INPUT/OUTPUT

INTRODUCTION

System-level programming, particularly in low-level languages like Assembly or C, involves directly interacting with hardware and the operating system. Understanding how to handle input and output (I/O) at this level is crucial for developing efficient and effective software.

Benefits of Effective I/O Handling

1. **Performance**: Efficient I/O operations can significantly reduce latency and increase the throughput of data transfer between the CPU and peripheral devices.
2. **Reliability**: Proper I/O management ensures stable and predictable system behavior, avoiding data corruption and system crashes.
3. **Resource Utilization**: Optimized I/O can lead to better utilization of system resources, such as CPU and memory, allowing for more efficient multitasking.
4. **User Experience**: Fast and responsive I/O operations contribute to a smoother and more satisfactory user experience.

BASIC CONCEPTS

I/O Devices

Includes keyboards, mice, displays, storage devices, and network interfaces.

I/O Ports

Special registers used for communicating with hardware devices.

Memory-Mapped I/O

Hardware devices map their registers into the system memory space.

Interrupts: Signals to the processor indicating an event that needs immediate attention.

TYPES OF I/O OPERATIONS

Programmed I/O (Polling)

The CPU actively waits and repeatedly checks the status of an I/O device to see if it is ready for data transfer. It is widely used in simple and small-scale systems where I/O latency is not critical e.g keyboard input in basic BIOS routines.

Example: Assembly example of polling a keyboard input poll_keyboard:

```
    in al, 0x60 ; Read from keyboard port
    test al, al    ; Check if a key is pressed
    jz poll_keyboard ; Repeat until a key is pressed
    ; Process the key press
```

Interrupt-Driven I/O:

The CPU performs other tasks and is interrupted by the I/O device when it is ready for data transfer. It is mostly used in systems requiring efficient CPU usage and low latency, such as real-time applications e.g handling a network packet arrival.

// C example of an interrupt service routine (ISR) for a network packet

```
void __interrupt() net_isr() {
    // Handle the network packet
}
```

Direct Memory Access (DMA)

A DMA controller handles data transfer between memory and devices, freeing the CPU for other tasks. A DMA controller transfers data directly between memory and an I/O device without CPU intervention. It is mostly used in high-speed data transfer requirements, such as disk I/O and memory-intensive applications e.g transferring data from a disk to memory

// Pseudocode example of setting up a DMA transfer

```
dma_setup(source, destination, size);
dma_start();
```

INPUT/OUTPUT IN ASSEMBLY LANGUAGE

IN and OUT Instructions

Used to read from and write to I/O ports.

Example 1: Read a byte from port 0x60 (keyboard data port)

```
in al, 0x60
```

Example 2: Write a byte to port 0x60

```
out 0x60, al
```

Memory-Mapped I/O

Example: Writing to a memory-mapped I/O register

```
mov eax, [io_address]
mov [io_register], eax
```

Interrupt Handling

```
; Setting up an interrupt service routine (ISR)
isr:
    pusha               ; Save all general-purpose registers
    ; Handle the interrupt
    popa                ; Restore all general-purpose registers
    iret                ; Return from interrupt
```

I/O in C

Standard I/O: Using standard library functions like printf, scanf, getchar, and putchar.

```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character: ");
```

```
    c = getchar();
    printf("You entered: %c\n", c);
    return 0;
}
```

File I/O

```
#include <stdio.h>
int main() {
    FILE *file;
    char data[100];

    file = fopen("example.txt", "r");
    if (file == NULL) {
        perror("Error opening file");
        return -1;
    }

    while (fgets(data, 100, file) != NULL) {
        printf("%s", data);
    }

    fclose(file);
    return 0;
}
```

Direct I/O with System Calls

```
#include <unistd.h>
#include <fcntl.h>
```

```c
int main() {
    int fd;
    char buffer[100];

    fd = open("example.txt", O_RDONLY);
    if (fd == -1) {
        perror("Error opening file");
        return -1;
    }

    read(fd, buffer, sizeof(buffer));
    write(STDOUT_FILENO, buffer, sizeof(buffer));

    close(fd);
    return 0;
}
```

Low-Level I/O and Device Drivers

Register-Level Programming: Accessing hardware registers directly.

Writing Device Drivers: Developing software that allows the operating system to communicate with hardware devices.

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>

int init_module(void) {
    printk(KERN_INFO "Hello, World!\n");
    return 0;
}
```

```
void cleanup_module(void) {
    printk(KERN_INFO "Goodbye, World!\n");
}

MODULE_LICENSE("GPL");
```

Examples with code samples

Keyboard Input (Polling):

```
.data
msg db 'Press a key: ', 0

.code
start:
    mov ah, 0x09
    lea dx, msg
    int 0x21       ; DOS interrupt to display message

    xor ah, ah
    int 0x16       ; BIOS interrupt to read key

    mov ah, 0x4C
    int 0x21       ; DOS interrupt to terminate program
```

Writing to a Port in C

```
#include <stdio.h>
#include <sys/io.h>

int main() {
```

```
    if (ioperm(0x60, 1, 1)) {
        perror("ioperm");
        return 1;
    }

    outb(0x60, 'A');  // Write 'A' to port 0x60

    if (ioperm(0x60, 1, 0)) {
        perror("ioperm");
        return 1;
    }

    return 0;
}
```

Advanced Topics

Interrupt Handling in C

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

void handle_sigint(int sig) {
    printf("Caught signal %d\n", sig);
}

int main() {
    signal(SIGINT, handle_sigint);
    while (1) {
        printf("Running...\n");
```

```
        sleep(1);
    }
    return 0;
}
```

Memory-Mapped I/O in C

```
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

int main() {
    int fd;
    volatile unsigned int *reg;

    fd = open("/dev/mem", O_RDWR | O_SYNC);
    if (fd == -1) {
        perror("open");
        return -1;
    }

    reg = mmap(NULL, sizeof(unsigned int), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x3F200000);
    if (reg == MAP_FAILED) {
        perror("mmap");
        close(fd);
        return -1;
    }

    *reg = 0x12345678;  // Example operation
```

```
    munmap((void *)reg, sizeof(unsigned int));
    close(fd);
    return 0;
}
```