CS35L – Winter 2019

Slide set:	5.1
Slide topics:	System Call Programming
Assignment:	5

Ternary Operator

- Short form for a conditional assignment

result = a > b ? x : y; is equivalent to:

```
if(a>b)
{
  result = x;
}
else
{
  result = y;
 }
```

Lab 4

- Download old version of coreutils with buggy ls program
  - Untar, configure, make
- Bug: ls -t mishandles files whose time stamps are very far in the past. It seems to act as if they are in the future

$ tmp=$(mktemp -d)

$ cd $tmp

$ touch -d '1918-11-11 11:00 GMT' wwi-armistice

$ touch now

$ sleep 1

$ touch now1

$ cd <your install-dir>/bin

$ ./ls –lt $tmp

Output:

-rw-r--r-- 1 eggert eggert 0 Nov 11 1918 wwi-armistice

-rw-r--r-- 1 eggert eggert 0 Feb 5 15:57 now1

-rw-r--r-- 1 eggert eggert 0 Feb 5 15:57 now

Goal: Fix the Bug

- **Reproduce the Bug**
  - Follow steps on lab web page
- **Simplify input**
  - Run ls with –l and –t options only
- **Debug**
  - Use gdb to figure out what's wrong
  - $ gdb ./ls
  - (gdb) run –lt wwi-armistice now now1

  (run from the directory where the compiled ls lives)
- **Patch**
  - Construct a patch "lab4.diff" containing your fix
  - It should contain a ChangeLog entry followed by the output of diff -u

Lab Hints

- Use “info functions” to look for relevant starting point
- Use “info locals” to check values of local variables
- Compiler optimizations: -O2 -> -O0
  - ./configure CFLAGS="…-O0“
  - Or, during make: make CFLAGS=‘-g –O0’

Initializing array using Malloc

```
int *arr = malloc (sizeof (int) * n); /* n is the length of the array */
int i;

for (i=0; i<n; i++)
{
  arr[i] = 0;
}
```

Task 1

Program Statement – Define a structure called student that will describe the following information.

name (char *array)
Uid (int)

Then create an array (of size 3) of this structure type.

struct student <array name>[3]; //access attributes using <array name>[index].attributename}

Using student, declare an array student with 3 elements and write a program to read the information about all the 3 students and print a sorted name wise list (sort by team name) containing names of students with their UIDs.

*you can hardcode the data for your convenience

Use the qsort function

Task 2

/*Using structures to calculate the area of a rectangle*/

Create two structs for Rectangle and Point.

Calculate the area of the rectangle using the given coordinates (top left and bottom right)

Use the below structure:

```
typedef struct {
   Point topLeft;   /* top left point of rectangle */
   Point botRight;  /* bottom right point of rectangle */
} Rectangle;
```

Task 3

Write a C program using getchar() and putchar() which continuously takes user input and prints it on the screen. This should keep on happening till the user inputs a string containing ‘#’ and Enters.

Hint: use while(getchar() != #)

Task 4

Write the following line in a file called file.txt

The value stored is 100

Use fscanf to read the value 100 from file.txt and store it in a variable <var>.

Then write this value to another file file1.txt “Value read is <var>” using fprintf

Gdb Important Resources

- Gdb cheat sheet
- Gdb command tutorial and slides
- Running gdb with emacs

Homework 4

- Write a C program called *sfrob*
  - Reads stdin byte-by-byte **(getchar)**
    - Consists of records that are newline-delimited
  - Each byte is frobnicated (XOR with dec 42)
    - Sort records without decoding (**qsort, frobcmp**)
    - Output result in frobnicated encoding to stdout **(putchar)**
  - Dynamic memory allocation (**malloc, realloc, free**)

Example

- Input: printf 'sybjre obl’
  - $ printf 'sybjre obl**\n**' | ./sfrob
- Read the records: sybjre, obl
- Compare records using *frobcmp* function
- Use *frobcmp* as compare function in *qsort*
- Output: obl

  sybjre

Homework Hints

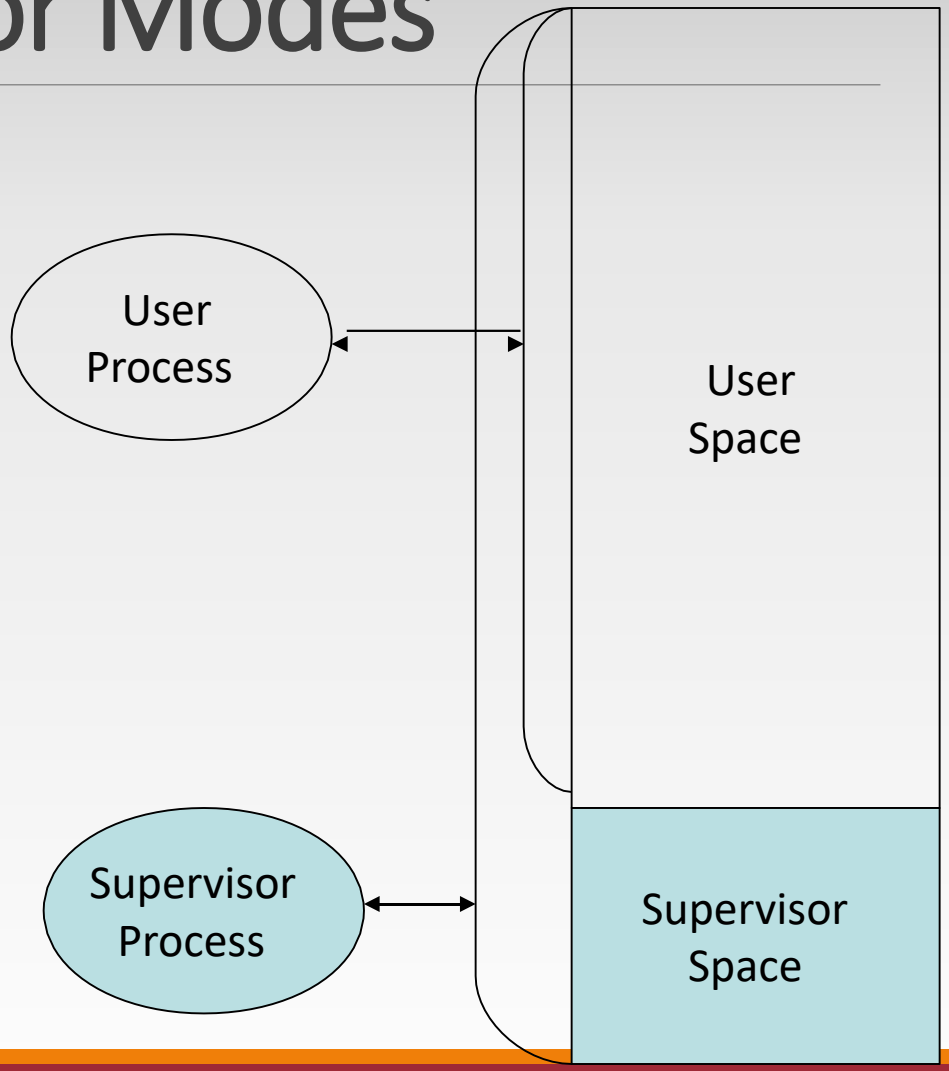
- Array of pointers to char arrays to store strings (char ** arr)
- Use the right cast while passing frobcmp to qsort
  - cast from void * to char ** and then dereference because frobcmp takes a char *
- Use realloc to reallocate memory for every string and the array of strings itself, dynamically
- Use *exit*, not *return* when exiting with error

System Call Programming

Processor Modes

Operating modes that place restrictions on the type of operations that can be performed by running processes

- User mode: restricted access to system resources
- Kernel/Supervisor mode: unrestricted access



User Mode vs. Kernel Mode

Hardware contains a mode-bit, e.g. 0 means kernel mode, 1 means user mode

User mode

- CPU **restricted** to unprivileged instructions and a specified area of memory

Supervisor/kernel mode

- CPU is **unrestricted**, can use all instructions, access all areas of memory and take over the CPU anytime

Why Dual-Mode Operation?

System resources are shared among processes

OS must ensure:

- **Protection**
  - an incorrect/malicious program cannot cause damage to other processes or the system as a whole
- **Fairness**
  - Make sure processes have a fair use of devices and the CPU

Goals for Protection and Fairness

Goals:

- **I/O Protection**
  - Prevent processes from performing illegal I/O operations
- **Memory Protection**
  - Prevent processes from accessing illegal memory and modifying kernel code and data structures
- **CPU Protection**
  - Prevent a process from using the CPU for too long

=> instructions that might affect goals are privileged and can only be executed by *trusted code*

Which Code is Trusted?
=> The Kernel *ONLY*

- Core of OS software **executing in supervisor** state
- **Trusted software:**
  - Manages hardware resources (CPU, Memory and I/O)
  - Implements protection mechanisms that could not be changed through actions of untrusted software in user space
- **System call interface** is a **safe way** to expose privileged functionality and services of the processor

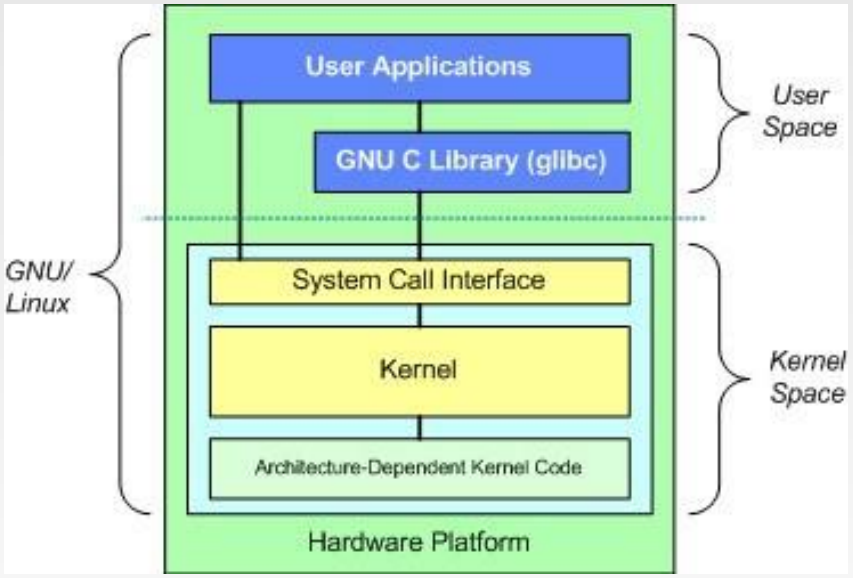
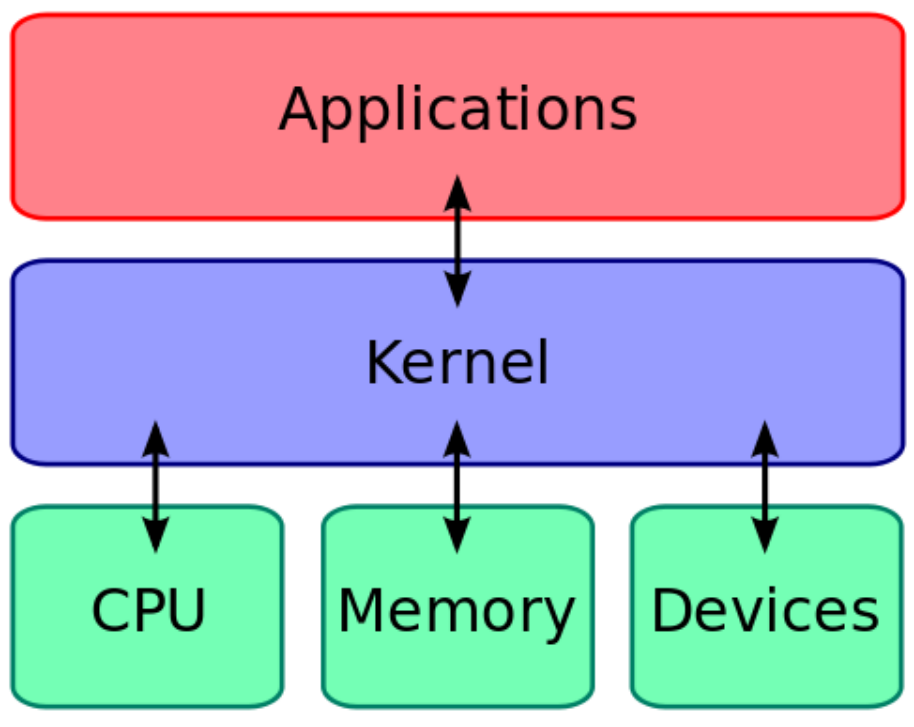


Image by: Tim Jones (IBM)

What About User Processes?

The kernel executes privileged operations on behalf of untrusted user processes



System Calls

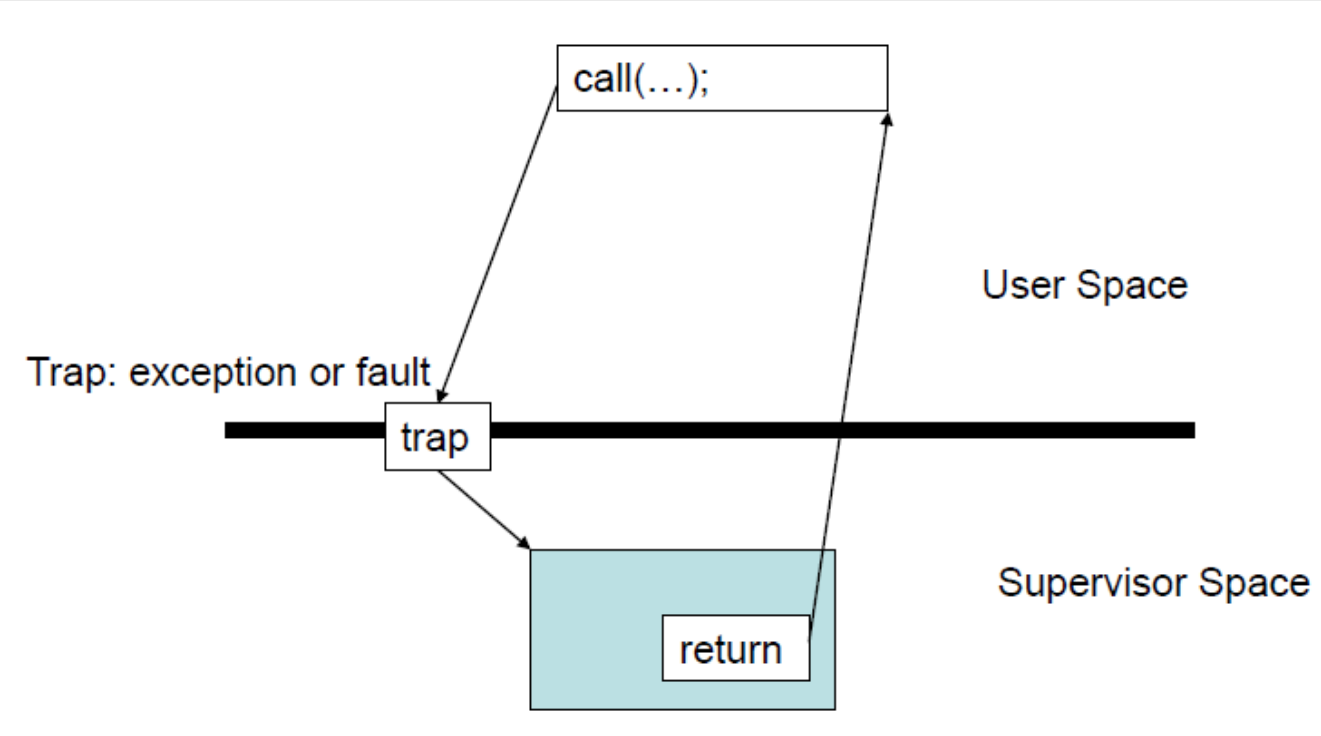
Special type of function that:

- Used by user-level processes to request a service from the kernel
- Changes the CPU's mode from user mode to kernel mode to enable more capabilities
- Is part of the kernel of the OS
- Verifies that the user should be allowed to do the requested action and then does the action (kernel performs the operation on behalf of the user)
- Is the ***only way*** a user program can perform privileged operations

System Calls

When a system call is made, the program being executed is interrupted and control is passed to the kernel

If operation is valid the kernel performs it



System Call Overhead

System calls are expensive and can hurt performance

The system must do many things

- Process is interrupted & computer saves its state
- OS takes control of CPU & verifies validity of op.
- **OS performs requested action**
- OS restores saved context, switches to user mode
- OS gives control of the CPU back to user process

Example System Calls

- `#include<unistd.h>`

  `ssize_t` **`read`**`(int fildes, void *buf, size_t nbyte)`
  - fildes: file descriptor
  - buf: buffer to write to
  - nbyte: number of bytes to read
- `ssize_t` **`write`**`(int` *`fildes`*`, const void *`*`buf`*`, size_t` *`nbyte`*`);`
  - fildes: file descriptor
  - buf: buffer to write from
  - nbyte: number of bytes to write
- `int` **`open`**`(const char *pathname, int flags, mode_t mode);`
- `int` **`close`**`(int fd);`
- File descriptors
  - 0 stdin
  - 1 stdout
  - 2 stderr

Example System Calls

- `pid_t` **`getpid`**`(void)`
  - Returns the process ID of the calling process
- `int` **`dup`**`(int fd)`
  - Duplicates a file descriptor fd. Returns a second file descriptor that points to the same file table entry as fd does.
- `int` **`fstat`**`(int filedes, struct stat *buf)`
  - Returns information about the file with the descriptor filedes into buf

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

Library Functions

Functions that are a part of standard C library

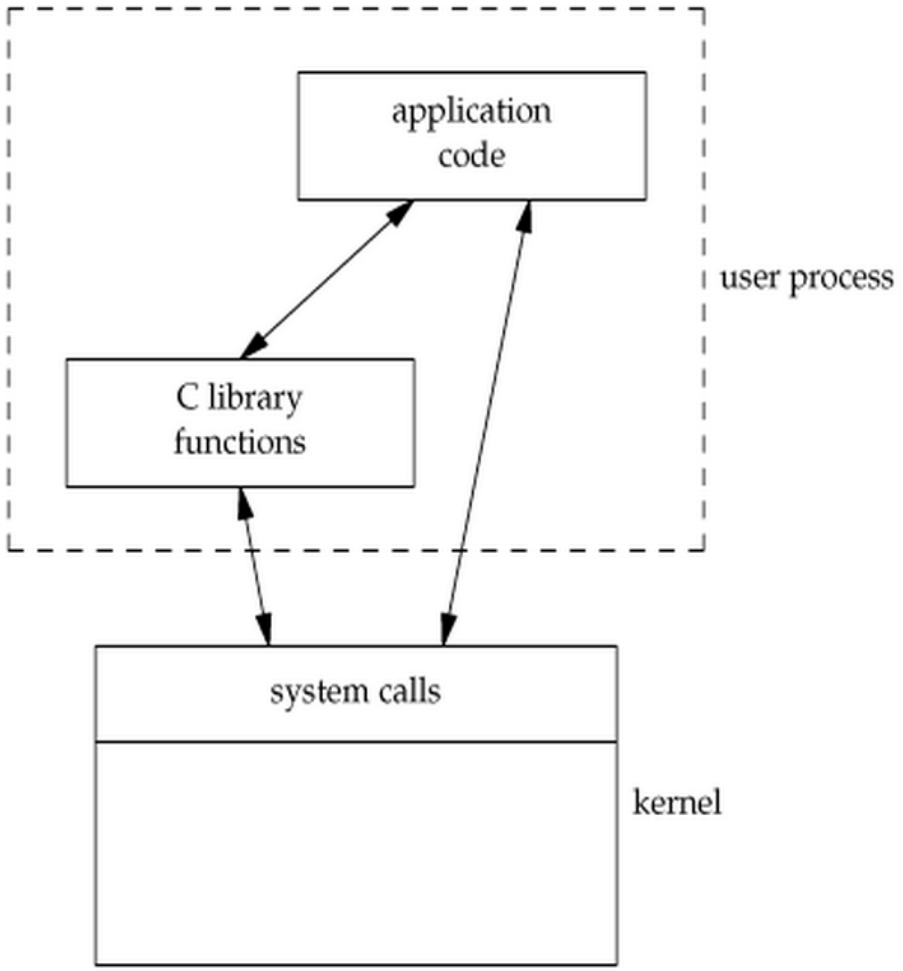
To avoid system call overhead use equivalent library functions

- getchar, putchar vs. read, write (for standard I/O)
- fopen, fclose vs. open, close (for file I/O), etc.

How do these functions perform privileged operations?

- They make system calls

So What's the Point?



- Many library functions invoke system calls indirectly
- So why use library calls?
- Usually equivalent library functions make fewer system calls
- non-frequent switches from user mode to kernel mode => less overhead

Task 1 Hint

```
int compare (const void * a, const void * b )
{
  struct student *pa = (struct student*)a;
  struct student *pb = (struct student*)b;
  return strcmp(pa->name, pb->name);
}
qsort(<arrayname>,5, sizeof(struct student),compare);
```

*you can also typedef to avoid writing ‘struct’

Task 2 Solution

```
#include <stdio.h>
#include <string.h>
#include <math.h>

typedef struct {
   double x;
   double y;
} Point;

typedef struct {
   Point topLeft;   /* top left point of rectangle */
   Point botRight;  /* bottom right point of rectangle */
} Rectangle;
double computeArea(Rectangle *r);
```

```
int main()
{
   Point p;
   Rectangle r;
   printf("\nEnter top left point: ");
   scanf("%lf", &r.topLeft.x);
   scanf("%lf", &r.topLeft.y);
   printf("Enter bottom right point: ");
   scanf("%lf", &r.botRight.x);
   scanf("%lf", &r.botRight.y);
   printf("Top left x = %lf  y = %lf\n", r.topLeft.x, r.topLeft.y);
   printf("Bottom right x = %lf  y = %lf\n", r.botRight.x, r.botRight.y);
   printf("Area = %f", computeArea(&r));
   return 0;
}
```

```
double computeArea(Rectangle *r)
{
   double height, width, area;

   height = ((r->topLeft.y) - (r->botRight.y));
   width = ((r->topLeft.x) - (r->botRight.x));
   area = height*width;
   return (area);
}
```

Task 3 solution

```
#include <stdio.h>
/* --  Copy input to output -- */
int main(void)
{
    int c;
    c = getchar();
    while ( c != "#" ) {
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

Task 4 solution

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int a;
FILE * fp;
FILE * fp1;
fp = fopen("file.txt","r+");
fp1 = fopen("file1.txt", "w+");
fscanf(fp, "This is the value %d", &a);
fprintf(fp1, "Value read is %d",a);
fclose(fp);
return 0;
}
```