HI!

CS 3841 Operating Systems
An Introduction to Operating Systems

- Objectives
  - List and characterize operating systems services (User interface, program execution, IO, file system manipulation, communications, error detection, resource allocation, accounting, protection and security)
  - Compare and contrast the command interpreter and graphical user interface approaches to interface with the computer.
  - Compare and contrast approaches to command interpreter implementation
  - List various UNIX shells
  - Explain how a system call is made
  - Explain the concept of a system call
  - Explain the usage of the malloc and free operations within the C programming language.
  - Construct simple C programs which use malloc and free to solve problems.
  - Implement Screen and File I/O in C, showing how the system calls are invoked
  - Describe various methods for handling parameters as they are passed to System calls.

Review

- What are the two view of an operating system?

System View

User View

- What are two modes within an operating system and why do we have them?

Kernel Mode

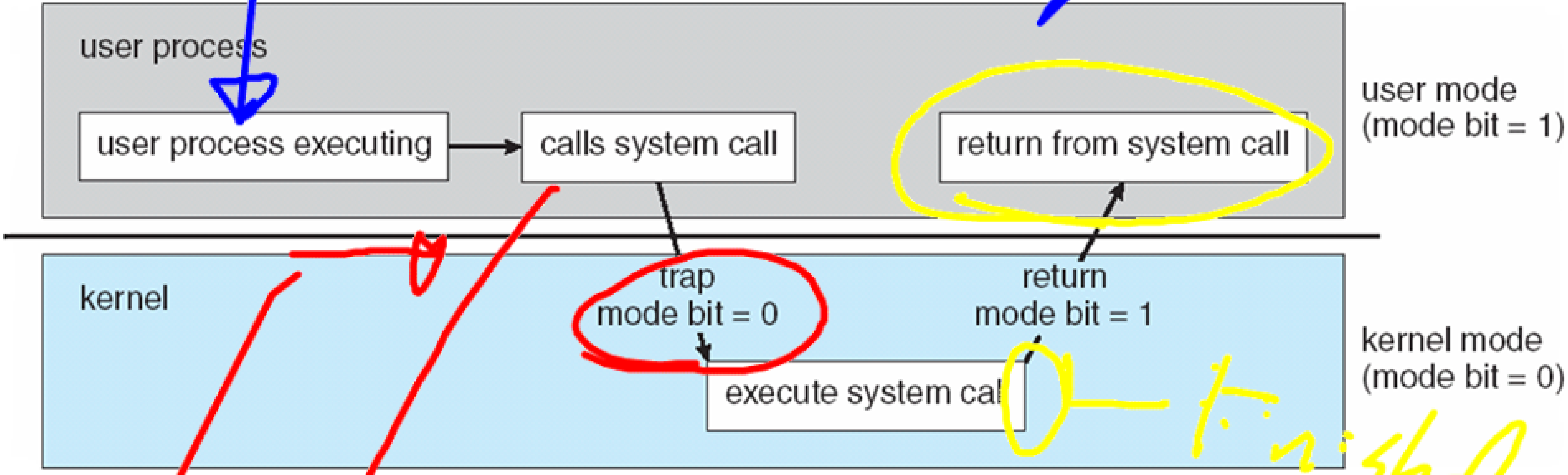
User Mode

Dual Mode Operation

- Kernel Mode
  - Also referred to as supervisor mode, system mode, or privileged mode
  - Protects the operating system from errant users
  - Typically used for Device driver code, timers, interrupts, etc.
- User mode
  - General mode in which the system operates
  - Trying to execute a privileged instruction will cause an exception handler to execute

Transitioning Between Modes

Doing something



user process

user process executing

calls system call

return from system call

user mode
(mode bit = 1)

kernel

trap
mode bit = 0

return
mode bit = 1

execute system call

kernel mode
(mode bit = 0)

finished

fwrite(1);

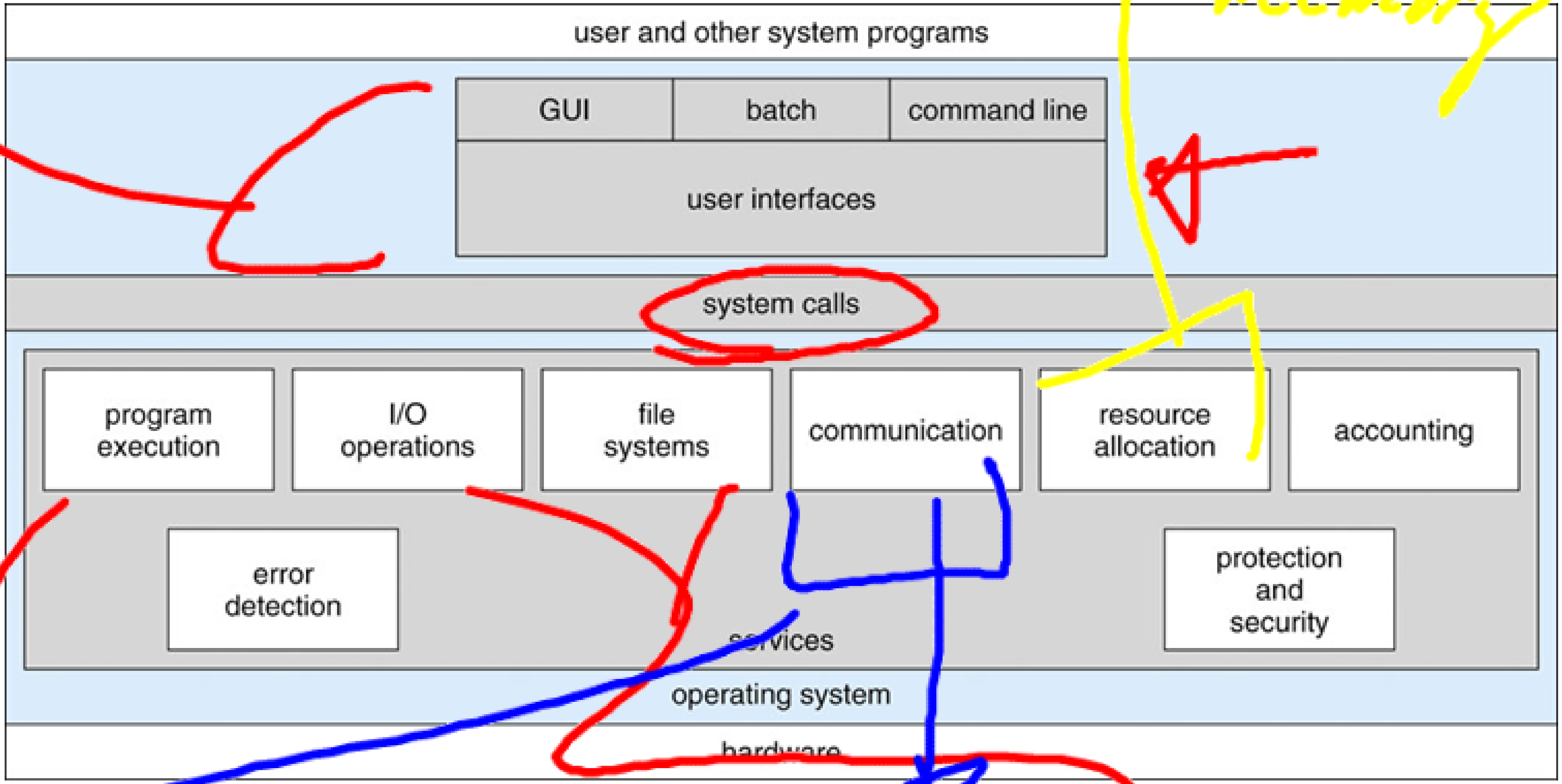
Context switch

Mode change

Operating Systems Services

High level

Give me memory



Spawn a thread

Open a file

Send a Socket msg.

Two models for Command Interpreters

CMDLine

Console

- Command Interpreter integrated into the Kernel =

Built as a piece of the kernel

- Command Interpreter is simply another running process

Same rules as any other process

Which ever model is used, main purpose is to interpret the user supplied command!

Two models for command interpreter design

- Monolithic command interpreter
  - Single large program contains the code to execute the command
- Independent system programs
  - Command interpreter simply knows how to search for the right program

Unix Shells

Scripting Language

CMD interpreter

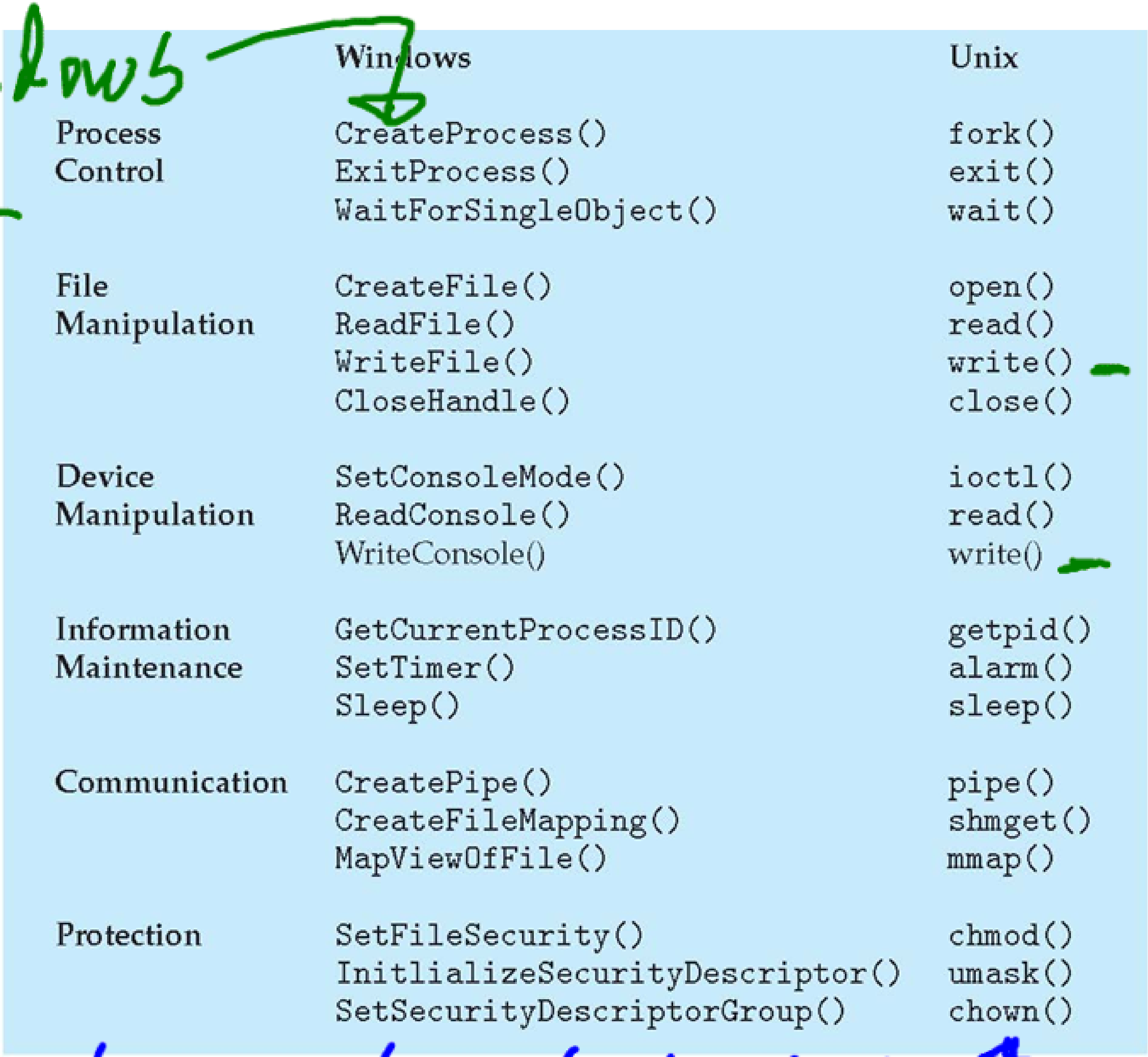
Bourn Shell -s:

Unix Shells

- Bourne Shell (1977) - sh
  - Unix Version 7 shell
- C shell (1978) csh
  - BSD Unix shell
  - Offered history, aliases, etc.
- Korn Shell (1983) - ksh
  - AT&T Bell Labs Development
  - Allows user to edit command entries in WSWIG Fashion
- Bourne Again Shell (1989) - bash
  - "Bourne Again Shell"
  - Superset of the Bourne Shell
  - Includes ideas from CSH and KSh

System Calls

Windows 32



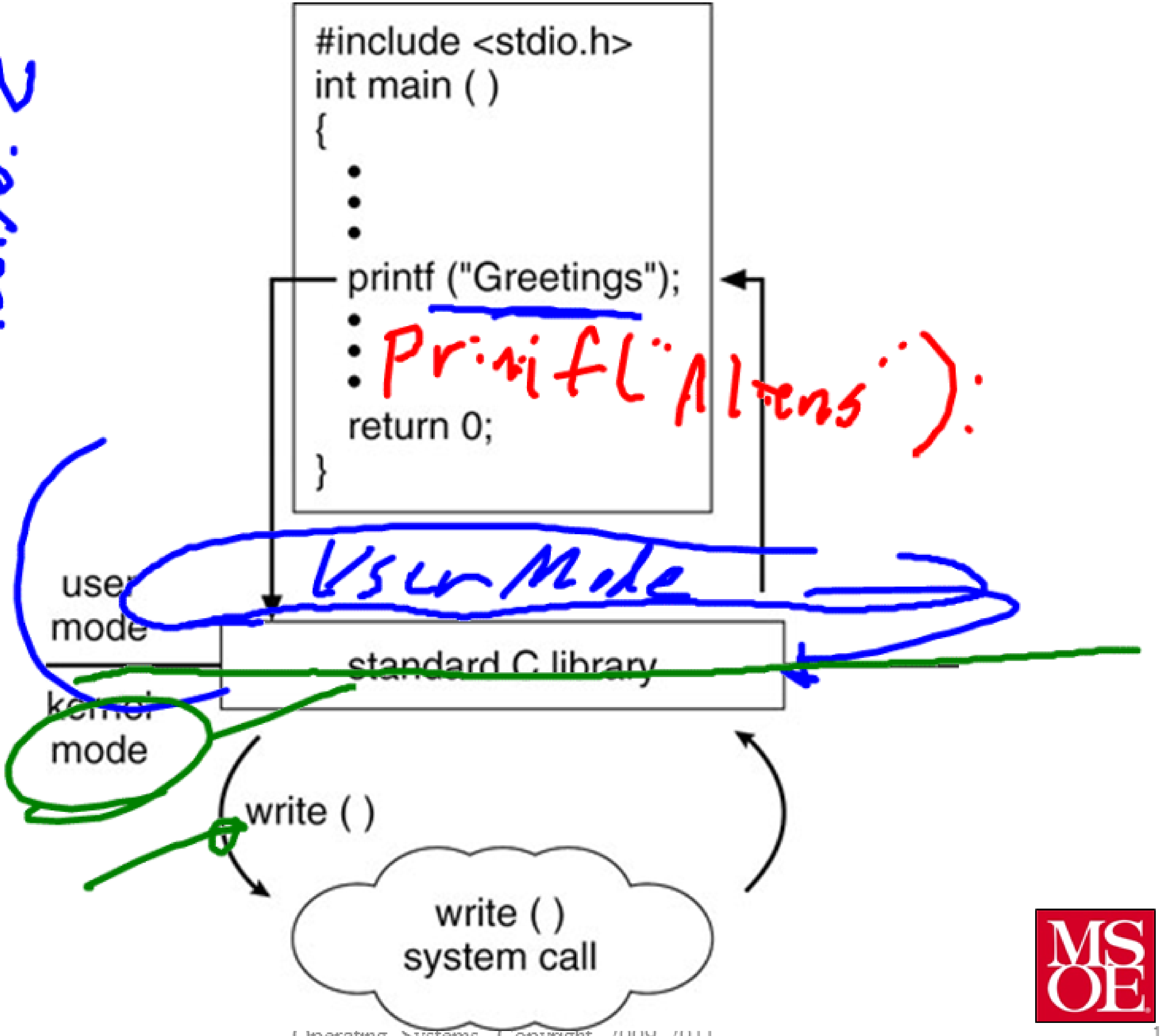
	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Portable Operating System Interface

POSIX

Printing to the Screen in C

stdlib. c

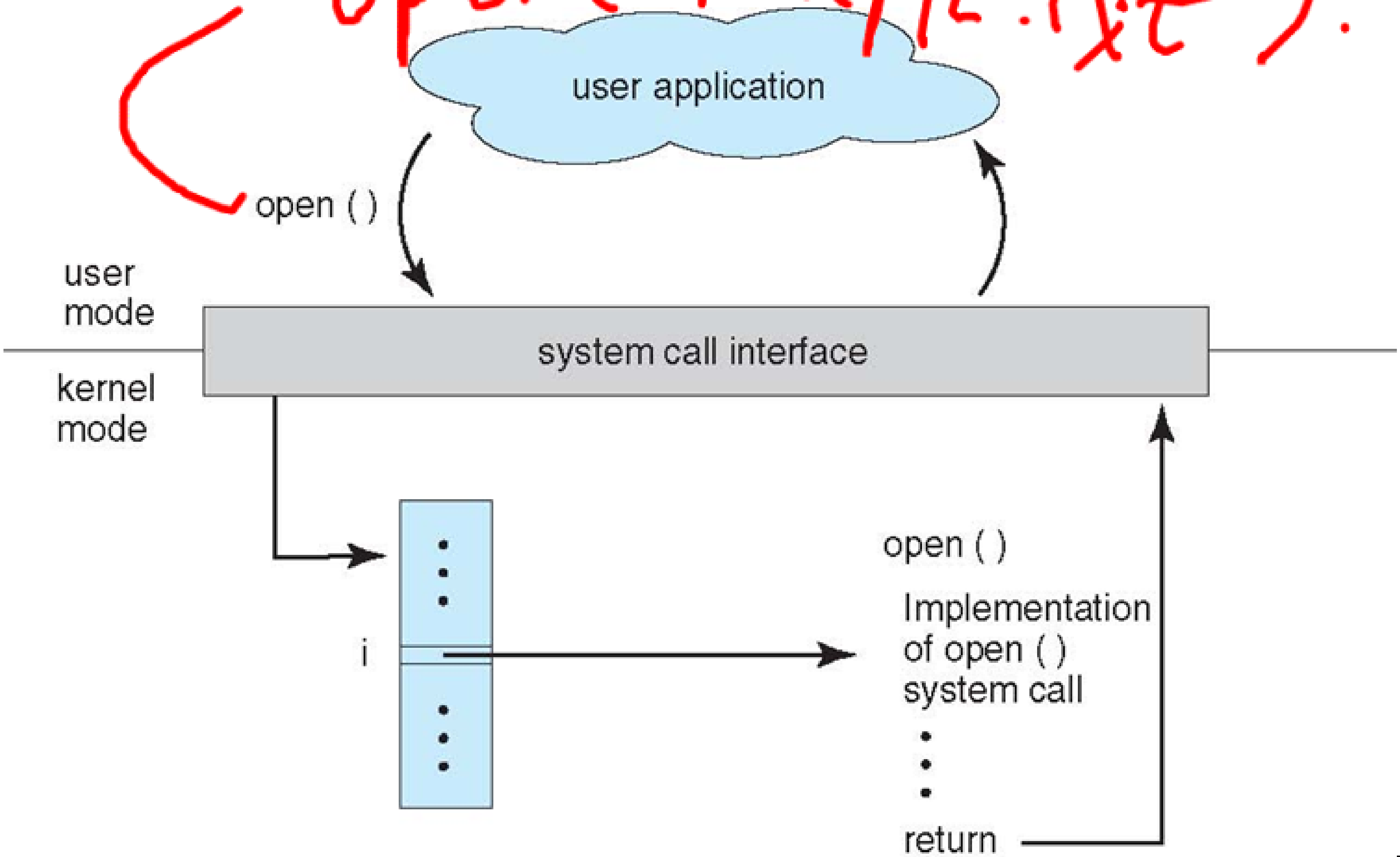


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

Printf("Aliens");

API – System Call – OS Relationship

Open("recipe.txt");



Lets write a c program to do this…

- Lets write a program to read a file and print it to the screen.

Lets write a c program to do this…

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  FILE* fptr;
  fptr = fopen(argv[1], "r");
  while (!feof(fptr))
    {
      unsigned char text[255];
      fscanf(fptr, "%s", text);
      printf("%s\n", text);
    }
  fclose(fptr);
}
```

Malloc and Free in C

- Malloc
  - Allocates a region in memory of a given size

NULL if a problem occurs.

  - Returns a void pointer
  - void* malloc (size_t size)

Sizeof block in bytes.

- Free
  - Deallocates a region of memory previously allocated by malloc
  - Must only be called once for a given region
  - void free(void *ptr)

Example

- Lets write a program to read a text file in and print it back out to the console
  - File to be read in and stored as an array of c strings
  - Each word to be stored as a separate entry.

Handling Parameters

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a *block,* or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed