9

Input Output

Modul 9

Input Output

9.1. Tujuan Praktikum

Praktikum ini bertujuan memperkenalkan konsep input dan output pada system operasi.

9.2. Capaian Praktikum

1. Mahasiswa mampu memahami konsep *input* pada system operasi.
2. Mahasiswa mampu memahami konsep *output* pada system operasi.

9.3. Dasar Teori

9.3.1. Pengantar I/O

Secara umum terdapat dua jenis siklus pekerjaan sebuah komputer yang dikelola oleh sistem operasi yaitu siklus pemrosesan CPU dan siklus akses input-output (I/O). Dalam konteks pemrosesan CPU, sistem operasi lebih banyak berperan untuk melakukan penjadwalan eksekusi, sinkronisasi dan pencegahan deadlock dari proses-proses yang berjalan. Di sisi lain, tugas sistem operasi menjadi lebih kompleks dalam pengelolaan I/O mengingat sangat bervariasinya karakteristik dari perangkat I/O meliputi aspek model transfer data, metode akses maupun arah akses. Sebagai ilustrasi, tabel berikut menggambarkan beberapa jenis perangkat I/O berikut dengan variasi karakteristiknya.

Aspek	Variasi	Contoh Perangkat I/O
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
I/O direction	read only write only	CD-ROM graphics controller

	read–write	disk

Penjelasan mengenai karakteristik perangkat I/O pada tabel di atas antara lain:

a. **Character-stream vs block**. Perangkat berjenis character stream mentransfer data byte-per-byte sementara perangkat block mentransfer data dalam satuan block-byte (kumpulan byte).

b. **Sequential vs random**. Perangkat berjenis sequential mengakses data secara berurutan sesuai sementara perangkat random memungkinkan akses data pada sebarang lokasi.

c. **Synchronous vs asynchronous**. Perangkat synchronous memungkinkan transfer data dengan waktu respon terprediksi sementara perangkat asynchronous waktu responnya tidak dapat diprediksi.

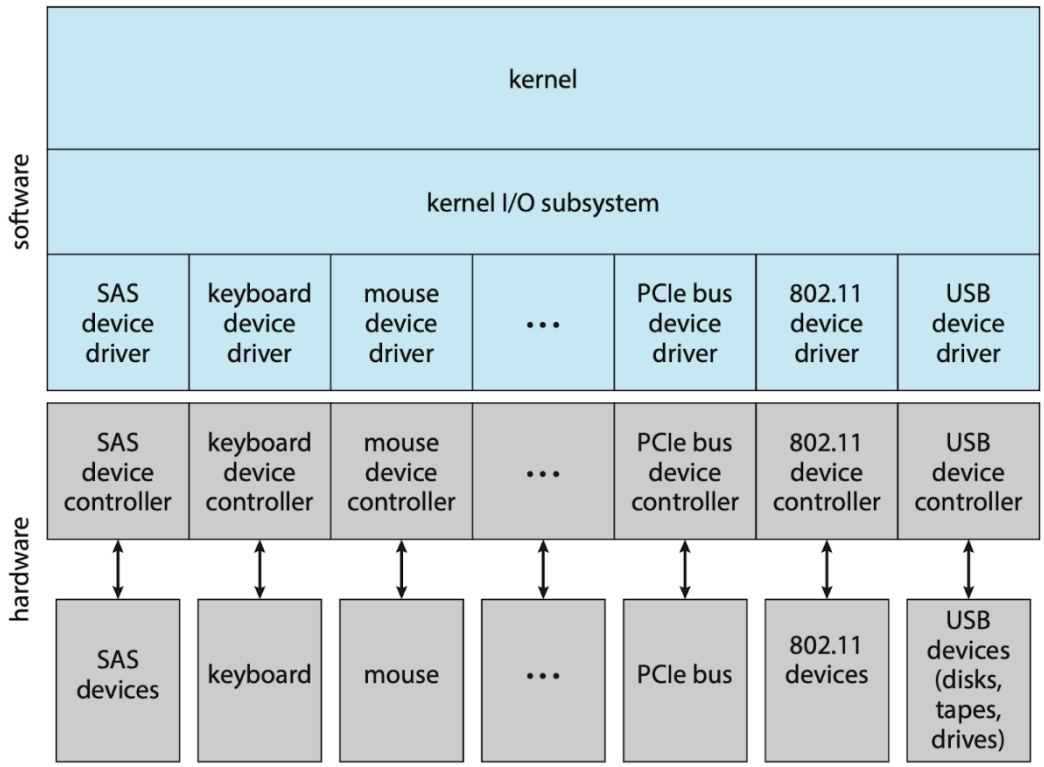
d. **Sharable vs dedicated**. Perangkat sharable dapat diakses secara concurrent oleh beberapa proses sementara perangkat dedicated tidak.

e. **Read–write,readonly,writeonce**. Beberapa perangkat mendukung model baca saja, tulis saja atau baca tulis.

9.3.2. Struktur Kernel I/O

Pada bagian software, terdapat tiga lapisan utama yaitu kernel I/O subsystem dan driver. Kernel I/O subsystem berperan untuk melakukan penjadwalan akses proses terhadap perangkat I/O, caching-buffering data antara perangkat I/O dengan memory dan CPU, penanganan error dari hardware, manajemen daya, dan proteksi I/O dari akses secara ilegal. Lapisan kernel I/O subsystem ini selanjutnya berkomunikasi dengan hardware melalui perantara driver. Dalam konteks ini, driver berperan untuk menyediakan abstraksi dan enkapsulasi terhadap perangkat keras I/O yang beragam. Abstraksi driver ini memudahkan pengembang sistem operasi maupun pabrikan perangkat keras dalam membuat perangkat keras baru agar dapat dikenali oleh sistem operasi. Pabrikan dapat mengikuti antar-muka driver terstandar yang sudah ada atau membuat driver perangkat keras baru untuk beberapa

sistem operasi populer. Komponen driver ini nantinya dapat menerjemahkan perintah dari kernel I/O subsystem menjadi instruksi spesifik yang dapat dipahami oleh device controller.



Gambar 9.1 Struktur Kernel I/O

9.3.3. Pengelolaan I/O pada Level Sistem Operasi dengan Kernel Module

Sistem operasi Linux mengimplementasikan driver untuk perangkat I/O dengan menggunakan kernel module. Kernel module di Linux adalah sebuah kode program yang dapat dimuat dan dimuat ulang ke dalam kernel Linux saat sistem berjalan. Kernel module memperluas fungsionalitas dan kemampuan kernel dengan menyediakan driver perangkat, sistem file, atau fitur lain yang tidak termasuk dalam kernel inti. Kernel module memungkinkan penambahan fungsionalitas ke dalam kernel Linux tanpa perlu mengompilasi ulang atau memuat ulang seluruh kernel. Modul-modul ini dapat dimuat saat diperlukan, dan jika tidak diperlukan lagi, mereka dapat dimuat ulang atau dibuang dari kernel.

Kernel module digunakan untuk berbagai tujuan, termasuk:

1. **Driver Perangkat**: Kernel module digunakan untuk mengimplementasikan driver perangkat yang berkomunikasi dengan perangkat keras, seperti keyboard, mouse, kartu jaringan, dan perangkat penyimpanan.

2. **Sistem File**: Kernel module memungkinkan penggunaan berbagai sistem file, seperti ext4, XFS, NTFS, yang memungkinkan akses dan pengelolaan file dan direktori.

3. **Pengelolaan Jaringan**: Modul jaringan di kernel digunakan untuk mengimplementasikan protokol jaringan dan menangani komunikasi jaringan antara sistem.

4. **Fungsionalitas Tambahan**: Kernel module juga dapat digunakan untuk menyediakan fungsionalitas tambahan, seperti pengoptimalan kinerja, pengaturan keamanan, pengelolaan daya, dan banyak lagi.

Kernel module Linux dapat dibangun dengan beberapa langkah berikut :

1. Pastikan bahwa paket untuk pengembangan kernel sudah terpasang

```
sudo apt-get install build-essential linux-headers-$(uname -r)
```

2. Buat kode kernel module dengan menggunakan bahasa C. Import library untuk pengembangan kernel module seperti " **linux/module.h**" dan "**linux/kernel.h**". Sebagai contoh, berikut adalah kernel module sederhana untuk mencetak informasi pada log file Linux.

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

static int __init hello_init(void)
{
    pr_info("Hello, Kernel!\n");
    return 0;
}

static void __exit hello_exit(void)
{
    pr_info("Goodbye, Kernel!\n");
}
```

```
        module_init(hello_init);
        module_exit(hello_exit);
        MODULE_LICENSE("GPL");
        MODULE_AUTHOR("Your Name");
       MODULE_DESCRIPTION("Hello Kernel Module");
```

3. Buat file bernama Makefile untuk proses kompilasi kode.

```
        obj-m += hello.o

        all:
            make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
    modules

        clean:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

4. Build kernel module dengan menjalankan perintah **`make`**. Kompilasi akan menghasilkan file bernama **hello.ko**.
5. Lakukan instalasi kernel module dengan perintah

```
sudo insmod hello.ko
```

6. Cek konten file /var/log/syslog untuk melihat output dari kernel module yang telah dipasang.
7. Kernel module dapat dihapus dengan perintah

```
sudo rmmod hello
```

Selain itu, untuk dapat mengakses fungsionalitas perangkat I/O secara programatik, sistem operasi menyediakan antar muka pemrograman yang disebut dengan system call. Sebagai sebuah antar muka pemrograman, system call dapat dipakai oleh pemrogram dengan

cara memanggil fungsi-fungsi yang tersedia dalam bahasa C. Beberapa contoh fungsi system call dapat dilihat pada tabel berikut.

Jenis System Call	Fungsi
Kontrol Proses	fork() exit() wait()
Manajemen File	open() read() write() close()
Manajemen Perangkat	ioctl() read() write()
Komunikasi	pipe() shm_open() mmap()

Berikut merupakan penjelasan detail dan contoh penerapan beberapa system call.

a. System Call **open()**

System call open() dipakai untuk membuka sebuah file dengan mode baca, tulis atau baca-tulis. Sintaks umum untuk mengakses fungsi open() adalah

```
int open (const char* Path, int flags [, int mode ]);
```

Dengan keterangan :

- Path: adalah lokasi file yang akan dibuka
- Flags: mekanisme akses file. Beberapa contoh flags antara lain: O_RDONLY: read only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: buat file jika belum ada, O_EXCL: tidak buat file ketika belum ada.

Contoh kode

```
1 #include <errno.h>
2 #include <fcntl.h>
```

```c
#include <stdio.h>

extern int errno;

int main()
{
    // if file does not have in directory
    // then file foo.txt is created.
    int fd = open("foo.txt", O_RDONLY | O_CREAT);

    printf("fd = %d/n", fd);

    if (fd == -1)
    {
        // print which type of error have in a code
        printf("Error Number % d\n", errno);

        // print program detail "Success or failure"
        perror("Program");
    }
    return 0;
}
```

b. System Call **read()**

System call read() dipakai untuk membaca isi dari sebuah file. Hasil pembacaan akan ditampung dalam memory pada sebuah variabel buffer.

```
size_t read(int fd, void* buf, size_t cnt);
```

Dengan keterangan :

1. Fd : file deskriptor
2. Buf : variabel sebagai buffer
3. Cnt : jumlah byte data yang dibaca

Contoh kode:

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

```

```c
int main()
{
    char c;
    int fd1 = open("sample.txt", O_RDONLY, 0);
    int fd2 = open("sample.txt", O_RDONLY, 0);
    read(fd1, &c, 1);
    read(fd2, &c, 1);
    printf("c = %c\n", c);
    exit(0);
}
```

c. System Call **write()**

System call write() dipakai untuk menulis pada sebuah file. Data yang akan ditulis ditampung pada sebuah variabel buffer.

```
size_t write (int fd, void* buf, size_t cnt);
```

Dengan keterangan:

1. Fd : file deskriptor
2. Buf : variabel sebagai buffer
3. Cnt : jumlah byte data yang akan ditulis

Contoh kode

```c
#include <fcntl.h>
#include <stdio.h>

main()
{
    int sz;

    int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0)
    {
        perror("r1");
        exit(1);
    }

    sz = write(fd, "hello geeks\n", strlen("hello geeks\n"));

    printf("called write(% d, \"hello geeks\\n\", %d)."
```

```
            " It returned %d\n",
            fd, strlen("hello geeks\n"), sz);

    close(fd);
}
```

d. System Call close()

System call close() dipakai untuk menutup sebuah file yang sudah dibuka. Sintaks umumnya adalah.

```
int close(int fd);
```

Dengan parameter fd yaitu file deskiptor.

9.4. Langkah Praktikum

```
    #include <linux/module.h>

    #include <linux/kernel.h>

    #include <linux/init.h>

    #include <linux/keyboard.h>


    static struct notifier_block keyboard_notifier;


    static int keyboard_event(struct notifier_block *nb, unsigned
long event, void *data)

    {

        struct keyboard_notifier_param *param = data;

        if (event == KBD_KEYCODE)

        {

            pr_info("Keyboard Event: keycode = %d, value = %d, down
= %d\n", param->value, param->down, param->down ? 1 : 0);

        }
```

```
        return NOTIFY_OK;
    }


    static int __init keyboard_io_init(void)
    {
        pr_info("Keyboard I/O Module: Initializing\n");
        keyboard_notifier.notifier_call = keyboard_event;
        register_keyboard_notifier(&keyboard_notifier);
        return 0;
    }


    static void __exit keyboard_io_exit(void)
    {
        pr_info("Keyboard I/O Module: Exiting\n");
        unregister_keyboard_notifier(&keyboard_notifier);
    }
    module_init(keyboard_io_init);
    module_exit(keyboard_io_exit);
    MODULE_LICENSE("GPL");
    MODULE_AUTHOR("Your Name");
    MODULE_DESCRIPTION("Keyboard I/O Module");
```

1. Lakukan kompilasi dan pemasangan kernel module di atas pada komputer Linux yang anda kelola! Dokumentasikan langkah pengerjaan anda dalam laporan.
2. Tugas apa saja yang dilakukan oleh kernel module tersebut dalam kaitannya dengan pengelolaan perangkat I/O? Buktikan dengan menganalisis output file log sistem.

3. Jelaskan apa saja yang dilakukan oleh kode kernel module tersebut!
4. (**Soal Bonus**) Dengan memanfaatkan system call di atas, modifikasi kernel module sehingga bisa menyimpan kata-kata tertentu (bisa didefinisikan sendiri) ke dalam sebuah file.

9.5. Kesimpulan

Jelaskan kesimpulan dari hasil percobaan pada Bab 9 ini

UNTUK DIPERHATIKAN

Setiap selesai melakukan praktikum, jangan lupa menonaktfikan instance agar billing kuota tidak terus berjalan. Lakukan dengan cara memilih button Action pada Instance Summary. Pilih Stop sebagai perintah yang harus dijalankan.