OS Lab 8

Tazmeen Afroz

1 Redirecting Output to Another Terminal

To send the output of a command from one terminal to another, you can use the following command:

```
echo "Hello" > /proc/X/fd/1
```

Here, X is the process ID of the target terminal.

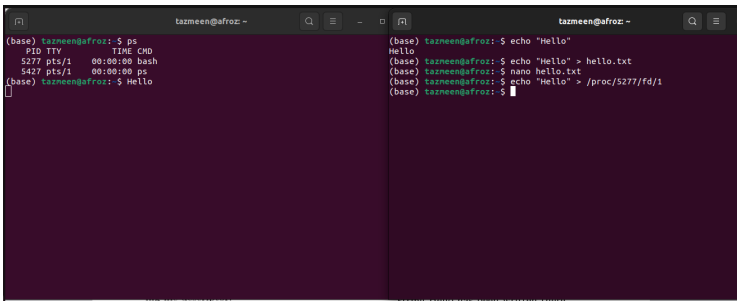

Figure 1: X Process Example

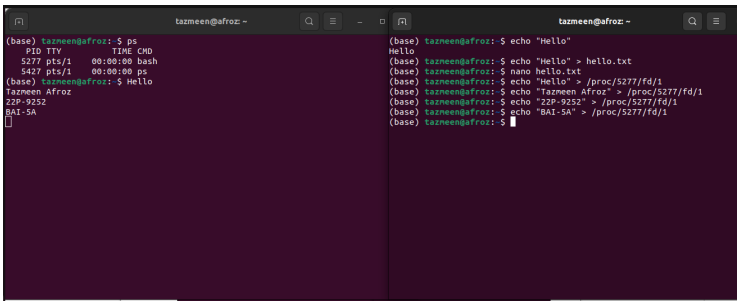

Figure 2: Output after redirecting

2 List of All Processes

Navigate to the `/proc/` directory using the following command:

```
cd /proc/
```

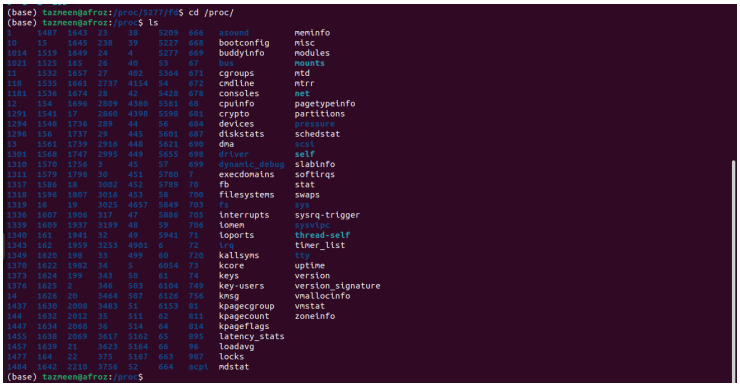

Figure 3: List of processes in the /proc/ directory

3 Navigating to a Specific Process

Go to the directory of a specific process by using the following command:

```
cd /proc/5277/
```

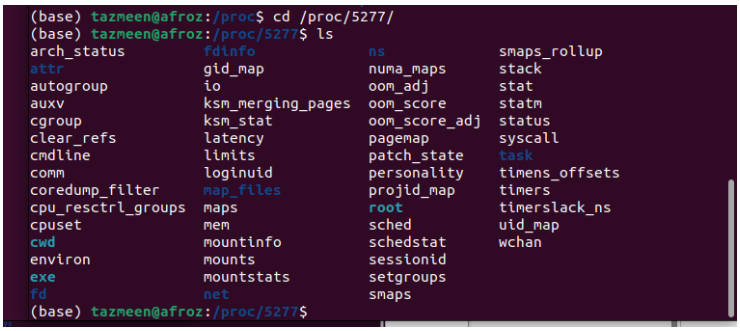

Figure 4: Navigating to Process 5277

4 File Descriptor Directory of a Process

Navigate to the `fd` (File Descriptor) directory of the process:

```
cd fd
```

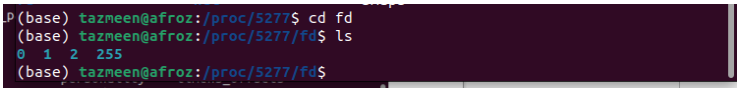

Figure 5: Navigating to fd directory of the process

```
ls
```

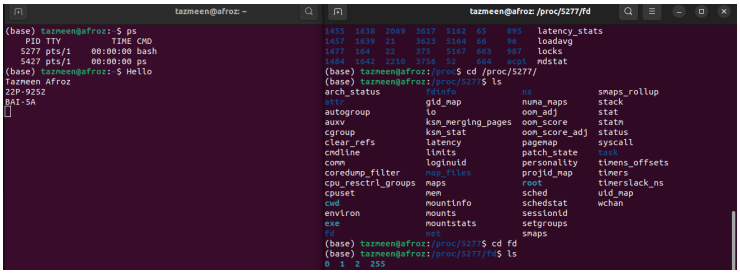

Figure 6: Listing file descriptors in the fd directory

5 C Code for File Creation

Execute the following C code to create a file:

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    char *path = argv[1];
    int fd = open(path, O_WRONLY | O_EXCL | O_CREAT);
    if (fd == -1)
    {
        printf("Error - File not Created\n");
        return 1;
    }
```

```
    return 0;
}
```

Compile and run the program:

```
gcc first.c -o first
./first createThisFile
```

Figure 7: Output after running the file creation code

6 File Size Question

Question: What is the size of the file created?

Answer: The file size is 0 bytes because no data was written to it, only created.

7 Understanding the `open()` System Call

The `open()` system call returns a file descriptor or an error code:

- **Success:** A non-negative integer representing the file descriptor.
- **Failure:** -1, with the `errno` variable set to indicate the specific error.

Common reasons for failure include:

- File doesn't exist
- Permission denied
- Too many open files

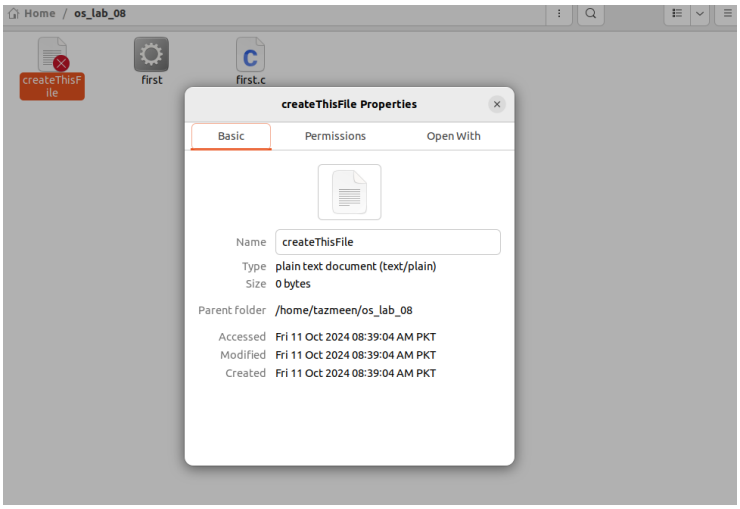

Figure 8: File size is 0 bytes

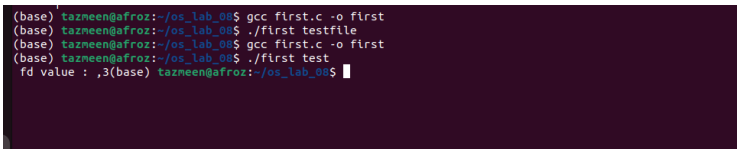

Figure 9: fd return value ,3 used for normal creation of file

8 Command-Line Arguments: `argc` and `argv[]`

When a C program is executed, it can take input from the command line. These inputs are passed to the program through the parameters `argc` (Argument Count) and `argv[]` (Argument Vector).

8.1 `argc` (Argument Count)

- **Type:** `int`
- **Purpose:** It represents the number of command-line arguments passed to the program.
- **Value:** `argc` is always at least 1, because the program name itself is considered the first argument.

8.2 `argv[]` (Argument Vector)

- **Type:** `char* argv[]` or `char** argv`

- **Purpose:** It's an array of strings containing the actual command-line arguments.
- **Contents:**
  - `argv[0]` is always the name of the program.
  - `argv[1]`, `argv[2]`, etc., are the additional arguments provided on the command line.

Code Snippet

Listing 1: C Program

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("Error: Run like this:");
        printf("%6s name-of-new-file\n", argv[0]);
        return 1;
    }
    char *path = argv[1];
    int i = 0;
    while(i<2)
    {
        int fd = open(path, O_WRONLY | O_CREAT);
        printf("Created! Descriptor is %d\n", fd);
        close(fd);  // Comment this line later for the test
        i++;
    }
    return 0;
}
```

Test: Comment Out close(fd)

Comment out the line close(fd); and then compile and run the program again.

Before Commenting

```
Created! Descriptor is 3
Created! Descriptor is 3
```

After Commenting

```
Created! Descriptor is 3
Created! Descriptor is 4
```

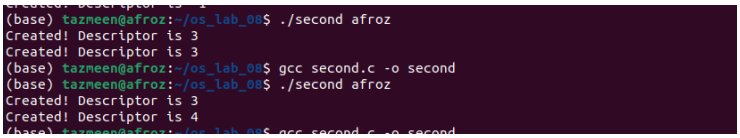

Figure 10: Output

Explanation of Output

When the close(fd); line is commented out, the file descriptor is not released after each file operation. As a result, the second call to open() assigns the next available file descriptor (which is 4, since descriptor 3 is still in use).

When the line is not commented out, the file descriptor is closed after each iteration, and the OS reuses the same descriptor (3) for both operations.

Code Snippet

Listing 2: C Program

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

char* get_timeStamp()
{
    time_t now = time(NULL);
    return asctime(localtime(&now));
}
```

```c
int main(int argc, char* argv[])
{
    char *filename = argv[1];
    char *timeStamp = get_timeStamp();
    int fd = open(filename, O_WRONLY | O_APPEND | O_CREAT, 0666);
    size_t length = strlen(timeStamp)-5;
    write(fd, timeStamp, length);
    close(fd);
    return 0;
}
```

Questions and Answers

Q1: What is 0666 that is specified in the open() call? What does it mean?

A1: The number 0666 represents the file permission mode. It means that the file will be created with read and write permissions for the owner, group, and others. In symbolic notation, it corresponds to rw—rw—rw—.

Q2: What is O_APPEND doing in the same call?

A2: The flag O_APPEND ensures that data is always written at the end of the file, without modifying the existing content. When this flag is used, every write operation appends the data to the current end of the file.

```
1 Fri Oct 11 09:51:47 2024
2 Fri Oct 11 09:58:35 2024
3 Fri Oct 11 09:58:39 2024
4 Fri Oct 11 10:03:27 2024
```

Figure 11: Output

Q3: Modify the following line in the code and then compile and run the program and check its output. What has happened?

textbfA3: By reducing the length by 5, you're truncating the last 5 characters of the timestamp.

Code Snippet

Listing 3: C Program

```c
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("Error: Run like this: ");
        printf("%6s name-of-existing-file\n",argv[0]);
        return 1;
    }
    char *path = argv[1];
    int fd = open(path, O_RDONLY);
    if (fd == -1)
    {
        printf("File does not exist\n");
        return 1;
    }
    char buffer[200];
    read(fd, buffer, sizeof(buffer)-1);
    printf("Contents of File are:\n");
    printf("%s\n", buffer);
    close(fd);
    return 0;
}
```

Explanation of Output

The program opens a file specified by the user as a command-line argument, reads its contents, and prints them to the console.

Case 1: Invalid Input

If the user does not provide the correct argument (file name), the program displays the error:

```
Error: Run like this: ./program_name name-of-existing-file
```

Case 2: File Not Found

If the file does not exist, the output will be:

```
File does not exist
```

Case 3: File Found

When the file exists, the program will read and display its contents. The output will be:

```
Contents of File are:
<file contents here>
```

Output Image

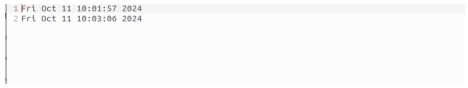

Figure 12: Program Output Displaying File Contents