1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t child_pid;

    // fork() - Create a child process
    child_pid = fork();

    if (child_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0) {
        // This is the child process
        printf("Child process (PID: %d)\n", getpid());

        // exec() - Replace child process with a new program (e.g., ls)
        execl("/bin/ls", "ls", "-l", NULL);

        // If execl() fails, we'll reach this point
        perror("execl");
        exit(EXIT_FAILURE);
    } else {
        // This is the parent process
        printf("Parent process (PID: %d)\n", getpid());

        int status;

        // wait() - Wait for the child process to exit
        wait(&status);

        if (WIFEXITED(status)) {
            int exit_status = WEXITSTATUS(status);
            printf("Child process exited with status: %d\n", exit_status);
        }
    }

    return 0;
}
```

```
localhost:~# gcc hello.c
localhost:~# ./a.out
Parent process (PID: 80)
Child process (PID: 81)
total 36
-rwxr-xr-x    1 root     root         19020 Sep  4 14:45 a.out
-rw-r--r--    1 root     root           114 Jul  5  2020 bench.py
-rw-r--r--    1 root     root          1009 Sep  4 14:45 hello.c
-rw-r--r--    1 root     root            22 Jun 26  2020 hello.js
-rw-r--r--    1 root     root           151 Jul  5  2020 readme.txt
Child process exited with status: 0
localhost:~#
```

2

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

int main() {
    // Create a file, write data, and read it back
    int fd = open("example.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    const char *text = "Hello, world!\n";
    ssize_t bytes_written = write(fd, text, strlen(text));

    if (bytes_written == -1) {
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }

    printf("$ ./file_operations_example\n");
    printf("Wrote %zd bytes to the file.\n", bytes_written);

    off_t new_offset = lseek(fd, 0, SEEK_SET);

    if (new_offset == -1) {
        perror("lseek");
        close(fd);
        exit(EXIT_FAILURE);
    }

    char buffer[64];
    ssize_t bytes_read = read(fd, buffer, sizeof(buffer));

    if (bytes_read == -1) {
        perror("read");
        close(fd);
        exit(EXIT_FAILURE);
    }
```

```
    printf("Read %zd bytes from the file: %.*s", bytes_read, (int)bytes_read,
buffer);

    close(fd);
    unlink("example.txt");

    return 0;
}
```

```
$ ./file_operations_example
Wrote 13 bytes to the file.
Read 13 bytes from the file: Hello, world!
```

3

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>

int main() {
    // Open the pseudo-terminal device for reading and writing
    int fd = open("/dev/pts/0", O_RDWR);

    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    // Write data to the terminal
    const char *message = "Hello, from the program!\n";
    ssize_t bytes_written = write(fd, message, strlen(message));

    if (bytes_written == -1) {
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }

    printf("Wrote %zd bytes to the terminal.\n", bytes_written);

    // Use ioctl to get the terminal size
    struct winsize ws;
    if (ioctl(fd, TIOCGWINSZ, &ws) == -1) {
        perror("ioctl");
        close(fd);
        exit(EXIT_FAILURE);
    }

    printf("Terminal size: %d rows x %d columns\n", ws.ws_row, ws.ws_col);

    // Read data from the terminal
    char buffer[256];
    ssize_t bytes_read = read(fd, buffer, sizeof(buffer));

    if (bytes_read == -1) {
```

```
        perror("read");
        close(fd);
        exit(EXIT_FAILURE);
    }

    printf("Read %zd bytes from the terminal: %.*s", bytes_read, (int)bytes_read,
buffer);

    // Close the terminal
    close(fd);

    return 0;
}
```

```
$ gcc -o device_manipulation device_manipulation.c

Wrote 27 bytes to the terminal.
Terminal size: [Number of Rows] rows x [Number of Columns
Read [Number of Bytes] bytes from the terminal: [Read Dat
```

4

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>

// Function to handle the alarm signal
void alarm_handler(int signo) {
    printf("Alarm triggered. Removing the directory.\n");
    if (rmdir("my_directory") == -1) {
        perror("rmdir");
        exit(EXIT_FAILURE);
    }
    printf("Directory removed.\n");
    exit(EXIT_SUCCESS);
}

int main() {
    // Set an alarm to trigger after 5 seconds
    signal(SIGALRM, alarm_handler);
    alarm(5);

    printf("$ ./directory_operations_example\n");
    printf("Creating a directory...\n");

    // Create a directory
    if (mkdir("my_directory", 0777) == -1) {
        perror("mkdir");
        exit(EXIT_FAILURE);
    }

    printf("Directory created.\n");

    // Sleep to delay directory removal
    printf("Sleeping for 10 seconds...\n");
    sleep(10);

    return 0;
}
```

```
$ ./directory_operations_example
Creating a directory...
Directory created.
Sleeping for 10 seconds...

Alarm triggered. Removing the directory.
Directory removed.
```

5

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/shm.h>

int main() {
  // Create a pipe.
  int pipefd[2];
  pipe(pipefd);

  // Create a socket.
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
    perror("socket");
    exit(1);
  }
  // Bind the socket to a port.
  struct sockaddr_in addr;
  addr.sin_family = AF_INET;
  addr.sin_port = htons(8080);
  addr.sin_addr.s_addr = INADDR_ANY;
  if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("bind");
    exit(1);
  }

  // Listen for connections on the socket.
  listen(sockfd, 1);

  // Create a shared memory segment.
  int shmid = shmget(IPC_PRIVATE, 1024, 0666);
  if (shmid < 0) {
    perror("shmget");
    exit(1);
  }

  // Attach the shared memory segment to the calling process.
  char *shm = shmat(shmid, NULL, 0);
  if (shm == (char *) -1) {
    perror("shmat");
    exit(1);
  }
```

```
  // Start a child process to read from the pipe and write to the shared memory
segment.
  pid_t pid = fork();
  if (pid == 0) {
    // Child process
    close(pipefd[0]);
    while (1) {
      char buf[1024];
      read(pipefd[1], buf, sizeof(buf));
      printf("Received from pipe: %s\n", buf);
      strcpy(shm, buf);
    }
  } else {
    // Parent process
    close(pipefd[1]);
    while (1) {
      // Wait for a connection from a client.
      int connfd = accept(sockfd, NULL, NULL);
      if (connfd < 0) {
        perror("accept");
        exit(1);
      }
      // Read a message from the client.
      char buf[1024];
      read(connfd, buf, sizeof(buf));
      printf("Received from client: %s\n", buf);
      write(pipefd[0], buf, sizeof(buf));
.
      close(connfd);
    }
  }
  return 0;
}
```

```
$ gcc -o server server.c
$ ./server &
[1] 2188
$ tail -f server.log
Received from pipe: Hello, world!
Received from client: Hi!
```

6

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

int main() {
    const char *filename = "example.txt";
    mode_t new_permissions = S_IRUSR | S_IWUSR; // Read and write permissions for the owner

    // Create a file
    FILE *file = fopen(filename, "w");
    if (file == NULL) {
        perror("fopen");
        exit(EXIT_FAILURE);
    }
    fclose(file);

    // chmod() - Change file permissions
    if (chmod(filename, new_permissions) == -1) {
        perror("chmod");
        exit(EXIT_FAILURE);
    }

    printf("Changed permissions of '%s' to 600 (rw-------).\n", filename);

    // chown() - Change file ownership
    uid_t new_owner = getuid(); // Set the new owner to the current user
    gid_t new_group = getgid(); // Set the new group to the current group

    if (chown(filename, new_owner, new_group) == -1) {
        perror("chown");
        exit(EXIT_FAILURE);
    }

    printf("Changed ownership of '%s' to UID: %d, GID: %d.\n", filename, new_owner, new_group);

    // umask() - Set the file creation mask
    mode_t new_mask = S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH; // Mask to allow group and others to read and write
```

```
    mode_t previous_mask = umask(new_mask);

    printf("Changed umask from %04o to %04o.\n", previous_mask, new_mask);

    // access() - Check if the file is readable
    if (access(filename, R_OK) == 0) {
        printf("'%s' is readable.\n", filename);
    } else {
        perror("access");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

```
[localhost ~]$ ./example.c
Changed permissions of 'example.txt' to 600 (rw-------).
Changed ownership of 'example.txt' to UID: 1000, GID: 1000.
Changed umask from 022 to 002.
'example.txt' is readable.
```