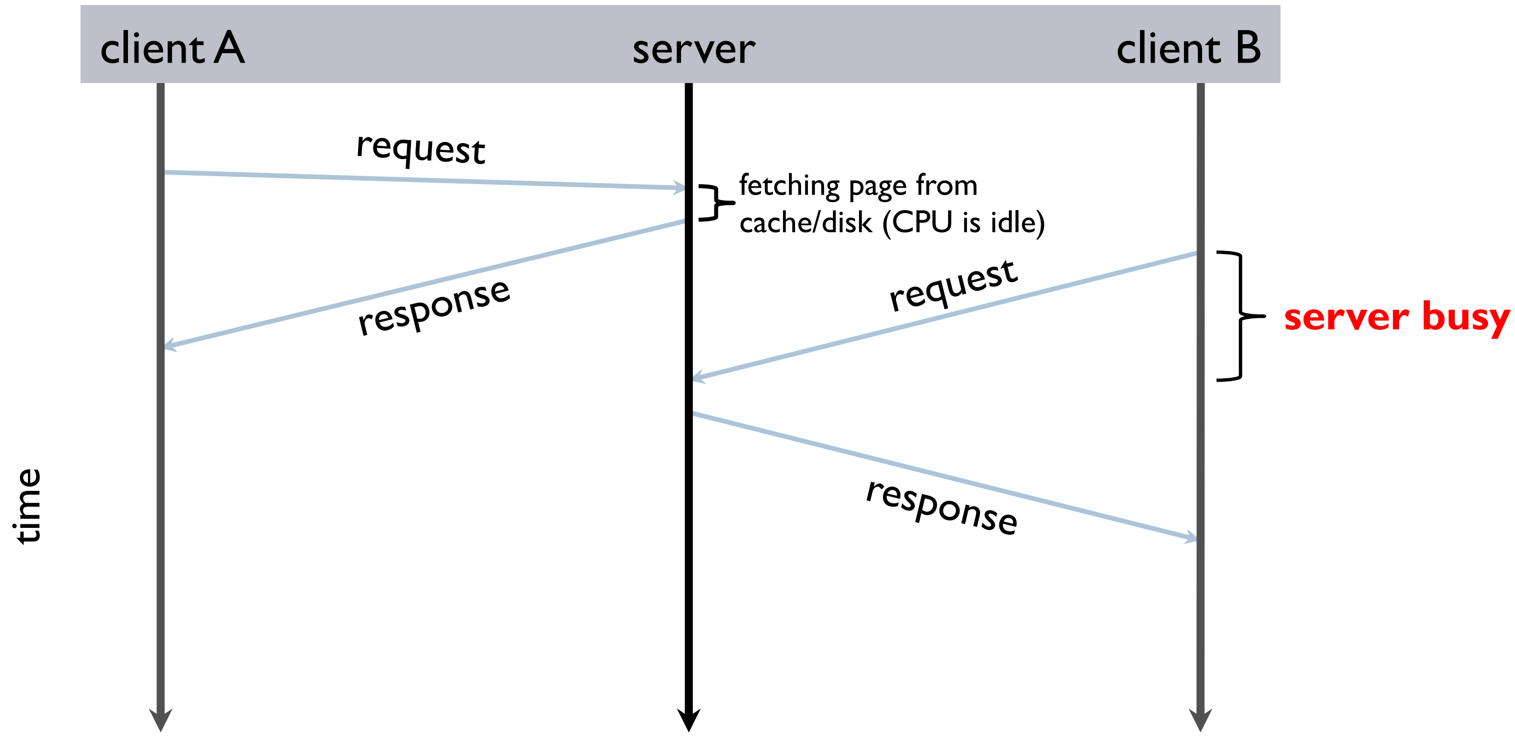
```
...
int numbers[1000000];
long even_sum;
long odd_sum;
...
...
...

long add_evens(int *numbers, int size) {
      int i;
      long sum = 0;
      for (i=0; i<size; i+=2) {
            sum += numbers[i];
      }
      return sum;
}

long add_odds(int *numbers, int size) {
      int i;
      long sum = 0;
      for (i=1; i<size; i+=2) {
            sum += numbers[i];
      }
      return sum;
}
```

two threads of control

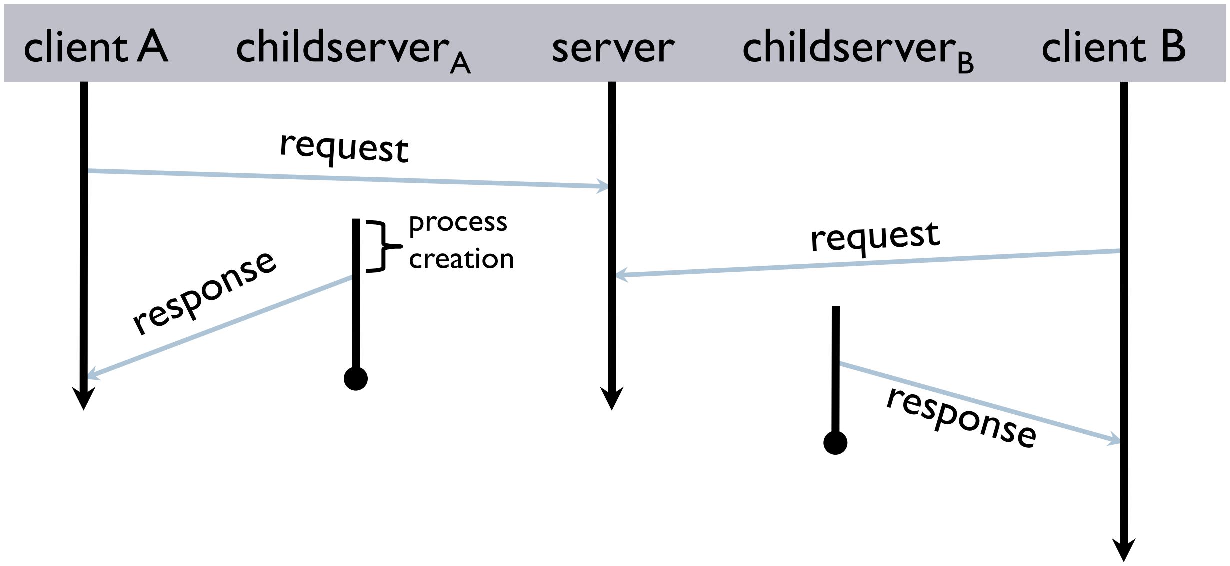
Threads

- A process by default has a single flow of control
  - a single **thread** of control
- A task can be parallelized by spawning multiple processes
  - the processes communicate with help from the kernel
- Another method is to have multiple flows of control in a process
  - each flow of control is a **thread**

Web Server with Single Thread

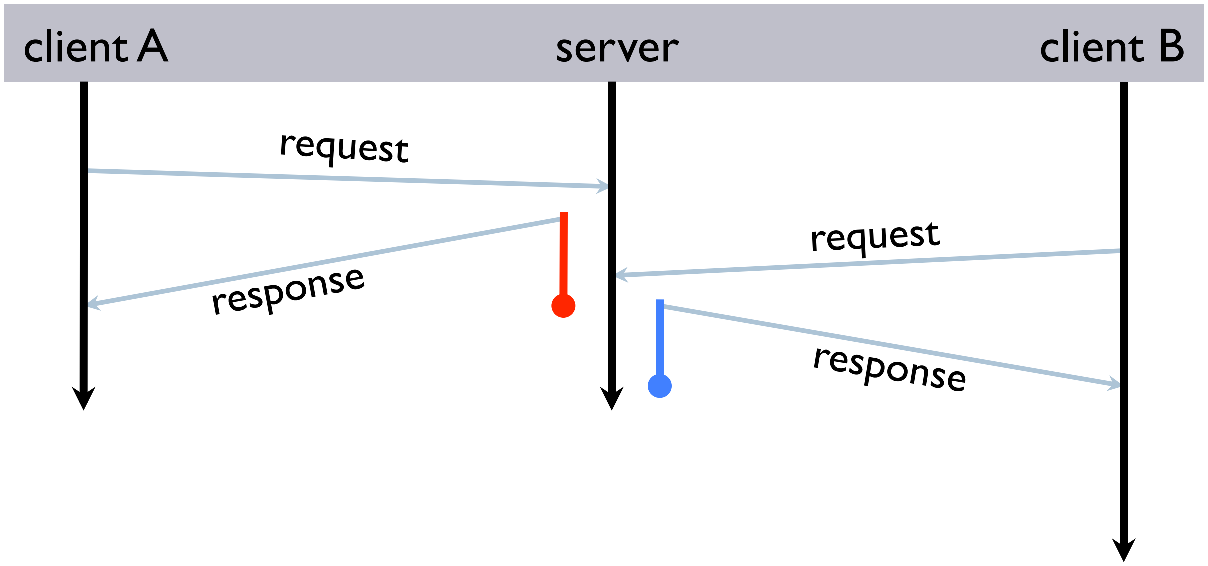


5 Web Server with Multiple Processes



Slow; process creation and deletion have high overhead

6 Web Server with Multiple Threads

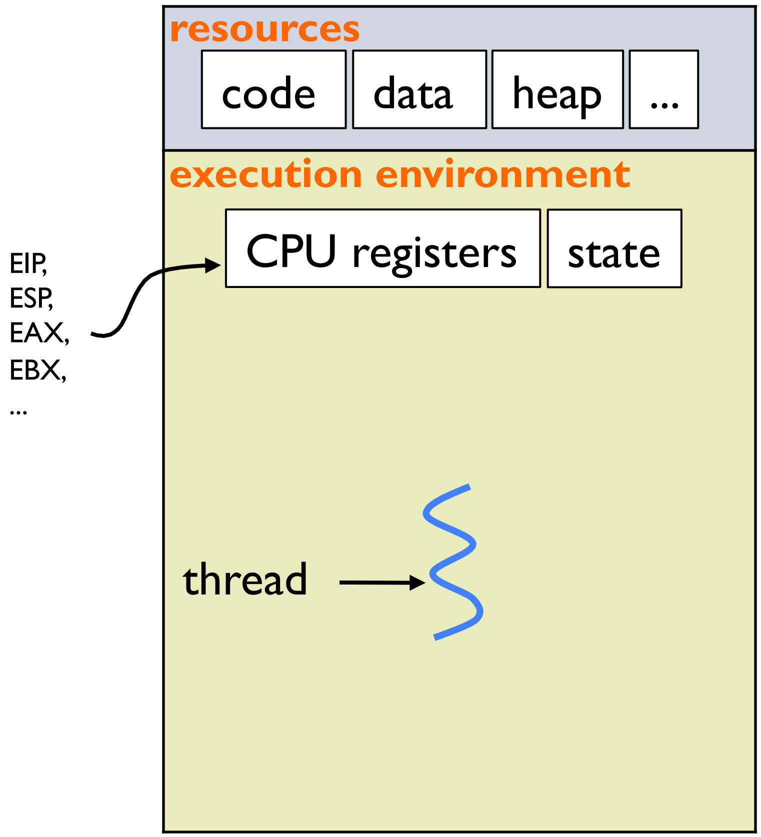


7

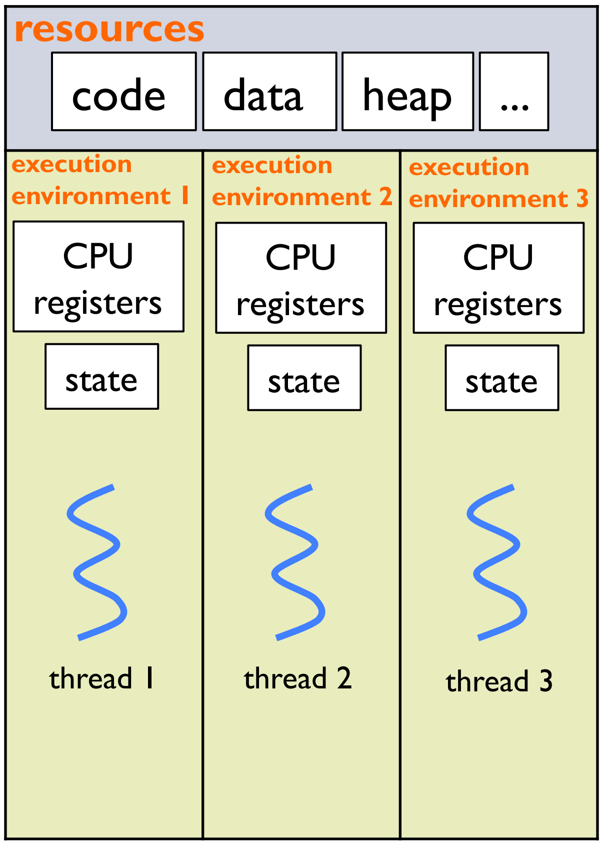
Why Use Threads?

- Multiprogramming
  - but shared address space (one thread can access data of another)
- Lighter weight than processes
  - threads carry less state information than processes
  - threads are sometimes called **lightweight processes**
- Performance
  - overlap CPU bound and I/O bound tasks of a process
- Scalability
  - multithreaded processes can occupy multiple CPUs

Single-Threaded vs. Multithreaded

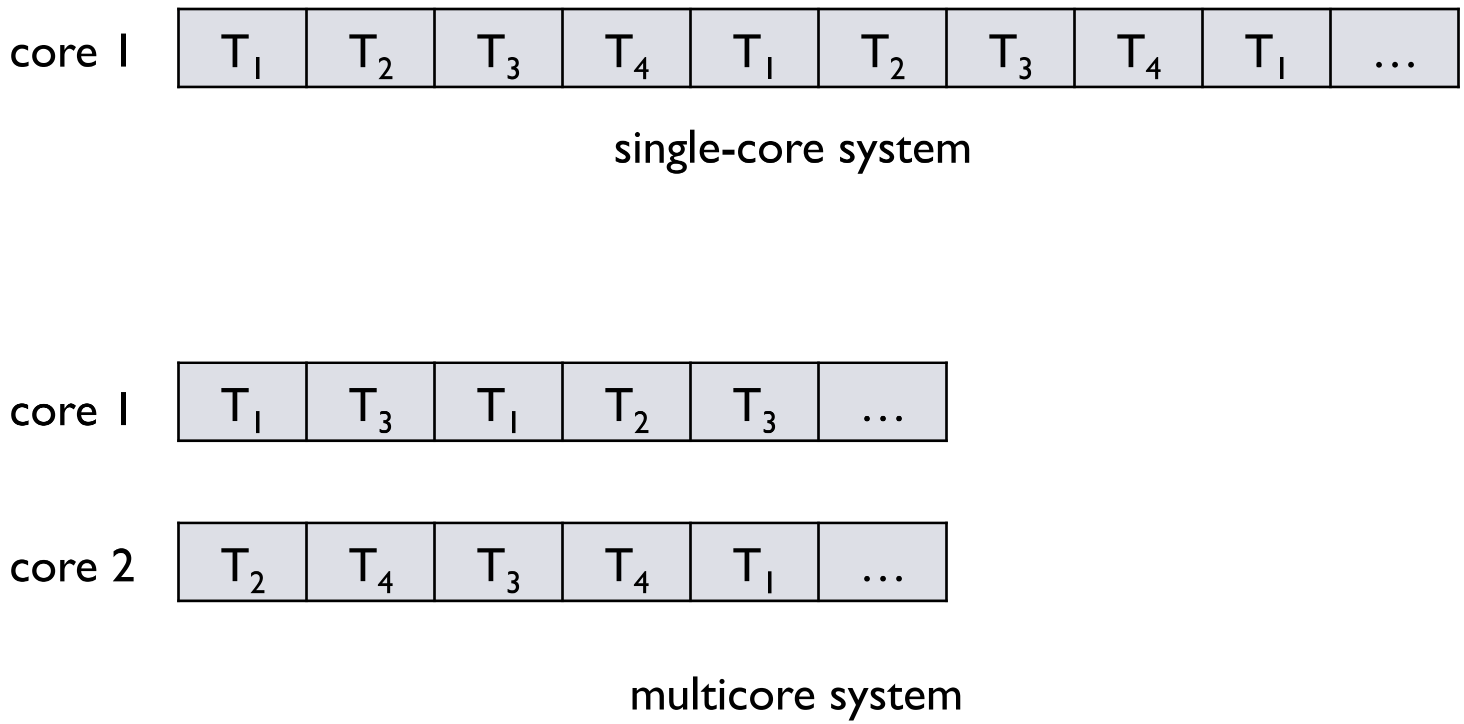


single-threaded process



multithreaded process

Concurrent Execution of Threads



POSIX Pthreads

- Only a standard that defines an API for thread creation and management

- Native POSIX Thread Library (NPTL) is an implementation of the specification in most Linux systems

11 Some Common Functions

- **`pthread_create`** : create a thread
- **`pthread_join`** : wait for a thread to finish
- **`pthread_cancel`** : terminate another thread
- **`pthread_yield`** : relinquish CPU
- **`pthread_exit`** : exit the thread (same as return)

Example

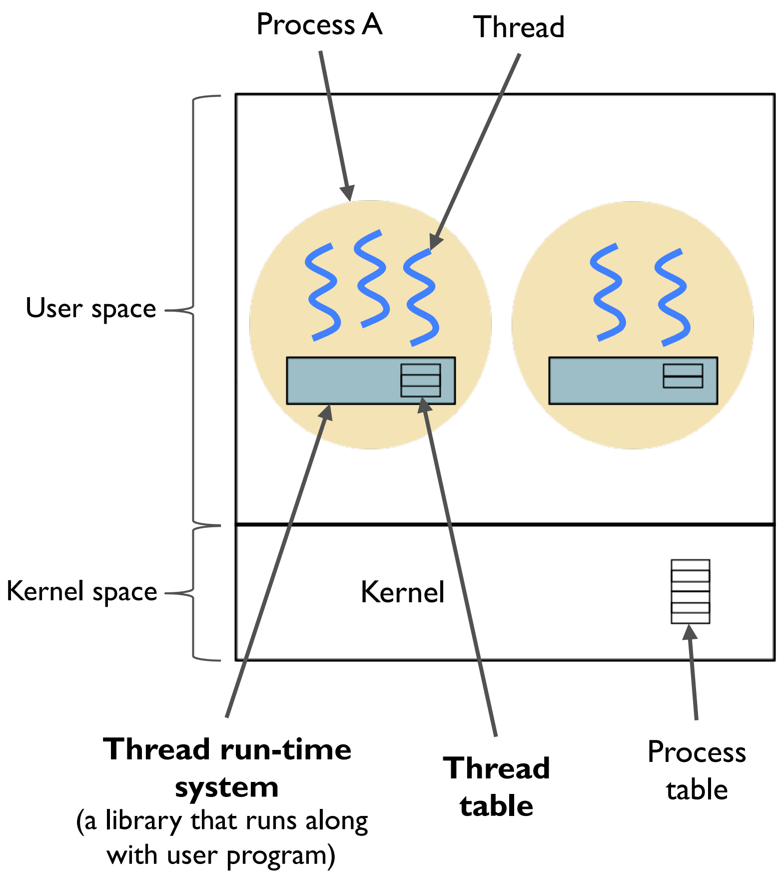
```
#include <pthread.h>
#include <stdio.h>

void *add_evens(void *data);
void *add_odds(void *data);

/* argument to pass to threads */
typedef struct {
     int *numbers;
     int size;
} TD;

int main() {
     int numbers[10] = {1,2,3,4,5,6,7,8,9,10};
     long odd_sum, even_sum;

     pthread_t tid1, tid2; /* thread identifiers */
     TD r;
     r.numbers = numbers;
     r.size = 10;

     /* create thread */
     pthread_create(&tid1, NULL, add_evens, &r);
     pthread_create(&tid2, NULL, add_odds, &r);
```

(continued on next slide)

13 Example (Contd...)

```
    /* wait for thread */
    pthread_join(tid1, (void *)&even_sum);
    pthread_join(tid2, (void *)&odd_sum);

    printf("sum = %ld\n",even_sum+odd_sum);
}

void *add_odds(void *arg) {
    int i;
    long sum = 0;
    TD *r = (TD *)arg;

    for (i=1; i<r->size; i+=2) sum += r->numbers[i];
    return (void *)sum;
}

void *add_evens(void *arg) {
    int i;
    long sum = 0;
    TD *r = (TD *)arg;

    for (i=0; i<r->size; i+=2) sum += r->numbers[i];
    return (void *)sum;
}
```

14 Implementing Threads in User Space

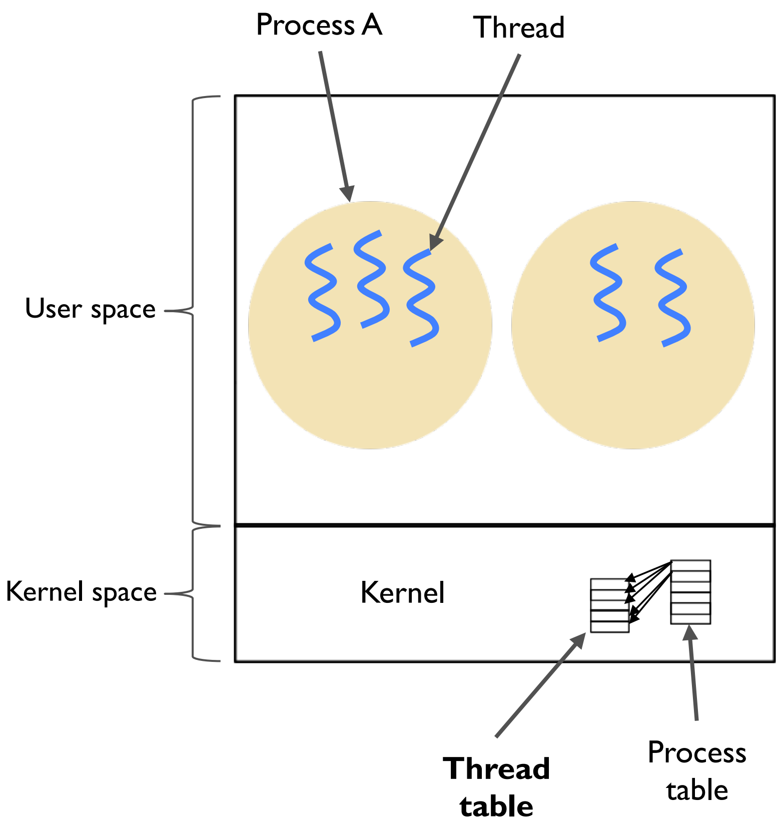


- When process A runs, code in one of the threads run
- When the running thread makes a thread-specific function call, run-time system gets control
- Run-time system can switch to another thread (fast because no system call)
  - saves CPU state in thread table
  - picks another ready thread for running
  - loads CPU with the state of chosen thread

These steps happen during the time the CPU is allocated to process A.

What happens if process A is in blocked state?

15 Implementing Threads in Kernel Space



- When process A runs, code in one of the threads run
- When the running thread makes a system call, scheduler in the kernel gets control
- Scheduler can switch to another thread of same process, or to a different process

16

Scheduler Activations

- A scheme for communication between user-thread library and the kernel
  - kernel provides virtual processors to run-time system
    - virtual processors: kernel threads that the OS allocates the CPU to
  - user-level threads are scheduled onto an available virtual processor by the run-time system
  - a blocked thread on a virtual processor is wastage of allocated resources
- Scheduler activations provide **upcalls**
  - a way of notifying the thread runtime system about *interesting* activities in the threads
    - e.g. a thread has blocked/unblocked

Threading Issues

- Need thread-wide global variables
  - variables seen by any procedure in a thread, but not in another thread
- Many library procedures are not re-entrant
  - when interrupted and then resumed, return value of procedure will be unreliable if another thread called the procedure in the meantime
- Signal handling
  - **signals** are notifications from the kernel about interesting events (e.g. CTRL+C is pressed)
  - process registers signal handler with OS
- Stack management
  - how will the kernel manage thread stacks?

References

- Chapter 2.2, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.