Operating Systems

Dmitry Zaitsev

Lecture 6: Structure of OS kernel. Database of OS. Case study: C program design in Linux.

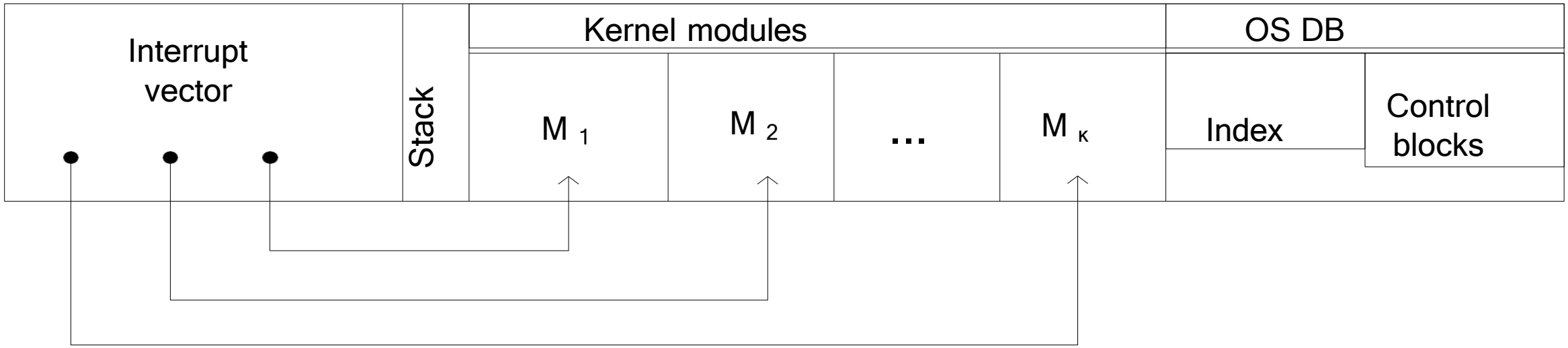
Kernel of OS

- The kernel is the core component of an operating system that acts as the main interface between software applications and the computer's hardware, managing crucial tasks. It controls all system resources, handles system calls from applications, and ensures that multiple programs can run simultaneously and share hardware without conflicts. The kernel is the first program to load on startup and the last to shut down, and if it fails, the entire system will crash.
- The kernel directly manages and controls the system's hardware resources, including the processor (CPU), memory, input/output (I/O) devices, and networking capabilities.

Kernel interaction scheme

A part of OS which is not a process.

A set of interrupt handlers, stack, data base, and pool.



Types of OS kernel

- Monolithic – a large, single kernel with all core operating system services running in a single, shared address space; high performance; low reliability
- Microkernel – a small, deliberately minimal kernel that provides only essential services, with other functions running in user space; higher reliability; reduced performance
- Hybrid – a combination of monolithic and microkernel; balanced performance and reliability
- *Nanokernel for minimal hardware management*
- *Exokernel exposes hardware directly*
- *Multikernel for distributed systems*

Basic entry points of kernel

- **Hardware interrupt, readiness or error of a device**
- Enter the corresponding I/O device driver
- **Software interrupt to implement a system call from a running process (either system or user)**
- Enter to dispatcher of system calls to manage parameters and forward to the corresponding module for the system call implementation
- **Exception/Trap, e.g., invalid instruction or absent page**
- Exception processing, e.g., process termination or swapping of page

Co-program call-return within kernel

- A system call differs from usual routine call that supposes some service and return
- For instance, a disc I/O request results in saving the processor’s context in the current process PCB, creation of I/O control block, inserting it into the corresponding driver’s queue, blocking the process and scheduler call to choose and run the new current process
- A disk driver gets started on the device readiness interrupt and starts I/O operation on the disc; with the next readiness interrupt, it finishes I/O request and changes the process state
- In due order, the ready process gets chosen by the scheduler and continues running after copying the context from PCB to processor

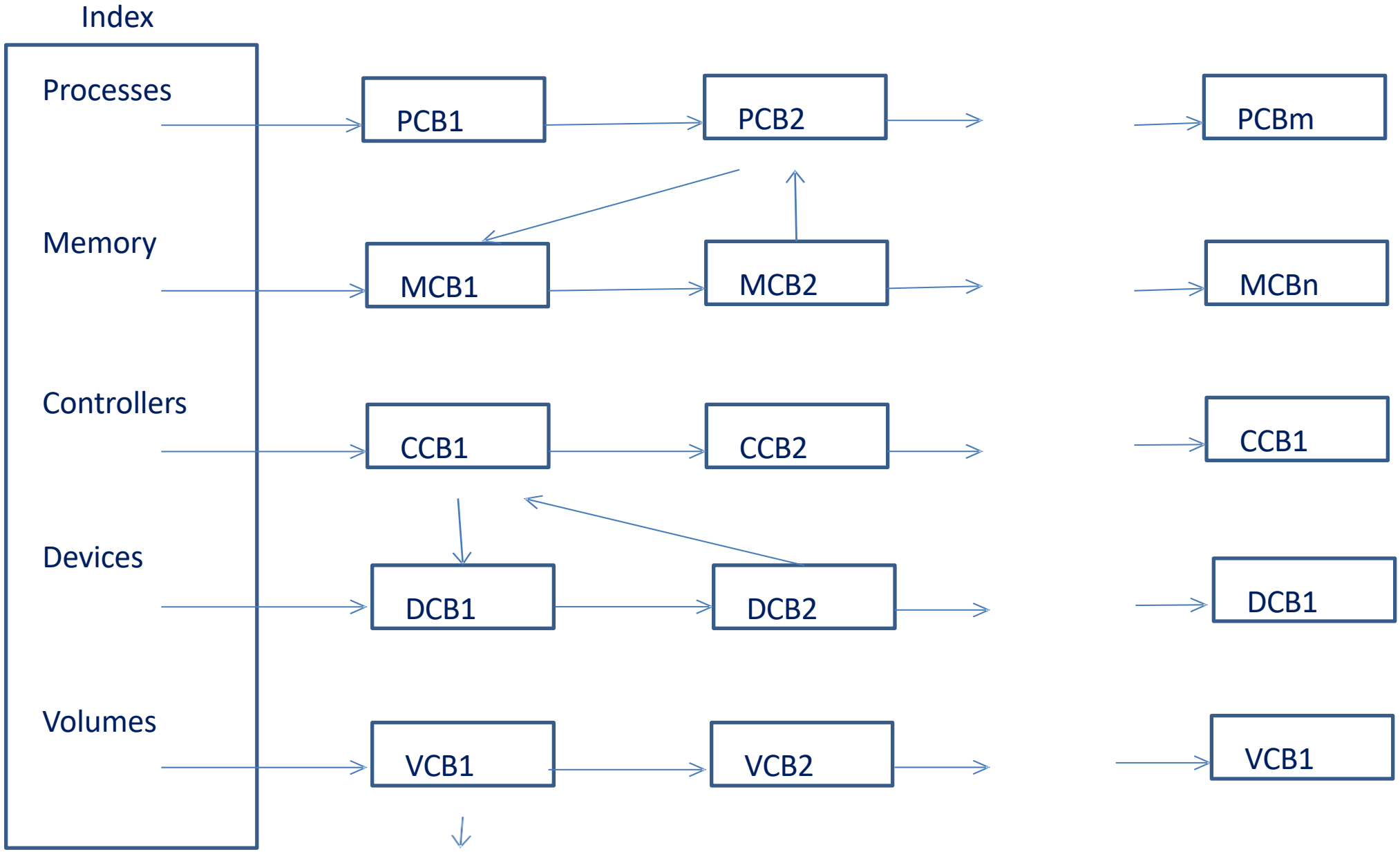
Environment of kernel

- Programs: dispatchers (switches), functional modules, auxiliary modules – working with queues, etc. Some programs among 4x4 (resources x operations) categories have special names: scheduler of processes, driver of device.
- To implement function call, kernel has stack
- Basic data of OS kernel is represented as multilinked listed of control blocks for various kinds of resources, traditionally called a data base of OS
- Control blocks are created dynamically for resources and requests, for instance I/O requests or IPC system calls using memory pool of kernel

Data base of OS

- Index – a list of control block queues headers
- Control blocks: process control block (PCB); memory control block (MCB) – recently page tables; device control block, controller (channel) control block; volume, directory, and file control block, etc.
- Multiple links of control blocks, specifying dependencies between resources, e.g., a link from PCB to MCB specifying memory allocated to a process
- Dynamical memory allocation from kernel pool, pre-allocation of a certain default number of empty blocks

Data base of OS scheme



Linux – OS DB access

- Kernel: vmlinuz
- List kernel modules: lsmod
- List of devices, represented as a special directory
- /dev
- Basic devices: tty – terminal; hd – hard drive; etc.
- Data base of OS represented as a special directory
- /proc
- Basic subdirectories: PID – process specification; meminfo – memory information; net – network information

C program design in Linux

- Traditional programming using CLI tools
  - Text editor: *nano, vi, gedit*
  - Compiler-linker: *gcc*
  - Project manager: *make*
- Interactive programming within IDE
  - Eclipse C/C++
  - MS VS Code
  - etc

Building and running a program

>gedit name.c # edit program

>gcc -o name name.c # compile & link program

>./name # run program

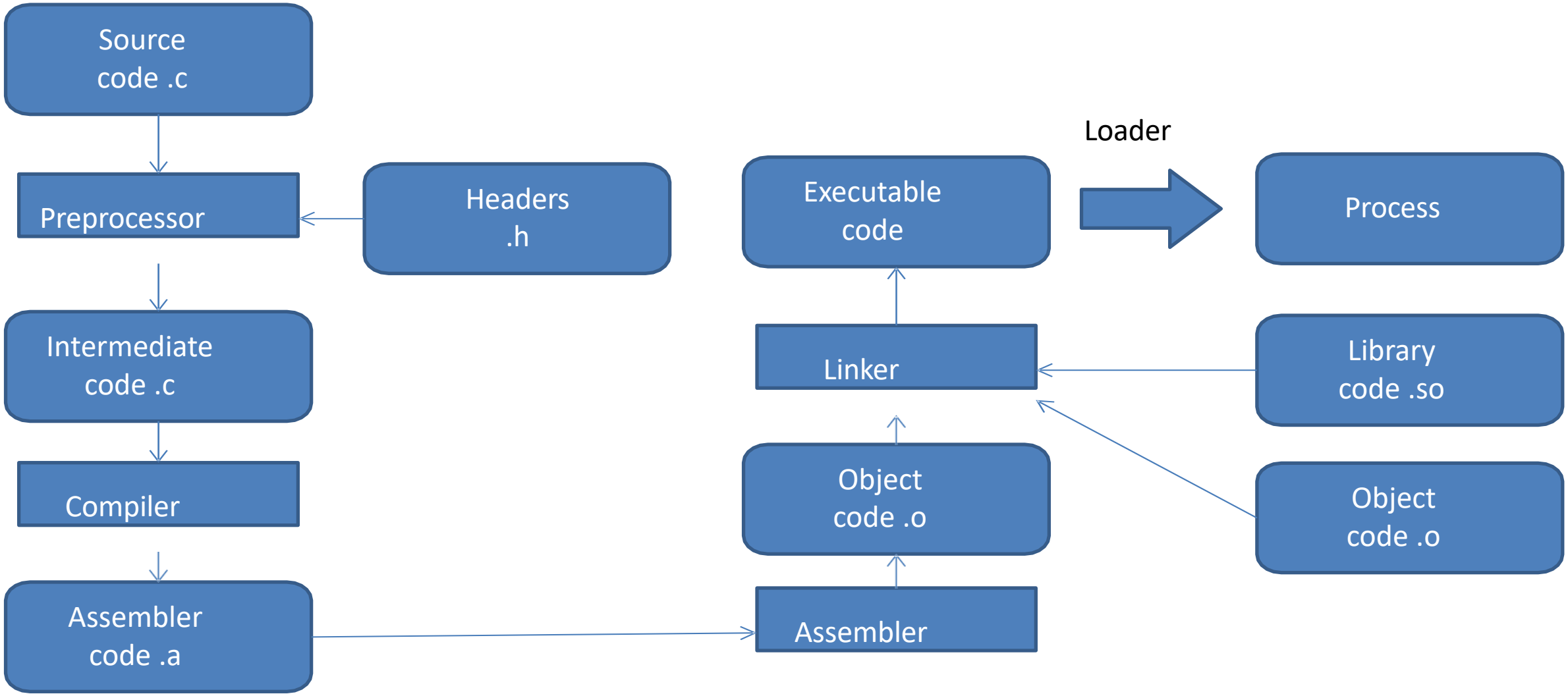
Keys:

-lm *math functions;*

-lrt *real time functions;*

-lpthread *multi-thread functions*

Stages of gcc work



An overview of libc functions

- Creation and termination of processes
- Inter Process Communication
- Input/Output: Streams & Low-Level
- Dynamic memory & shared memory
- Sockets
- Mathematics
- Searching, Sorting, and Pattern Matching
- Date & Time
- etc

Saving intermediate results of gcc

- Preprocessor – obtain C code without preprocessor directives

  gcc -E

- Compile (to Assembly) – obtain assembly code

  gcc -S

- Compile (to Object) – obtain object code – machine code with external symbols unresolved

  gcc -c

An example of C program

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

#define min2(x,y) (((x)<(y))? (x): (y))


int main(int argc, char * argv[])
{
    int a=atoi(argv[1]), b=atoi(argv[2]);
    printf("minimum of %d and %d is %d\n", a, b, min2(a,b));
}
```

Basic headers of libc functions

- stdio.h – stream input/output
- stdlib.h – frequently used functions
- unistd.h – Unix standard
- fcntl.h – control flags
- semaphore.h – semaphores
- mqueue.h – message queue

I/O on Streams

- FILE
- FILE * stdin; FILE * stdout; FILE * stderr;
- Opening Streams

FILE * fopen (const char *filename, const char *opentype)

- Closing Streams

int fclose (FILE *stream)

Simple I/O by Characters or Lines

- int fputc (int c, FILE *stream)
- int putchar (int c)
- int fputs (const char *s, FILE *stream)

- int fgetc (FILE *stream)
- int getchar (void)
- char * fgets (char *s, int count, FILE *stream)

Block Input/Output

- Read block

  size_t fread (void *data, size t size, size t count, FILE *stream)

- Write block

  size_t fwrite (const void *data, size t size, size t count, FILE *stream)

Formatted Output

- int printf (const char *template, . . .)
- int fprintf (FILE *stream, const char *template, . . .)
- int sprintf (char *s, const char *template, . . .)
- int scanf (const char *template, . . .)
- int fscanf (FILE *stream, const char *template, . . .)
- int sscanf (const char *s, const char *template, . . .)

I/O formats

- Output

% [flags width [.precision] type] conversion

%c, %s, %d, %f

- Input

% [flags width type] conversion

- Flags: +, -, 0, #
- Type: h, l

Low-Level I/O

- #include <unistd.h>
- #include <sys/types.h>
- #include <sys/stat.h>
- #include <fcntl.h>
- Given a pathname for a file, open() returns a file descriptor, a small, nonnegative integer.
- A call to open() creates a new open file description, an entry in the system-wide table of open files.
- The open file description records the file offset and the file status flags (see below).

Open and possibly create a file

- int open(const char *pathname, int flags);
- int open(const char *pathname, int flags, mode_t mode);
- Flags: access mode O_RDONLY, O_WRONLY, O_RDWR;

  file creation O_CLOEXEC, O_CREAT, O_DIRECTORY, O_EXCL, O_NOCTTY, O_NOFOLLOW, O_TMPFILE, O_TRUNC
- Mode: S_IRWXU, S_IRUSR, S_IXUSR,…

Read, Write, and Close

- Read from a file descriptor

ssize_t read(int fd, void *buf, size_t count);

- Write to a file descriptor

ssize_t write(int fd, const void *buf, size_t count);

- Close a file descriptor int close(int fd);

Reposition read/write file offset

- off_t lseek(int fd, off_t offset, int whence);
- Whence: SEEK_SET, SEEK_CUR, SEEK_END
- How: SEEK_DATA, SEEK_HOLE