Network Programming

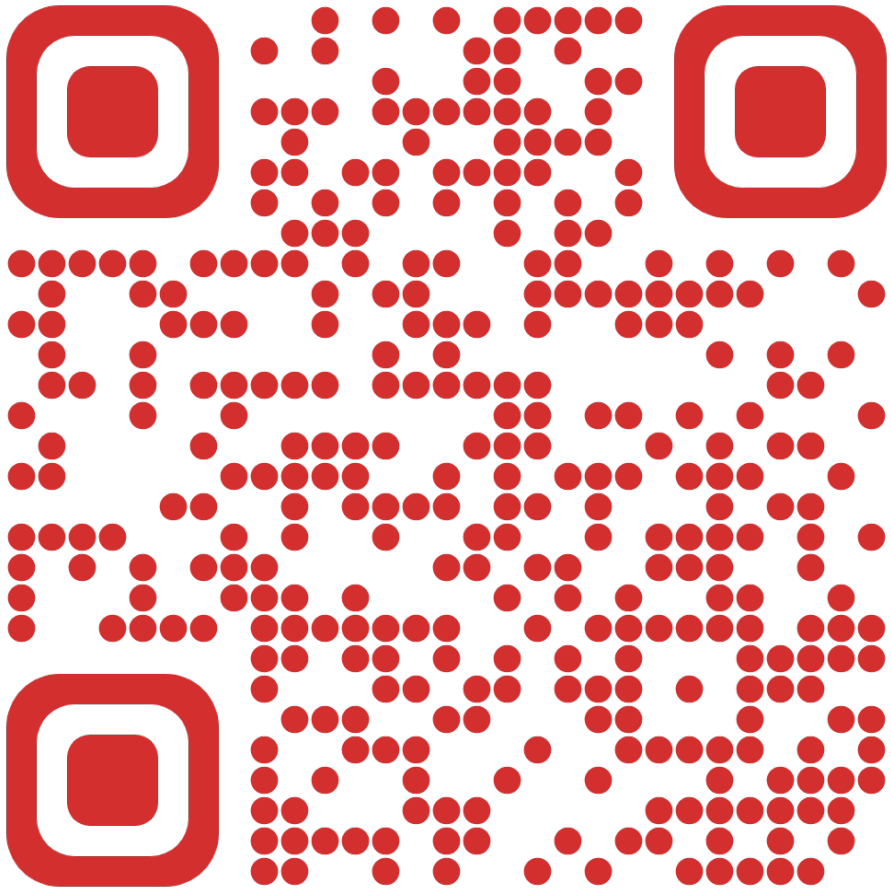
Preliminaries

Rathindra Nath Dutta

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata

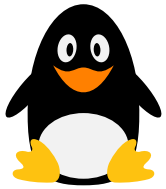
October 9, 2023

https://www.isical.ac.in/~rathin_r/uploads/CN/



WEB PAGE

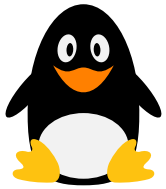
Communication via Protocol Stack



Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Communication via Protocol Stack

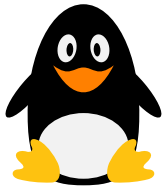


"Rathin"

Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

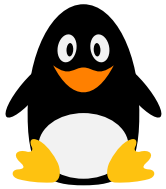
Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Communication via Protocol Stack

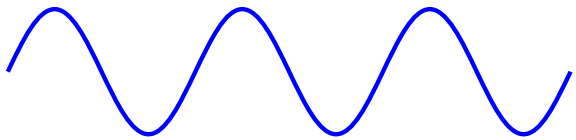


Application Layer	Application Layer
Transport Layer	Transport Layer
Network Layer	Network Layer
Data Link Layer	Data Link Layer
Physical Layer	Physical Layer

Communication via Protocol Stack

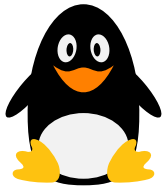


Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

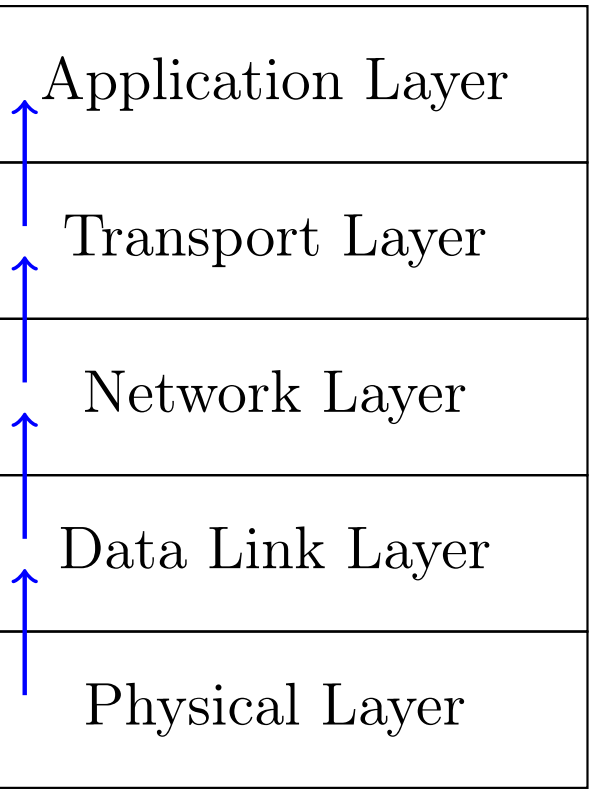


Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

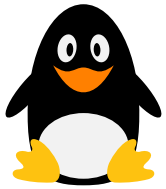
Communication via Protocol Stack



Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer



Communication via Protocol Stack



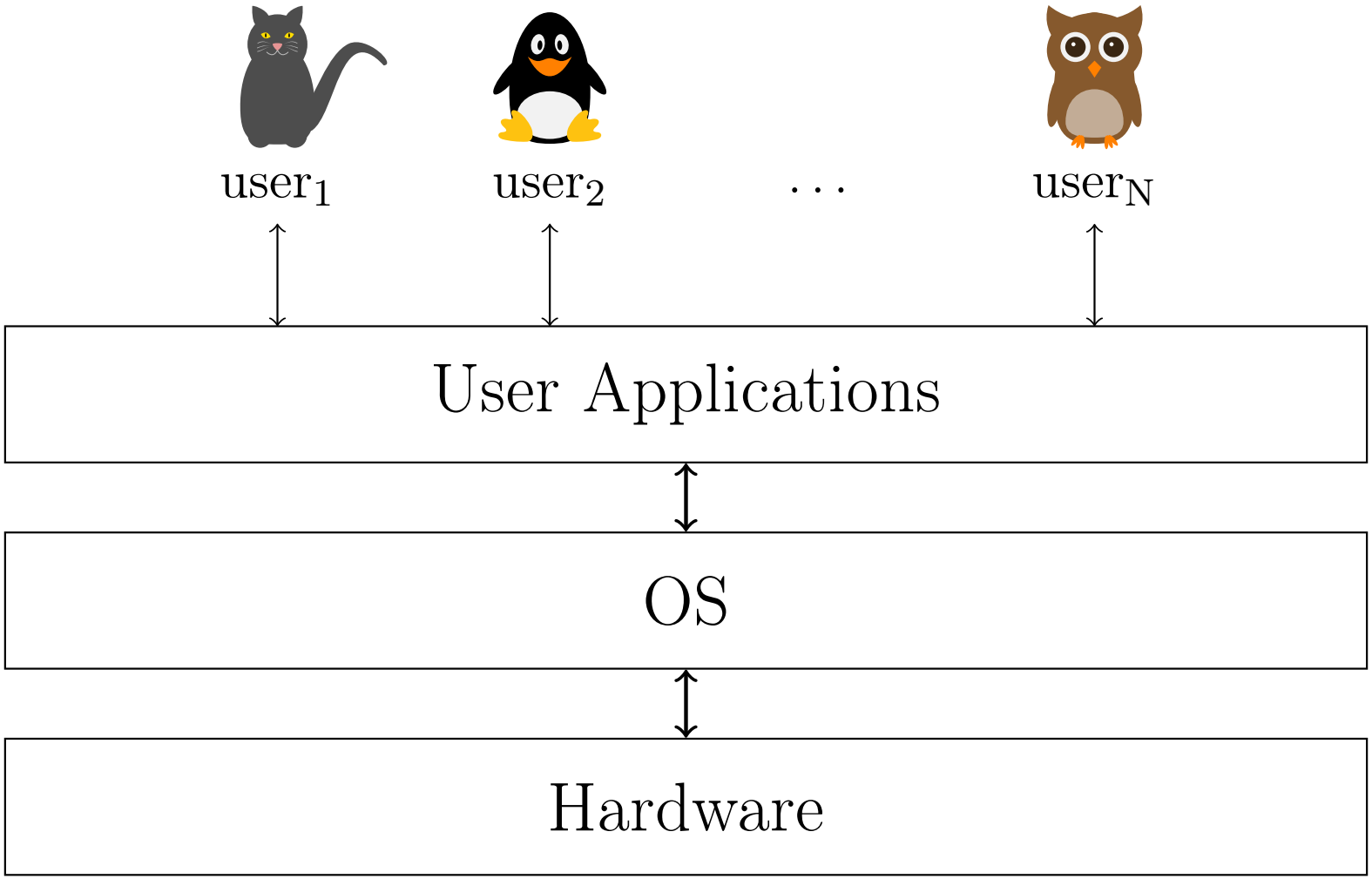
Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

"Rathin"

Application Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

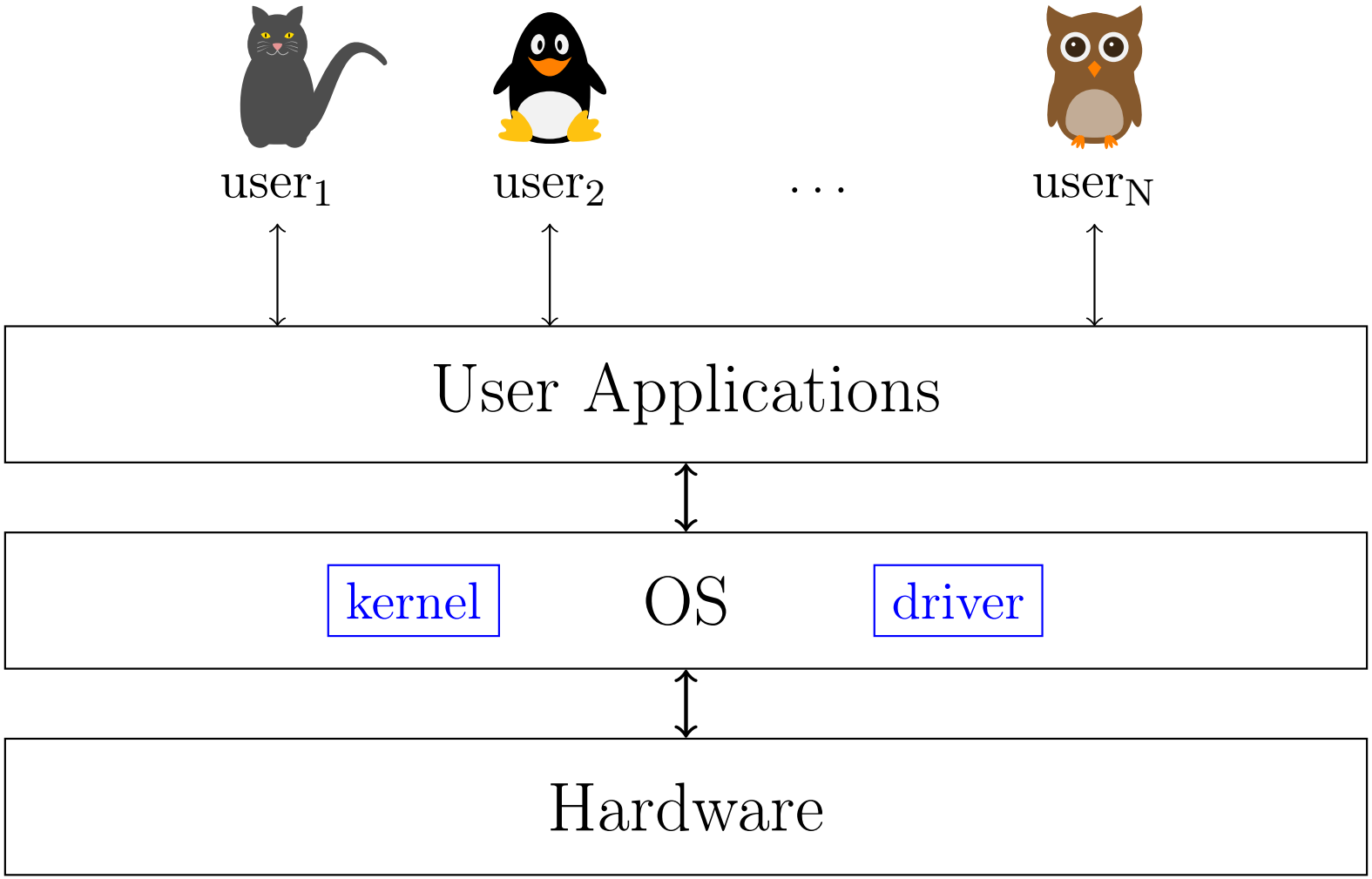
Role of Operating System

The Operating System (OS) acts as an interface between the user application software and the underlying hardware



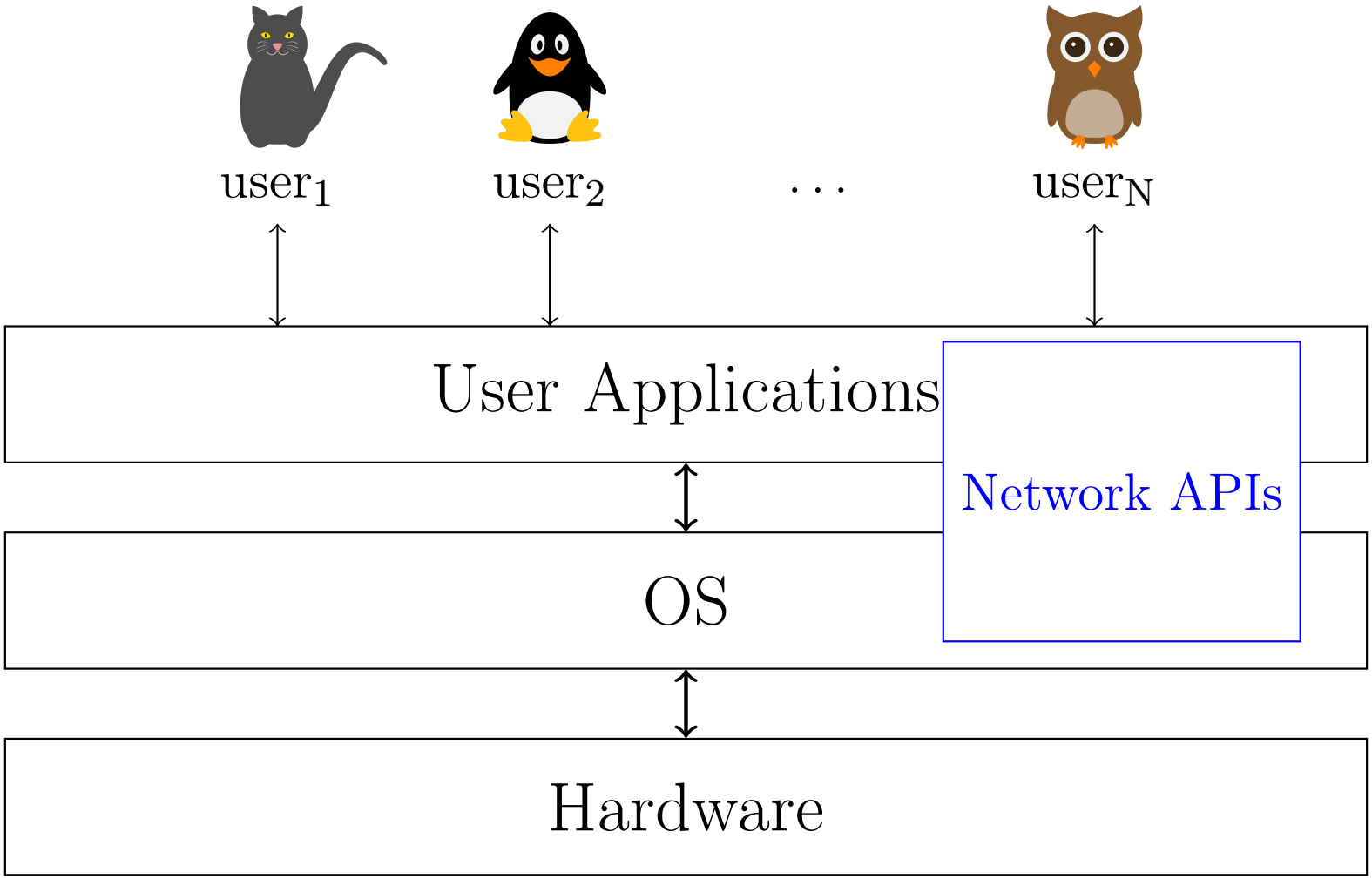
Role of Operating System

The Operating System (OS) acts as an interface between the user application software and the underlying hardware



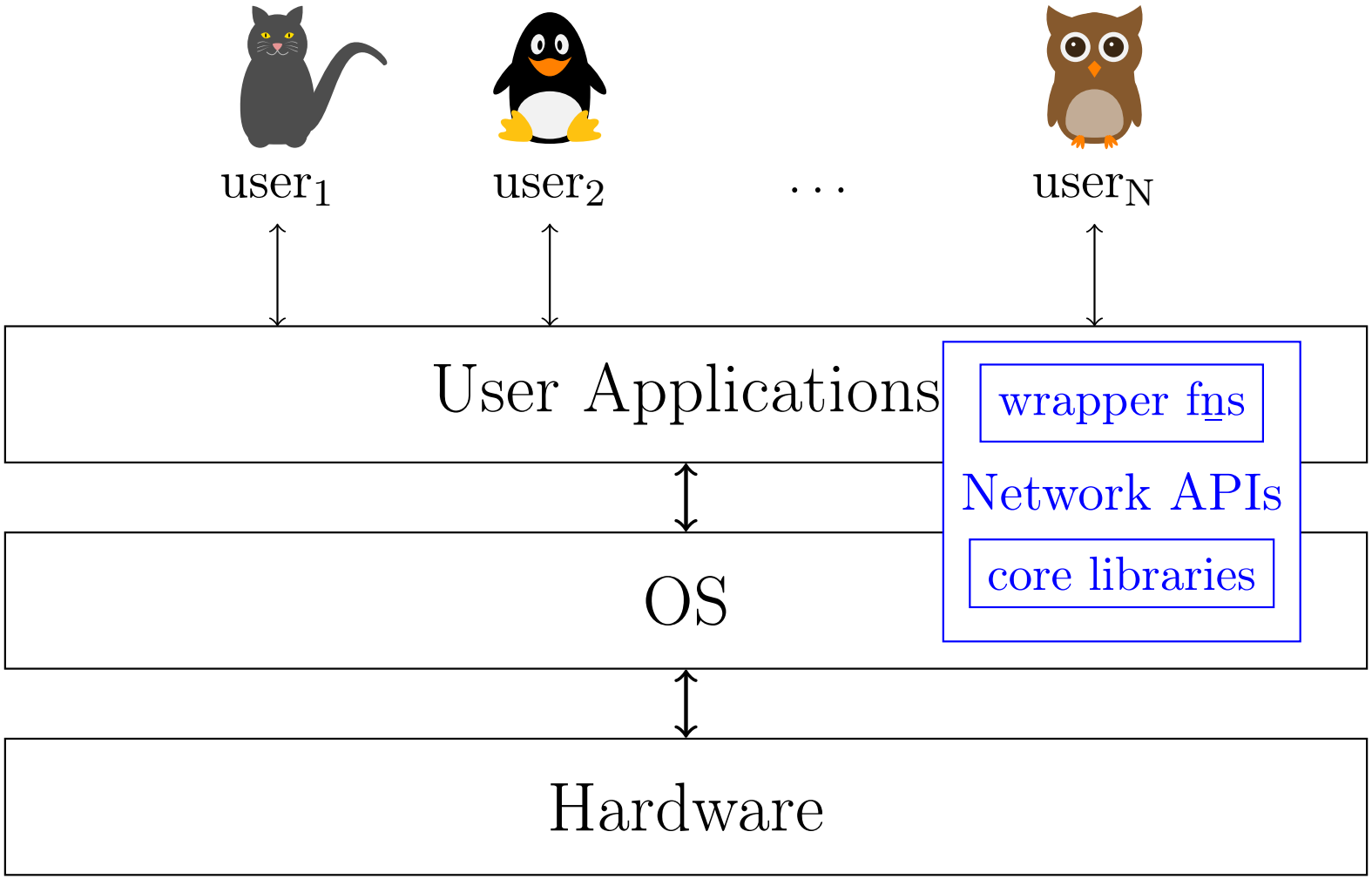
Role of Operating System

The Operating System (OS) acts as an interface between the user application software and the underlying hardware



Role of Operating System

The Operating System (OS) acts as an interface between the user application software and the underlying hardware



Agenda

- Using Sockets for the communications
- Using Remote Procedure Calls

What is a Socket?

- "A network **socket** is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network . . . Sockets are created only during the lifetime of a process of an application running in the node" – WIKI[1]

- "A **socket** is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes" – IBM[2]

- "A **socket** is one endpoint of a two-way communication link between two programs running on the network" – ORACLE[3]

[1] https://en.wikipedia.org/wiki/Network_socket

[2] https://www.ibm.com/docs/en/i/7.5?topic=communications-socket-programming

[3] https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

How are Sockets uniquely identified?

- A *Socket* is identified to other hosts by its **Socket Address**

How are Sockets uniquely identified?

- A *Socket* is identified to other hosts by its **Socket Address**
- A *Socket Address* consists of a **Transport Protocol** (TCP, UDP, etc.), an **IP Address** (IPv4 or IPv6) and a **Port Number**

How are Sockets uniquely identified?

- A *Socket* is identified to other hosts by its **Socket Address**
- A *Socket Address* consists of a **Transport Protocol** (TCP, UDP, etc.), an **IP Address** (IPv4 or IPv6) and a **Port Number**
- A *Port Number* is a (unique) 16-bit unsigned integer assigned to one endpoint

How are Sockets uniquely identified?

- A *Socket* is identified to other hosts by its **Socket Address**
- A *Socket Address* consists of a **Transport Protocol** (TCP, UDP, etc.), an **IP Address** (IPv4 or IPv6) and a **Port Number**
- A *Port Number* is a (unique) 16-bit unsigned integer assigned to one endpoint
- ports 0 through 1023 are well-known ports (aka system ports)
- ports 1024 through 49151 are registered ports[4]
- ports from 49152 through 65535 are dynamic or private ports; commonly known as ephemeral ports[5]

[4] IANA maintains the official list https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

[5] ephemeral (adj.): lasting for a very short time or having a very short life cycle

Why another address?

TCP/IP Layer

Application
Transport
Network
Data-Link Physical

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols
Application	TELNET, HTTP, DHCP, PING, FTP, ...
Transport	TCP, UDP, ...
Network	IP, ARP, TCMP, ...
Data-Link Physical	Ethernet, WiFi, ...

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message
Transport	TCP, UDP, ...	Segment/Datagram
Network	IP, ARP, TCMP, ...	Datagram
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet	Address
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message	Application Specific
Transport	TCP, UDP, ...	Segment/Datagram	Port
Network	IP, ARP, TCMP, ...	Datagram	Logical (IP)
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits	Physical(MAC)

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet	Address	Objective
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message	Application Specific	
Transport	TCP, UDP, ...	Segment/Datagram	Port	
Network	IP, ARP, TCMP, ...	Datagram	Logical (IP)	
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits	Physical(MAC)	identification

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet	Address	Objective
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message	Application Specific	
Transport	TCP, UDP, ...	Segment/Datagram	Port	
Network	IP, ARP, TCMP, ...	Datagram	Logical (IP)	logical organization[1]
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits	Physical(MAC)	identification

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet	Address	Objective
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message	Application Specific	
Transport	TCP, UDP, ...	Segment/Datagram	Port	host-to-host delivery[2]
Network	IP, ARP, TCMP, ...	Datagram	Logical (IP)	logical organization[1]
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits	Physical(MAC)	identification

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Why another address?

TCP/IP Layer	Common Protocols	Data Packet	Address	Objective
Application	TELNET, HTTP, DHCP, PING, FTP, ...	Message	Application Specific	end-to-end delivery[3]
Transport	TCP, UDP, ...	Segment/Datagram	Port	host-to-host delivery[2]
Network	IP, ARP, TCMP, ...	Datagram	Logical (IP)	logical organization[1]
Data-Link Physical	Ethernet, WiFi, ...	Frame Bits	Physical(MAC)	identification

[3] session control, a process(browser) may have multiple active session(tabs) to same client(google search), uses application specific URIs

[2] process-to-process, a node can run multiple processes each talking via different protocol

[1] better management, efficient routing

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6]

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6] – sockets are no exception

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6] – sockets are no exception
- We need a good understanding of C file descriptors and stream I/O

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6] – sockets are no exception
- We need a good understanding of C file descriptors and stream I/O
- Most programs/tools takes arguments as input

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6] – sockets are no exception
- We need a good understanding of C file descriptors and stream I/O
- Most programs/tools takes arguments as input
  e.g. `gcc -g -Wall filename.c -o prog`

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Socket Programming Using C

- Unix philosophy: "Everything is a file" [6] – sockets are no exception
- We need a good understanding of C file descriptors and stream I/O
- Most programs/tools takes arguments as input
  e.g. `gcc -g -Wall filename.c -o prog`
- We also need a little understanding of command-line arguments

[6] https://en.wikipedia.org/wiki/Everything_is_a_file

Command-line Argument Examples

- Print all arguments

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ⏎

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ↵
  Output:

```
Number of arguments passed:  6
The executable name is:  ./a.out
Given arguments are:  i, am, rathin, this, is, 2023
```

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ⏎
  Output:

```
Number of arguments passed:  6
The executable name is:  ./a.out
Given arguments are:  i, am, rathin, this, is, 2023
```

  See: print_args.c

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ↵
  Output:
  ```
  Number of arguments passed:  6
  The executable name is:  ./a.out
  Given arguments are:  i, am, rathin, this, is, 2023
  ```
  See: print_args.c

- Write an integer add program

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ↵
  Output:
  ```
  Number of arguments passed:  6
  The executable name is:  ./a.out
  Given arguments are:  i, am, rathin, this, is, 2023
  ```
  See: print_args.c

- Write an integer add program
  Example Usage:
  `./add 1 10 3 -5 8` ↵ output: 17
  `./add` ↵ output: 0

Command-line Argument Examples

- Print all arguments
  Example execution: `./a.out i am rathin this is 2023` ↵
  Output:
  ```
  Number of arguments passed:  6
  The executable name is:  ./a.out
  Given arguments are:  i, am, rathin, this, is, 2023
  ```
  See: print_args.c

- Write an integer add program
  Example Usage:
  `./add 1 10 3 -5 8` ↵ output: 17
  `./add` ↵ output: 0
  See: add_args.c

File Descriptor

- Unix treats (almost) everything as a file
  plain file, directory, streams, sockets etc.

File Descriptor

- Unix treats (almost) everything as a file
  plain file, directory, streams, sockets etc.

- A *file descriptor* (fd) is a small, nonnegative integer that refers to the index of an opened file by the current process

File Descriptor

- Unix treats (almost) everything as a file
  plain file, directory, streams, sockets etc.

- A *file descriptor* (fd) is a small, nonnegative integer that refers to the index of an opened file by the current process

- By default a process has three open files:

fd value	file stream
0	stdin (standard input)
1	stdout (standard output)
2	stderr (standard error)

File Descriptor

- Unix treats (almost) everything as a file
  plain file, directory, streams, sockets etc.

- A *file descriptor* (fd) is a small, nonnegative integer that refers to the index of an opened file by the current process

- By default a process has three open files:

fd value	file stream
0	stdin (standard input)
1	stdout (standard output)
2	stderr (standard error)

- Other fds can be created with `open()` system call

The `read()` API

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Attempts to read (up to `count` bytes) from file descriptor `fd` into the buffer starting at `buf`

Return value: On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. On error, -1 is returned

Parameters:
`fd`: a file descriptor to read from
`buf`: pointer to a buffer area (array) to read into
`count`: maximum number of bytes to read

[1] See man page for `read` system call: https://man7.org/linux/man-pages/man2/read.2.html

[2] See for `size_t` and `ssize_t`: https://man7.org/linux/man-pages/man3/size_t.3type.html

The `write()` API

```
#include <unistd.h>
ssize_t write(int fd, void *buf, size_t count);
```

Writes (up to `count` bytes) from buffer starting at `buf` into the file descriptor `fd`

Return value: On success, the number of bytes written is returned. On error, -1 is returned

Parameters:
`fd`: a file descriptor to write into
`buf`: pointer to a buffer area (array) to write from
`count`: maximum number of bytes to write

[1] See man page for `write` system call: https://man7.org/linux/man-pages/man2/write.2.html

Using `read()`/`write()` for I/O

See: system_call_io.c

The open() API

```
#include <fcntl.h>
int open(char *pathname, int flags[, mode_t mode]);
```

Attempts to opens the file specified by `pathname`

Return value: On success, the new file descriptor (a nonnegative integer) is returned. On error, -1 is returned

Parameters:
`pathname`: a file path to open
`flags`: specifies the open mode flag

flag	description
O_RDONLY	open in read only mode
O_WRONLY	open in write only mode
O_RDWR	open in read-write mode
O_CREAT	creates the file if it does not exist
O_TRUNC	truncates if the file exists
O_APPEND	opens in append mode

`mode`: (optional) access mode, useful for creating new files

[1] See man page for `open` system call: https://man7.org/linux/man-pages/man2/open.2.html

[2] See for `mode_t`: https://man7.org/linux/man-pages/man3/mode_t.3type.html

The open() API

```
#include <unistd.h>
int close(int fd);
```

Closes a file descriptor specified by `fd`, so that it no longer refers to any file and may be reused

Return value: returns zero on success. On error, -1 is returned

Parameters:
`fd`: a file descriptor to close

[1] See man page for `close` system call: https://man7.org/linux/man-pages/man2/close.2.html

Using `read()`/`write()` for File I/O

See: file_copy.c

Some Useful Tools

- `ip`

Some Useful Tools

- `ip address show`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

- `ping`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

- `ping <domain name or ip address>`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

- `ping <domain name or ip address>`, e.g.
  `ping google.com`
  `ping localhost` ⟵ same as `ping 127.0.0.1`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

- `ping <domain name or ip address>`, e.g.
  `ping google.com`
  `ping localhost` ⟵ same as `ping 127.0.0.1`
  Mapping of loopback address is usually specified in `/etc/hosts`
  Try: `host loopback`

Some Useful Tools

- `ip address show`
  `ip neighbor`
  `ip route`
  `ip help`

- `ping <domain name or ip address>`, e.g.
  `ping google.com`
  `ping localhost` ⟵ same as `ping 127.0.0.1`
  Mapping of loopback address is usually specified in `/etc/hosts`
  Try: `host loopback`

- Also see `netstat`, `ifconfig`, `nmap`, `nslookup`, `dig` etc.

Using Telnet

Download server1.c
Compile and run the server1 in one terminal; and leave it be

```
gcc server1.c -o server1 && ./server1
```

[1] Telnet is an application protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility - WIKI

Using Telnet

Download server1.c
Compile and run the server1 in one terminal; and leave it be
`gcc server1.c -o server1 && ./server1`

Open another terminal and run a `telnet`[1] client in that
`telnet <server ip> <server port>`
`telnet localhost 54321`

Write anything in telnet
Type "quit" (or send `ctrl+]`) to close the connection

Use `netstat -nlp | grep 54321` to check if the port 54321 is blocked
Then maybe use `kill <pid>` to kill it

[1] Telnet is an application protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility - WIKI

Practice Exercises

- Write a program that accepts a file name as a command-line argument and outputs the number of vowels and consonants in the file on the standard output. You may use `fopen()` and other file I/O library functions, or you may stick to bare `open()` and other system calls for I/O.

- Repeat the above task and use only `read()` and `write()` for all I/O operations (including console outputs).

- Run the Echo Server program (`server1.c`) on your computer. Find the IP address of another computer connected on the same local network as yours. Run `telnet` on that computer and try to connect to the Echo Server running on your computer.