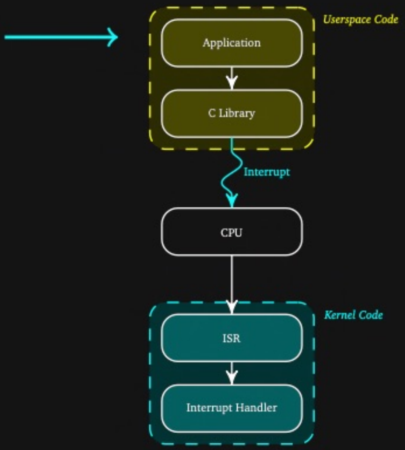
System Call Implementation Example

- The following example is taken from a miniature operating system written by Prof. Michael Black[1]
- The operating system consists of a small kernel, a shell, a simple GUI and a rudimentary filesystem
- You can download the complete source from the lecture server

SHELL.C

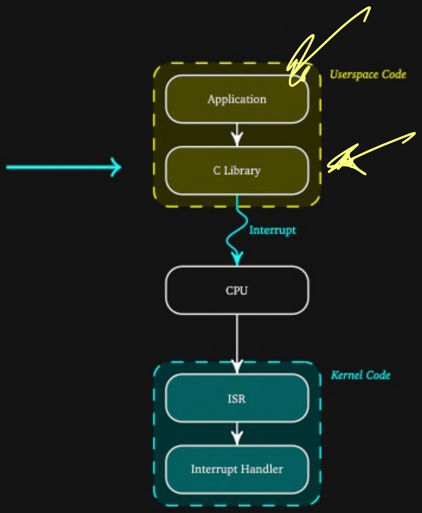
#include <iostream>

```c
/* delete a file */
void dodelete(char* line) {
  char* name=getargument(line);
  /* make the system call */
  deletefile(name);
}
```



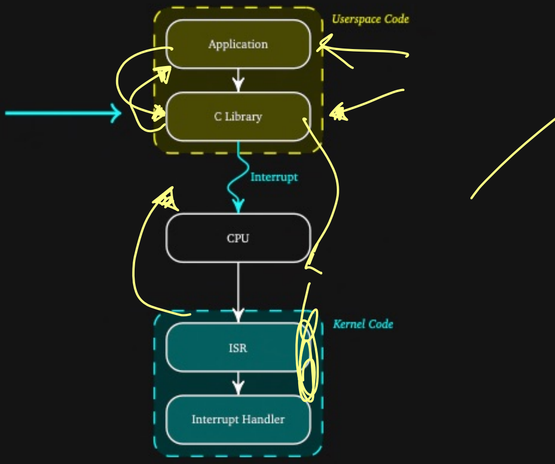
LIB.H

```c
void readsector(int, char*);
void writesector(int, char*);
void putchar(char);
char getchar();
void printstring(char*);
void printnumber(int);
void readstring(char*);
void readfile(char*, char*);
void writefile(char*, char*, int);
void deletefile(char*);
void exit();
void executeprogram(char*, int,char*);
void allow_preemption();
int mod(int,int);
int div(int,int);
void setvideo(int);
void setpixel(int,int,int);
void clearscreen();
void setcursor(int,int);
void setchar(char,char,int,int);
void setstring(char*,char,int,int);
void getnumberstring(char*,int);
```



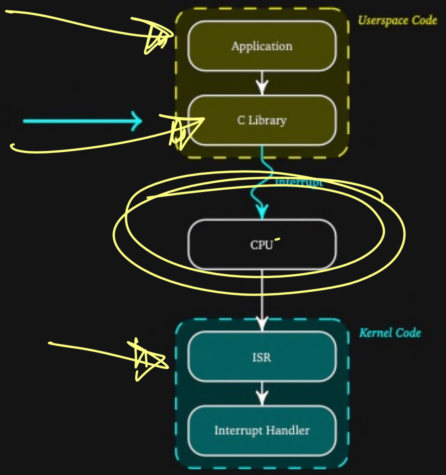
LIB.C

```c
/* delete the file name[] */
void deletefile(char* name) {
  int21(5,name);
}
```



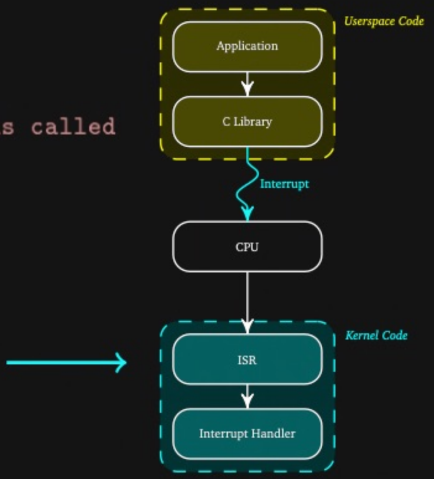
LIB.ASM

```
;invoke int 21
;this can take an arbitrary number of parameters - extra parameters will just be garbage
;inputs: AH code (char), BX (int / address), CX (int / address), DX (int / address)
_int21:
    sti

    mov di,sp
    mov ah,[di+2]
    mov bx,[di+4]
    mov cx,[di+6]
    mov dx,[di+8]

    int 0x21

    ret
```



KERNEL.ASM

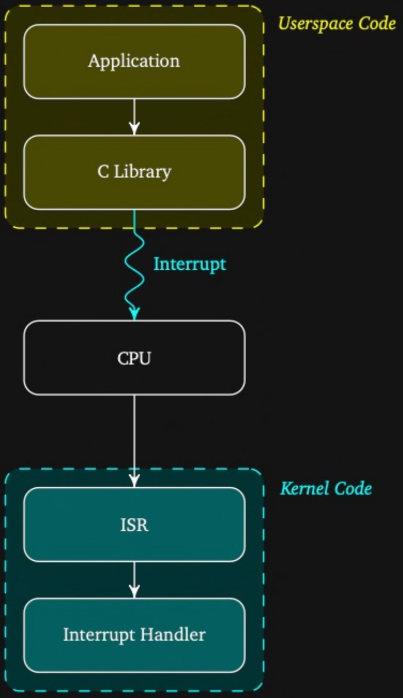
```
;this is called immediately on an interrupt 0x21.
int21_ISR:
    push ds

    ;let's call a C interrupt handler
    ;pass it the contents of ah - this tells which interrupt was called
    ;pass it the contents of bx,cx,dx - the parameters
    mov al,ah
    mov ah,#0

    push dx
    push cx
    push bx
    push ax
    call _handleinterrupt21
    pop ax
    pop bx
    pop cx
    pop dx

    pop ds

    iret
```



Introduction

- System calls are initiated by the userspace applications
- To provide ease of use, there is usually an intermediate library routine
  - In Linux – libc
  - In Windows – the Win32 API
- The library encapsulates the interrupt number and other complexities (including setting the registers)
- Interrupt (trap) is received by the kernel's ISR
- The assembly code of the ISR routes the call to a handler written in a high-level language

System Call Flow of Control


Userspace Code
Application
C Library
Interrupt
CPU
Kernel Code
ISR
Interrupt Handler