Input/Output

WEEK 10 – Lab 10

I/O Devices

Block Devices: Stores Information in Fixed Size Blocks.

// Commands include open(), read(), write(), seek().

Character Devices: Delivers or accepts stream of characters.

// Commands include get(), put()

I/O Devices

All I/O devices are represented as files.

$ /dev	Contains special device files for all devices.
$ /proc	Virtual file system.

Major and Minor numbers

```
$ /dev
$ ls -l
```

- Major number identifies driver associated with the device.
- Minor number is used by the kernel to determine which device is being referred to.

Accessing I/O ports

```
ioperm(from, num, turn_on)    // Can only give access to ports 0x000 through 0x3ff.

iopl()                        //  For higher ports, (which gives access to all ports at once).

inw(x)                        // Input from a port

outw(value, x)                // Output from a port
```

Example

```
// Ioperm Example
if (ioperm(BASEPORT, 3, 1)) {perror("ioperm");
{                                          /* Get access to the ports */
    exit(1);
}
outb(0, BASEPORT);                         /* Set the data signals (D0–7) of the port to all low (0) */
usleep(100000);                            /* Sleep for a while (100 ms) */
printf("status: %d\n", inb(BASEPORT + 1)); /* Read from the status port (BASE+1) and display the result */
if (ioperm(BASEPORT, 3, 0))                /* We don't need the ports anymore */
{
    perror("ioperm");
}
```

Memory Mapped I/O

The mmap() system call creates a new memory mapping in the calling process's virtual address space.

Once a file is mapped, its contents can be accessed by operations on the bytes in the corresponding memory region.

```
# include<sys/man.h>
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

Exercise 1

Create a file.txt. Save a string: “This is a good day” in it.

Write a C program that can change the string in file.txt to “That is a good day” by using mmap().

Hint:

- Open the file.txt in O_RDWR mode
- Use fstat () to get the size of the file

Direct Memory Access

DMA-TO-DEVICE: DMA from the Main Memory to the Device.

DMA-FROM-DEVICE: DMA from the Device to the Main Memory.

DMA-BIDIRECTIONAL: DMA from the device to main memory or from the main memory to device.

Direct Memory Access

```
int dad_transfer(struct dad_dev *dev, int write, void *buffer,size_t count)
{
dma_addr_t bus_addr;
/* Map the buffer for DMA */
dev->dma_dir = (write ? DMA_TO_DEVICE : DMA_FROM_DEVICE);
dev->dma_size = count;
bus_addr = dma_map_single(&dev->pci_dev->dev, buffer, count, dev->dma_dir);
dev->dma_addr = bus_addr;
/* Set up the device */
writeb(dev->registers.command, DAD_CMD_DISABLEDMA);
writeb(dev->registers.command, write ? DAD_CMD_WR : DAD_CMD_RD);
/* Start the operation */
writeb(dev->registers.command, DAD_CMD_ENABLEDMA); return 0;      }
```

Buffered I/O

Full buffering – Invoke read() or write() system call when the buffer becomes full.

Line buffering – On output, data is written when a newline character is inserted into the stream or when the buffer is full, what so ever happens first. On Input, the buffer is filled till the next newline character when an input operation is requested and buffer is empty.

No buffering – Directly translate every library call into a read() or write() system call.

Buffered I/O

```
int setvbuf(FILE *stream, char *buf, int mode, size_t size);

_IONBF        // unbuffered
_IOLBF        // line buffered
_IOFBF        // full buffered
```

Who can answer?

What is the default buffering type of standard streams:

- Stdout
- Stdin
- Stderr

Exercise 2

Write a C program using line buffer. Write your code according to the instructions:

Each of the 5 characters of Hello should be put in separate printf()’s. And a 1 sec sleep after every printf().

//Output should be like that:

A 5 sec wait and then Hello printed simultaneously.

Exercise 3

Run the following two codes and explain the difference in output. Save your answer in ex3.txt file.

Program 1

```
#include <stdio.h>
#include<unistd.h>
int main()
{
printf(“Hello”);
fork();
printf(“\n”);
return 0;
}
```

Program 2

```
#include <stdio.h>
#include<unistd.h>
int main()
{
printf(“Hello\n”);
fork();
printf(“\n”);
return 0;
}
```

Exercise 4

Provide answers to the following questions in ex4.txt file.

(1) Explain how an OS can facilitate installation of a new device without any need for recompiling the OS.

(2) In which of the four I/O software layers each of the following is done.

(a) Computing the track, sector, and head for a disk read.

(b) Writing commands to the device registers.

(c) Checking to see if the user is permitted to use the device.

(d) Converting binary integers to ASCII for printing.

Memcpy()

Void *memcpy(void *dest, const void *src, size_t n)

// The memcpy function copies n bytes from memory area src to memory area dest.

Parameters:

void *dest: A pointer to the destination array where the content is to be copied.

void *src: A pointer to the source of data to be copied.

n: Number of Bytes to copy

Exercise 5

Write a C program to copy the content of one file to another file using memory mappings.

//Use mmcpy()