LAPORAN PRAKTIKUM
SISTEM OPERASI RD
MODUL 3

Oleh :

Rayhan Fadel Irwanto (122140236)

Program Studi Teknik Informatika

Institut Teknologi Sumatera

2024

Daftar Isi

1. Dasar Teori

System call adalah suatu metode yang digunakan oleh program komputer untuk meminta layanan dari kernel sistem operasi tempat program tersebut berjalan. Hal ini dilakukan melalui antarmuka yang telah terdefinisi, biasanya berupa API, yang menyediakan interaksi antara program pengguna dan sistem operasi. Fungsi utama dari system call mencakup berbagai aspek, mulai dari memberikan interface bagi program untuk membuat permintaan layanan tertentu, hingga melindungi sistem dari akses yang tidak sah dengan hak istimewa, serta mengalihkan program ke mode kernel untuk mengakses sumber daya sistem.

Jenis-jenis system call meliputi kontrol proses, manajemen file, pengelolaan perangkat keras, pemeliharaan informasi sistem, komunikasi antar proses dan jaringan, serta manajemen memori. Dalam penggunaannya, system call juga berperan dalam penanganan kesalahan, menyinkronkan akses ke sumber daya bersama, dan menghadapi biaya tambahan akibat pengalihan konteks yang diperlukan saat program berpindah dari mode pengguna ke mode kernel untuk melakukan operasi tertentu.

2. Hasil & Jawaban

Percobaan 1 Open

1. Membuat folder “Data” dan mengakses isi folder tersebut

```
Vbox@rehan:~$ mkdir data
Vbox@rehan:~$ cd data
```

2. Membuat file dengan nama “identitas.txt” dengan isi Nama, Nim, Kelas

```
Vbox@rehan:~/data$ nano identitas.txt
Vbox@rehan:~/data$ cat identitas.txt
Rayhan Fadel Irwanto
122140236
RD
```

3. Membuaf file dengan ektensi C dengan nama “open.c”

```
Vbox@rehan:~/data$ nano open.c
Vbox@rehan:~/data$ cat open.c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main(){
int fd;

fd=open("identitas.txt", O_RDWR);

if(fd==-1){
printf("file tidak ditemukan.\n");
return 1;
}

printf("file berhasil dibuka.\n");
close(fd);

return 0;
}
```

4. Lakukan instalasi gcc dengan perintah “sudo apt install gcc” dan jalankan perintah “gcc open.c -o openfile”

Sudah Install Sebelumnya

5. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./openfile”. Jika file berhasil dibuka maka akan menampilkan luaran berikut

```
Vbox@rehan:~/data$ gcc open.c -o openfile
Vbox@rehan:~/data$ ./openfile
file berhasil dibuka.
```

Perobaan 2 Close

1. Mengakses folder “Data” dan membuaf file dengan ektensi C dengan nama “close.c”

```
Vbox@rehan:~/data$ nano close.c
Vbox@rehan:~/data$ cat close.c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main(){
int fd;
fd=open("identitas.txt", O_RDONLY);

if(fd==-1){
printf("file tidak ditemukan.\n");
return 1;
}

printf("file berhasil dibuka.\n");

close(fd);
printf("file berhasil ditutup.\n");

return 0;
}
```

2. Jalankan perintah “gcc close.c -o closefile”

```
Vbox@rehan:~/data$ gcc close.c -o closefile
Vbox@rehan:~/data$ ./closefile
file berhasil dibuka.
file berhasil ditutup.
```

3. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./closefile”

```
Vbox@rehan:~/data$ gcc close.c -o closefile
Vbox@rehan:~/data$ ./closefile
file berhasil dibuka.
file berhasil ditutup.
```

Percobaan 3 Write

1. Mengakses folder “Data” dan membuaf file dengan ektensi C dengan nama “write.c”

```
Vbox@rehan:~/data$ nano write.c
Vbox@rehan:~/data$ cat write.c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(){
int fd;
char buf[1024];

fd=open("contoh.txt", O_CREAT | O_WRONLY | O_TRUNC);

if(fd==-1){
printf("error ketika membuka file.\n");
exit(EXIT_FAILURE);
}

strcpy(buf, "Saya sedang melakukan praktikum mata kuliah sistem operasi.\n");
write(fd, buf, strlen(buf));

printf("file berhasil ditulis.\n");
close(fd);

return 0;
}
```

2. Jalankan perintah “gcc write.c -o writefile”.

```
Vbox@rehan:~/data$ gcc write.c -o writefile
Vbox@rehan:~/data$ ./writefile
file berhasil ditulis.
Vbox@rehan:~/data$ ls
close.c  closefile  contoh.txt  identitas.txt  open.c  openfile  write.c  writefile
Vbox@rehan:~/data$ cat contoh.txt
cat: contoh.txt: Permission denied
```

3. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./writefile”. Jika file berhasil ditulis maka akan menghasilkan file “contoh.txt”

```
Vbox@rehan:~/data$ gcc write.c -o writefile
Vbox@rehan:~/data$ ./writefile
file berhasil ditulis.
Vbox@rehan:~/data$ ls
close.c  closefile  contoh.txt  identitas.txt  open.c  openfile  write.c  writefile
Vbox@rehan:~/data$ cat contoh.txt
cat: contoh.txt: Permission denied
```

Percobaan 4 Delete

1. Mengakses folder “Data” dan membuaf file dengan ektensi C dengan nama “delete.c”

```
Vbox@rehan:~/data$ nano delete.c
Vbox@rehan:~/data$ cat delete.c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

int main(){

char filename[]="contoh.txt";
int result=unlink(filename);

if(result==0){
printf("file %s berhasil dihapus. \n", filename);
}else{
printf("gagal menghapus file %s. \n", filename);
}

return 0;
}
```

2. Jalankan perintah “gcc delete.c -o deletefile”

```
Vbox@rehan:~/data$ gcc delete.c -o deletefile
Vbox@rehan:~/data$ ./deletefile
file contoh.txt berhasil dihapus.
```

3. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./deletefile”. Jika file berhasil dihapus maka akan menghasilkan luaran

```
Vbox@rehan:~/data$ gcc delete.c -o deletefile
Vbox@rehan:~/data$ ./deletefile
file contoh.txt berhasil dihapus.
```

Percobaan 5 Fork

1. Mengakses folder “Data1” dan membuaf file dengan ektensi C dengan nama “fork.c”

```
Vbox@rehan:~$ mkdir data1
Vbox@rehan:~$ cd data1
Vbox@rehan:~/data1$ nano fork.c
Vbox@rehan:~/data1$ cat fork.c
#include <stdio.h>
#include <unistd.h>

int main(){
pid_t pid;
pid=fork();

if(pid==1){
printf("Fork gagal");
}else if(pid==0){
printf("Ini adalah proses child. \n");
}else{
printf("Ini adalah proses parent. \n");
}

return 0;
}
```

2. Jalankan perintah “gcc fork.c -o ForkTes”

```
Vbox@rehan:~/data1$ gcc fork.c -o forkTes
Vbox@rehan:~/data1$ ./forkTes
Ini adalah proses parent.
Ini adalah proses child.
```

3. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./Fork.Test”. Jika file berhasil dihapus maka akan menghasilkan luaran

```
Vbox@rehan:~/data1$ gcc fork.c -o forkTes
Vbox@rehan:~/data1$ ./forkTes
Ini adalah proses parent.
Ini adalah proses child.
```

Percobaan 6 Wait

1. Mengakses folder “Data1” dan membuaf file dengan ektensi C dengan nama “wait.c”

```
Vbox@rehan:~/data1$ nano wait.c
Vbox@rehan:~/data1$ cat wait.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(){
pid_t pid;
int status;

pid=fork();

if(pid==1){
printf("Fork gagal");
}else if(pid==0){
printf("Ini adalah proses child. \n");
exit(0);
}else{
wait(&status);
printf("Ini adalah proses parent. \n");
}

return 0;
}
```

2. Jalankan perintah “gcc wait.c -o WaitTes”

```
Vbox@rehan:~/data1$ gcc wait.c -o waitTes
Vbox@rehan:~/data1$ ./waitTes
Ini adalah proses child.
Ini adalah proses parent.
```

3. Lakukan pemanggilan sistem yang telah kita buat dengan perintah “./Wait.Test”. Jika file berhasil dihapus maka akan menghasilkan luaran

```
Vbox@rehan:~/data1$ gcc wait.c -o waitTes
Vbox@rehan:~/data1$ ./waitTes
Ini adalah proses child.
Ini adalah proses parent.
```

3. Kesimpulan dan Saran

Pemahaman tentang System Call mengungkapkan bahwa System Call berperan sebagai jembatan penting yang menghubungkan program pengguna dengan kernel sistem operasi, memungkinkan program untuk meminta layanan seperti akses file, manajemen proses, pengelolaan perangkat keras, dan komunikasi antar proses. Dengan antarmuka yang terdefinisi dengan baik, interaksi antara program dan sistem operasi dapat dilakukan secara efisien dan aman. Namun, dalam penggunaannya, perlu diperhatikan aspek perlindungan data, hak istimewa, pengelolaan Context Switching untuk mengoptimalkan kinerja sistem, dan penanganan kesalahan agar program dapat merespons kondisi yang tidak diinginkan dengan tepat. Dengan pemahaman yang matang tentang System Call, program dapat berjalan lebih efisien dan dapat diandalkan dalam mengakses sumber daya sistem.