CMSC421: Principles of Operating Systems

Nilanjan Banerjee

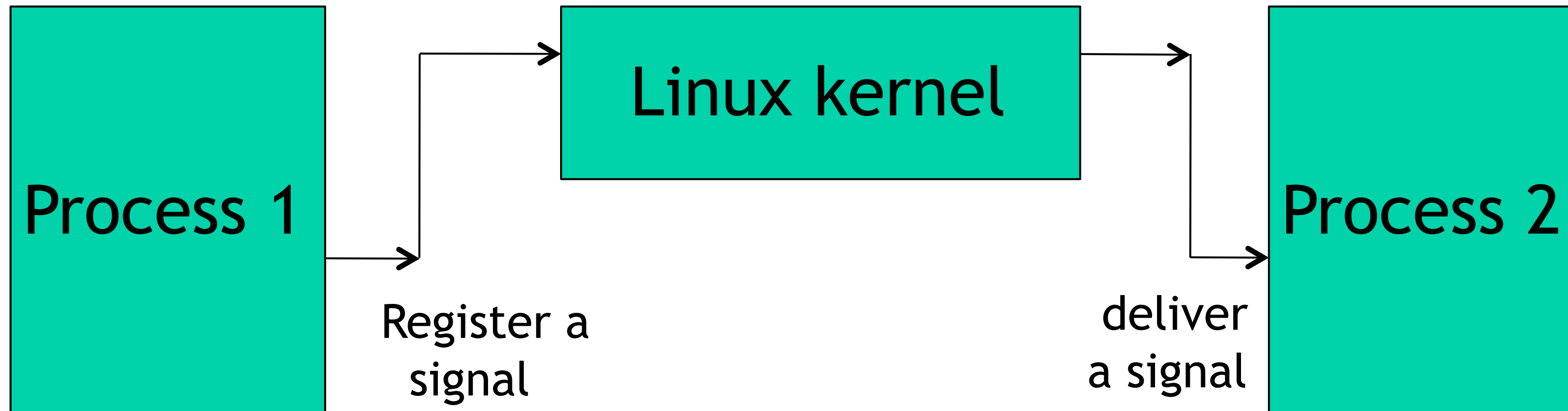
Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

Acknowledgments: Some of the slides are adapted from Prof. Mark Corner and Prof. Emery Berger's OS course at Umass Amherst

Announcements

- Readings from Silberchatz [4th chapter]
- Project 1 is out
- Homework 2 would be out end of this week
- Homework 1 grades are out

POSIX Signals



Name	*Description*	*Default Action*
`SIGINT`	Interrupt character typed	terminate process
`SIGQUIT`	Quit character typed (`^\`)	create core image
`SIGKILL`	`kill -9`	terminate process
`SIGSEGV`	Invalid memory reference	create core image
`SIGPIPE`	Write on pipe but no reader	terminate process
`SIGALRM`	`alarm()` clock 'rings'	terminate process
`SIGUSR1`	user-defined signal type	terminate process
`SIGUSR2`	user-defined signal type	terminate process

signals

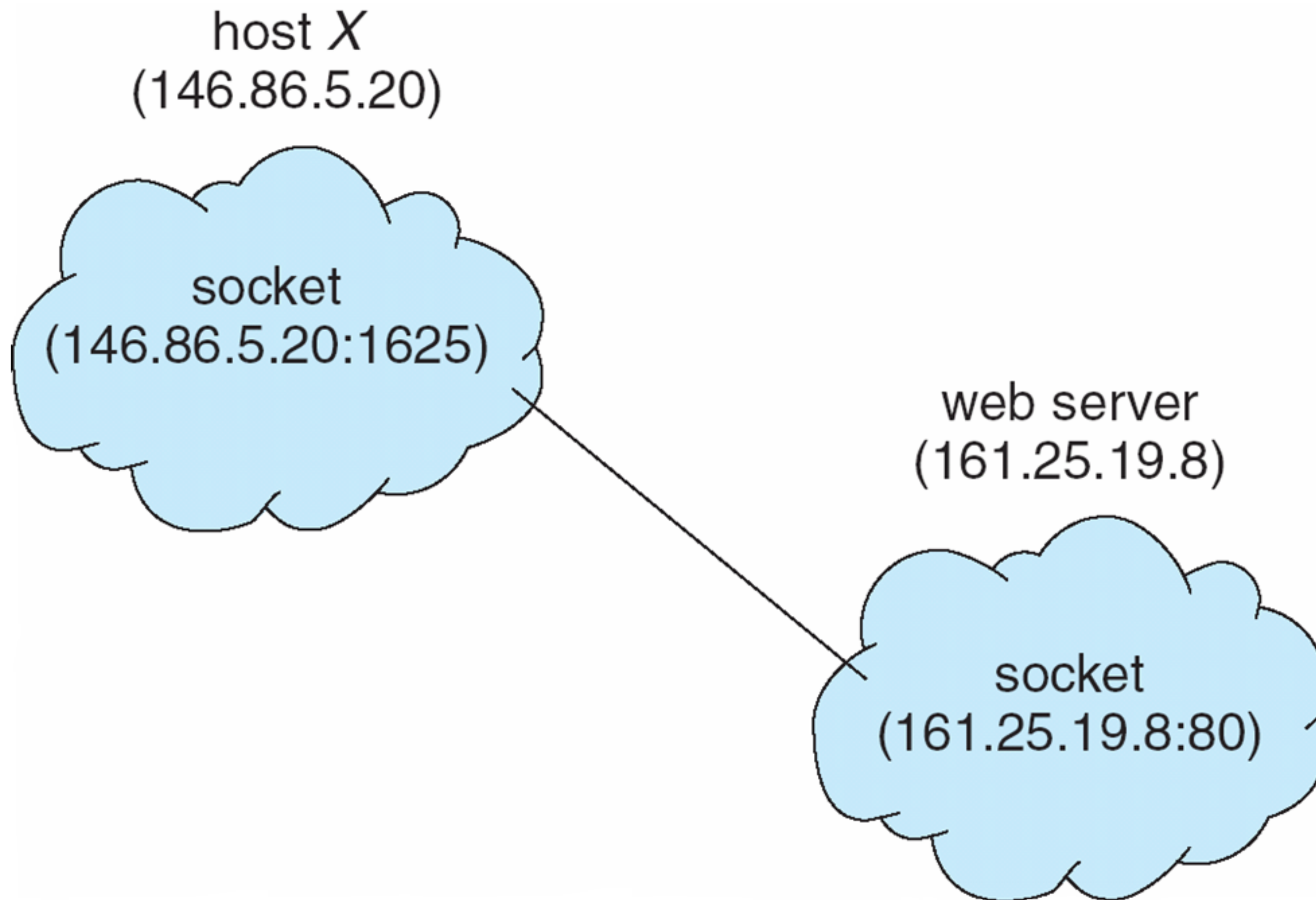
- `int kill( pid_t pid, int signo );`
  - Send a signal to a process with a process id

- `signal(<signal name>, <pointer to handler>)`
  - Handle a maskable signal in your code

Message Passing Using Sockets

- A **socket** is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets

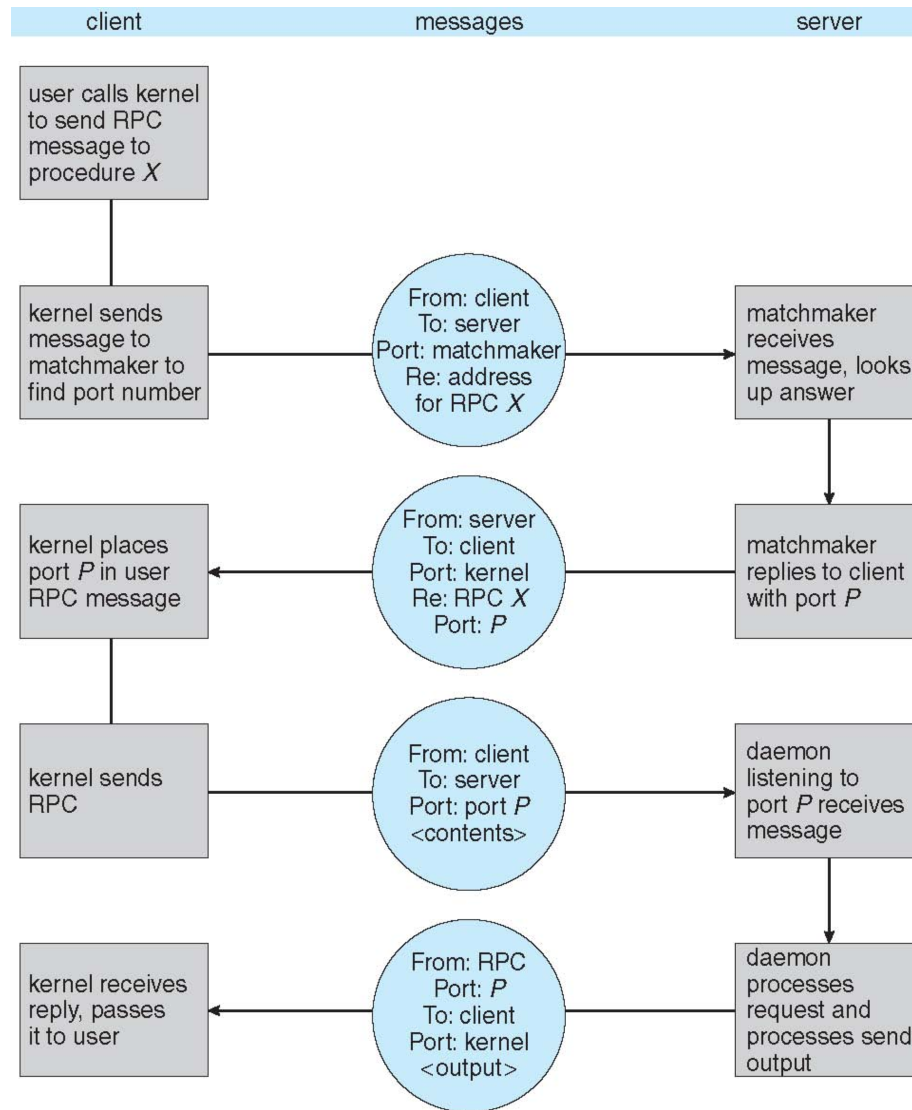
Message Passing Using Sockets



Concept of Remote Procedure calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and *marshalls* the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server

Execution of RPC

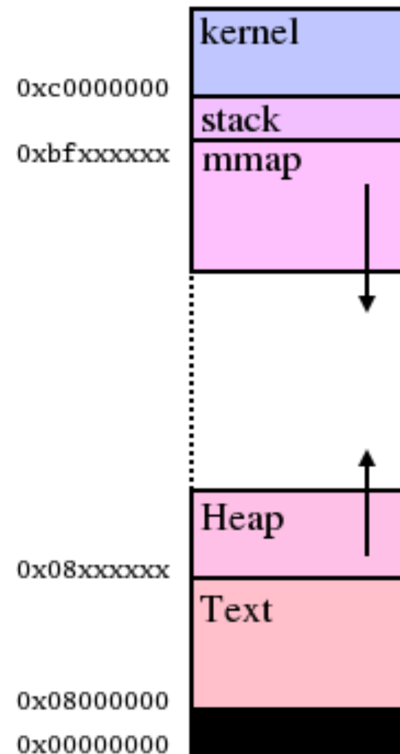


A Step back: Definition of a Process

- One or more threads running in an address space

What is an address space?

- A collection of data and code for the program organized in an memory (addressable) region



Why do we need processes?

- A process (with a single thread) supports a serial flow of execution

- Is that good enough for all purposes?
- What if something goes wrong in one process?
- What if something takes a long amount of time in one process?
- What if we have more than one user?

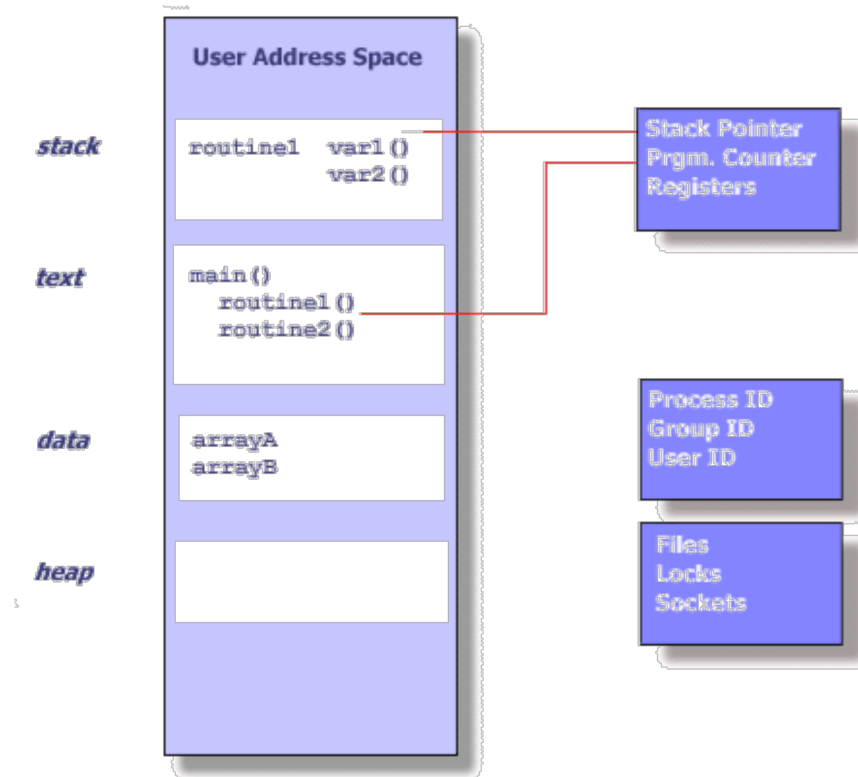
Processes Vs Threads

- Both abstractions are very important
  - They can provide parallel programming & concurrency
  - They can hide latency from the end user
  - Maximize CPU utilization
  - Handle multiple, asynchronous events

- But they support different programming styles, have different performances

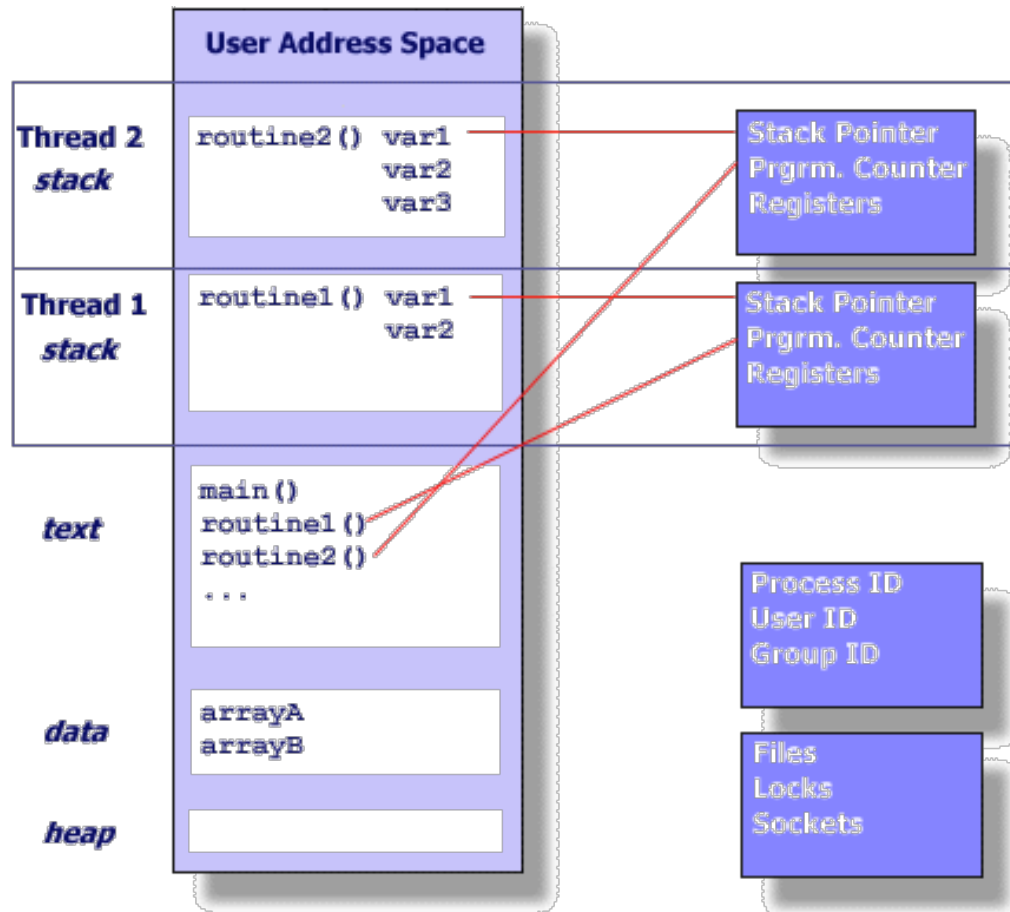
What is difference between Process and Threads

- The execution context of the process are (Program Counter, registers), address space, files, mmaped regions etc.



What is difference between Process and Threads

- Threads share an address space. They have same files, sockets etc.
- They have their own stack, program counters, registers, and stack specific data



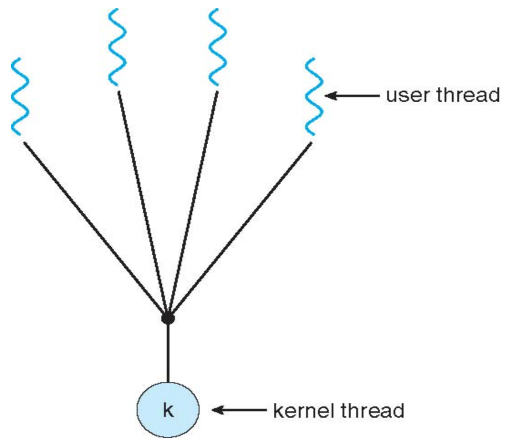
Creating Threads

- UNIX
  - Pthreads (POSIX threads)
  - Pthread_create() --- creating a thread
  - Pthread_join() --- wait for thread to finish

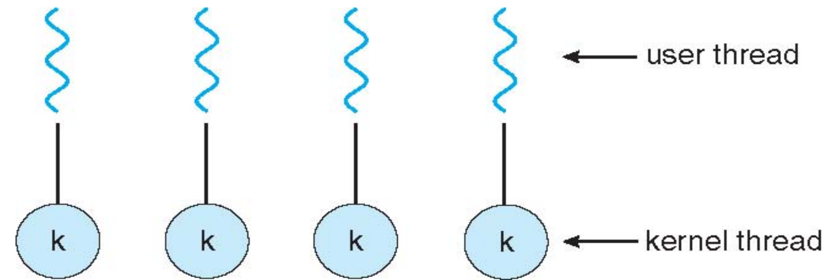
Lets see a demonstration of using pthreads.

Scheduling

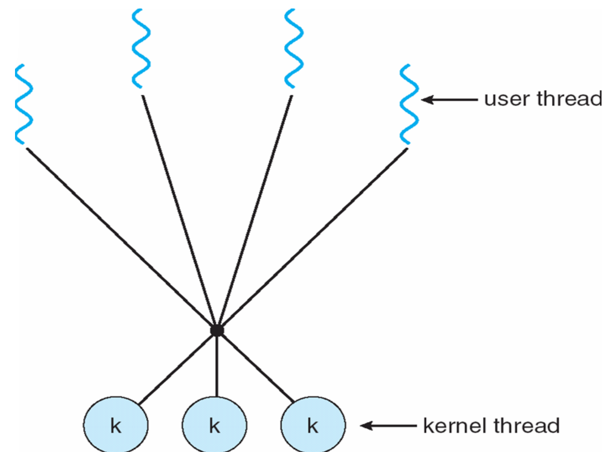
Many-one (Green Threads)



One-one (linux)



Many-many (Window-NT)



When to use Threads and When to use processes

- Processes or Threads
  - Performance?
  - Flexibility/Ease-of-use
  - Robustness

- Simple Scheduling
  - OS has a list of Threads and schedules them similar to Processes. In Linux, threads/processes treated similarly
  - Chooses one of the threads and loads them to be run
  - Runs them for a while, runs another.

How does it work in Linux

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through `clone()` system call
- `clone()` allows a child task to share the address space of the parent task (process)
- `struct task_struct` points to process data structures (shared or unique)

Threads Vs Processes

- Processes or Threads
  - Performance?
  - Flexibility/Ease-of-use
  - Robustness

- Simple Scheduling
  - OS has a list of Threads and schedules them similar to Processes. In Linux, threads/processes treated similarly
  - Chooses one of the threads and loads them to be run
  - Runs them for a while, runs another.

Flexibility/Ease of use?

- Process are more flexible
  - Easy to spawn processes, I can ssh into a server and spawn a process
  - Can communicate over sockets= distributes across machines

- Threads
  - Communicate using memory – must be on the same machine
  - Requires thread-safe code

Robustness

- Process are more robust
  - Processes are separate or sandboxed from other processes
  - If one process dies, no effect on other processes

- Threads
  - If one thread crashes, whole process terminates
  - Since there is sharing, there are problems with using the stack efficiently

An in-class discussion