1. Print the type of file and inode number (Slip 3, 19, 24)

Question:
Write a C program to print the type of a file and its inode number, where the file name is passed through command line arguments.
Example Slips: 3, 19, 24

Answer:

```
c
Copy code
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    struct stat fileStat;

    // Check if file name is provided as command line argument
    if (argc != 2) {
        printf("Usage: %s <file_name>\n", argv[0]);
        return 1;
    }

    // Get file status
    if (stat(argv[1], &fileStat) < 0) {
        perror("stat error");
        return 1;
    }

    // Print inode number
    printf("Inode number: %ld\n", fileStat.st_ino);

    // Print file type
    if (S_ISREG(fileStat.st_mode))
        printf("File type: Regular file\n");
    else if (S_ISDIR(fileStat.st_mode))
        printf("File type: Directory\n");
    else if (S_ISLNK(fileStat.st_mode))
        printf("File type: Symbolic link\n");
    else
        printf("File type: Other\n");

    return 0;
}
```

Explanation:

1. `stat()` is used to retrieve information about the file passed as a command-line argument.
2. The inode number is printed using `fileStat.st_ino`.
3. The file type is determined using macros like `S_ISREG()`, `S_ISDIR()`, and `S_ISLNK()`.

---

2. Check if file is present in the current directory (Slip 4, 18, 23)

Question:
Write a C program to find whether a given file is present in the current directory or not.
Example Slips: 4, 18, 23

Answer:

```
c
Copy code
#include <stdio.h>
#include <dirent.h>

int main() {
    struct dirent *entry;
    DIR *dir = opendir(".");

    if (dir == NULL) {
        perror("opendir failed");
        return 1;
    }

    // Ask user for file name
    char filename[100];
    printf("Enter file name to search: ");
    scanf("%s", filename);

    int found = 0;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, filename) == 0) {
            printf("File %s found in the current directory.\n", filename);
            found = 1;
            break;
        }
    }

    if (!found)
        printf("File %s not found in the current directory.\n", filename);

    closedir(dir);
    return 0;
}
```

Explanation:

1. The `opendir()` function is used to open the current directory (`"."`).
2. The program iterates through the directory using `readdir()` and compares each entry with the filename entered by the user.
3. If a match is found, it prints that the file is present; otherwise, it shows a not found message.

---

3. Read the current directory and display file names (Slip 5, 17, 21)

Question:
Write a C program to read the current directory and display the names of the files, as well as the total number of files.
Example Slips: 5, 17, 21

Answer:

```
c
Copy code
#include <stdio.h>
#include <dirent.h>

int main() {
    struct dirent *entry;
    DIR *dir = opendir(".");

    if (dir == NULL) {
        perror("opendir failed");
        return 1;
    }

    int fileCount = 0;
    printf("Files in the current directory:\n");

    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
        fileCount++;
    }

    printf("Total number of files: %d\n", fileCount);

    closedir(dir);
    return 0;
}
```

Explanation:

1. The program uses `opendir()` to open the current directory and `readdir()` to read each file name.
2. Each file name is printed, and the `fileCount` is incremented.
3. After the loop finishes, the total number of files is displayed.

---

4. Display files by creation month (Slip 6, 16)

Question:
Write a C program to display all the files from the current directory that were created in a particular month.
Example Slips: 6, 16

Answer:

```
c
Copy code
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>

int main() {
    struct dirent *entry;
    DIR *dir = opendir(".");
    struct stat fileStat;

    if (dir == NULL) {
        perror("opendir failed");
        return 1;
    }

    char month[20];
    printf("Enter the month (e.g., Jan, Feb, Mar): ");
    scanf("%s", month);

    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) < 0) {
            perror("stat error");
            continue;
        }

        // Get the month from file creation time
        struct tm *timeInfo = localtime(&fileStat.st_ctime);
        char fileMonth[4];
        strftime(fileMonth, sizeof(fileMonth), "%b", timeInfo);

        if (strcmp(fileMonth, month) == 0) {
            printf("%s was created in %s.\n", entry->d_name, month);
        }
    }

    closedir(dir);
    return 0;
}
```

Explanation:

1. The `stat()` function is used to get the file information, including the creation time (`st_ctime`).
2. The `localtime()` function converts the creation time to a `tm` structure.
3. The month is extracted from the `tm` structure and compared with the user-provided month.

---

5. Display files larger than n bytes (Slip 14, 15)

Question:
Write a C program to display all the files from the current directory whose size is greater than

n bytes (where n is accepted from the user).
Example Slips: 14, 15

Answer:

```
c
Copy code
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>

int main() {
    struct dirent *entry;
    DIR *dir = opendir(".");
    struct stat fileStat;

    if (dir == NULL) {
        perror("opendir failed");
        return 1;
    }

    int sizeLimit;
    printf("Enter the size limit (in bytes): ");
    scanf("%d", &sizeLimit);

    while ((entry = readdir(dir)) != NULL) {
        if (stat(entry->d_name, &fileStat) < 0) {
            perror("stat error");
            continue;
        }

        if (fileStat.st_size > sizeLimit) {
            printf("%s is larger than %d bytes.\n", entry->d_name,
sizeLimit);
        }
    }

    closedir(dir);
    return 0;
}
```

Explanation:

1. The program uses `stat()` to get file information, specifically the size (`st_size`).
2. It compares each file's size with the user-provided limit (`n`), and if it exceeds the limit, it is printed.

---

6. Redirection of standard output to a file (Slip 7, 22, 25)

Question:
Write a C program to demonstrate the redirection of standard output to a file.
Example Slips: 7, 22, 25

Answer:

```
c
Copy code
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        perror("open failed");
        return 1;
    }

    // Redirect stdout to the file
    if (dup2(fd, STDOUT_FILENO) == -1) {
        perror("dup2 failed");
        return 1;
    }

    // Now, printf will write to output.txt instead of the terminal
    printf("This will be written to output.txt\n");

    close(fd);
    return 0;
}
```

Explanation:

1. The `open()` function opens the file `output.txt` for writing.
2. `dup2()` is used to redirect standard output (`STDOUT_FILENO`) to the file.
3. All `printf()` outputs are now redirected to `output.txt`.

---

7. File properties using stat() (Slip 2, 9, 14)

Question:
Write a C program to find file properties like inode number, number of hard links, file permissions, file size, access and modification times, etc., using the stat() system call.
Example Slips: 2, 9, 14

Answer:

```
c
Copy code
#include <stdio.h>
#include <sys/stat.h>
#include <time.h>

int main() {
    struct stat fileStat;

    if (stat("testfile.txt", &fileStat) < 0) {
        perror("stat failed");
        return 1;
    }
```

```
    printf("Inode number: %ld\n", fileStat.st_ino);
    printf("Number of hard links: %ld\n", fileStat.st_nlink);
    printf("File size: %ld bytes\n", fileStat.st_size);
    printf("File permissions: ");
    printf((fileStat.st_mode & S_IRUSR) ? "r" : "-");
    printf((fileStat.st_mode & S_IWUSR) ? "w" : "-");
    printf((fileStat.st_mode & S_IXUSR) ? "x" : "-");
    printf("\n");

    printf("Last access time: %s", ctime(&fileStat.st_atime));
    printf("Last modification time: %s", ctime(&fileStat.st_mtime));
    printf("Last status change time: %s", ctime(&fileStat.st_ctime));

    return 0;
}
```

Explanation:

1. The `stat()` function is used to gather information about the file.
2. The program prints the inode number, number of hard links, file size, and permissions.
3. It also prints the last access, modification, and status change times using `ctime()`.

8. Get and set resource limits (Slip 11)

Question:
Write a C program to get and set resource limits, such as files or memory associated with a process.
Example Slips: 11

Answer:

```
c
Copy code
#include <stdio.h>
#include <sys/resource.h>

int main() {
    struct rlimit rl;

    // Get the current limit for the number of open files
    if (getrlimit(RLIMIT_NOFILE, &rl) == 0) {
        printf("Current file limit: soft = %ld, hard = %ld\n", rl.rlim_cur,
rl.rlim_max);
    } else {
        perror("getrlimit failed");
        return 1;
    }

    // Set the soft limit to 100
    rl.rlim_cur = 100;
    if (setrlimit(RLIMIT_NOFILE, &rl) == 0) {
```

```
        printf("Successfully set new file limit.\n");
    } else {
        perror("setrlimit failed");
        return 1;
    }

    return 0;
}
```

Explanation:

1. The `getrlimit()` function retrieves the current limits of a system resource (in this case, the number of open files).
2. The `setrlimit()` function sets a new soft limit (the current limit) for the resource. In this example, we set the soft limit for the number of open files to 100.
3. The program prints the current and updated resource limits.

---

9. Exit status of a child process (Slip 12)

Question:
Write a C program to print the exit status of a terminated child process.
Example Slips: 12

Answer:

```
c
Copy code
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

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

    if (pid == 0) {
        // Child process
        printf("Child process exiting with status 42.\n");
        _exit(42); // Exit with status 42
    } else {
        // Parent process
        int status;
        wait(&status); // Wait for the child process to terminate

        if (WIFEXITED(status)) {
            printf("Child exited with status: %d\n", WEXITSTATUS(status));
        } else {
            printf("Child did not exit normally.\n");
        }
    }

    return 0;
}
```

Explanation:

1. The `fork()` system call creates a new child process. In the child process, we use `_exit()` to terminate with a specific status code (42).
2. The parent process waits for the child to finish using `wait()`. The exit status of the child is checked with `WIFEXITED()` and `WEXITSTATUS()`.
3. The program prints the exit status of the child process.

---

10. Suspend and resume processes using signals (Slip 13, 20)

Question:
Write a C program to illustrate suspending and resuming processes using signals.
Example Slips: 13, 20

Answer:

```
c
Copy code
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

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

    if (pid == 0) {
        // Child process
        printf("Child process started. Pausing...\n");
        pause();  // Wait for a signal to resume
        printf("Child process resumed.\n");
    } else {
        // Parent process
        sleep(2); // Give child time to start and pause
        printf("Parent process sending SIGCONT signal to child.\n");
        kill(pid, SIGCONT);  // Send signal to resume the child
        sleep(2);  // Allow child to resume
        kill(pid, SIGSTOP);  // Send SIGSTOP signal to suspend child
        printf("Parent process sent SIGSTOP signal to child.\n");
    }

    return 0;
}
```

Explanation:

1. The program creates a child process using `fork()`.
2. In the child process, we use `pause()` to suspend the process until it receives a signal.
3. The parent process sends the `SIGCONT` signal to resume the child process and the `SIGSTOP` signal to suspend it.
4. The program demonstrates suspending and resuming processes using signals.

1. ls -l | wc -l using fork, pipe, and exec (Slip 7, 8, 19, 22)

Question:
Write a C program to implement the Unix/Linux command `ls -l | wc -l` using `fork`, `pipe`, and `exec` system calls.
Example Slips: 7, 8, 19, 22

Answer:

```
c
Copy code
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int pipefd[2];
    pid_t pid1, pid2;

    if (pipe(pipefd) == -1) {
        perror("pipe");
        return 1;
    }

    pid1 = fork();
    if (pid1 == 0) {
        // Child process 1 (ls -l)
        close(pipefd[0]); // Close read end
        dup2(pipefd[1], STDOUT_FILENO); // Redirect stdout to pipe write
end
        close(pipefd[1]);

        execlp("ls", "ls", "-l", (char *)NULL); // Execute ls -l
        perror("execlp"); // If execlp fails
        return 1;
    }

    pid2 = fork();
    if (pid2 == 0) {
        // Child process 2 (wc -l)
        close(pipefd[1]); // Close write end
        dup2(pipefd[0], STDIN_FILENO); // Redirect stdin to pipe read end
        close(pipefd[0]);

        execlp("wc", "wc", "-l", (char *)NULL); // Execute wc -l
        perror("execlp"); // If execlp fails
        return 1;
    }

    // Parent process
    close(pipefd[0]);
    close(pipefd[1]);
    wait(NULL); // Wait for child 1 to finish
    wait(NULL); // Wait for child 2 to finish

    return 0;
}
```

Explanation:

1. **Pipe Creation:** A pipe is created using `pipe(pipefd)`. This provides a communication channel between processes.
2. **First Child Process (ls -l):**
   - Fork a child process to run `ls -l`.
   - Close the read end of the pipe and redirect the write end of the pipe to the standard output (`stdout`) using `dup2()`.
   - Execute `ls -l` using `execlp()`.
3. **Second Child Process (wc -l):**
   - Fork another child process to run `wc -l`.
   - Close the write end of the pipe and redirect the read end of the pipe to the standard input (`stdin`) using `dup2()`.
   - Execute `wc -l` using `execlp()`.
4. **Parent Process:** The parent closes both pipe ends and waits for both child processes to finish using `wait()`.

---

2. Parent process manages child timeout (Slip 3, 15, 24)

Question:
Write a C program to create a child process to execute a Unix/Linux command or user-defined program. The parent sets signal handlers for child termination and alarm signals. If the child doesn't complete execution in 5 seconds, the parent kills the child process.
Example Slips: 3, 15, 24

Answer:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

void timeout_handler(int sig) {
    printf("Timeout! Killing the child process...\n");
    exit(0); // Terminate the parent process after timeout
}

int main() {
    pid_t pid;
    struct sigaction sa;

    // Setup the timeout signal handler
    sa.sa_handler = timeout_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGALRM, &sa, NULL);

    // Create child process
    pid = fork();
```

```
    if (pid == 0) {
        // Child process (simulate a task that runs for more than 5
seconds)
        sleep(10); // Child process sleeps for 10 seconds
        printf("Child completed its task.\n");
        exit(0);
    }

    // Parent process
    alarm(5); // Set an alarm to trigger after 5 seconds
    int status;
    waitpid(pid, &status, 0); // Wait for the child to terminate

    if (WIFEXITED(status)) {
        printf("Child terminated with exit status %d\n",
WEXITSTATUS(status));
    }

    return 0;
}
```

Explanation:

1. **Timeout Handler:** The parent process sets a signal handler for `SIGALRM` using `sigaction()`. This handler will terminate the parent if the timeout occurs.
2. **Child Process:** The child process is created using `fork()` and is made to sleep for 10 seconds, simulating a long-running task.
3. **Parent Process:** The parent sets an alarm to trigger after 5 seconds using `alarm()`. If the child does not finish in this time, the signal handler will terminate the parent process.

---

3. Signal handling: SIGHUP, SIGINT, SIGQUIT (Slip 4, 16)

Question:
Write a C program where a child process catches the signals `SIGHUP`, `SIGINT`, and `SIGQUIT`. The parent sends `SIGHUP` or `SIGINT` every 3 seconds and sends `SIGQUIT` after a specific time. The child then terminates with a message.
Example Slips: 4, 16

Answer:

```
c
Copy code
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

void signal_handler(int sig) {
    if (sig == SIGHUP) {
        printf("Received SIGHUP signal\n");
    } else if (sig == SIGINT) {
        printf("Received SIGINT signal\n");
    } else if (sig == SIGQUIT) {
```

```
        printf("Received SIGQUIT signal, terminating.\n");
        exit(0);
    }
}

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

    if (pid == 0) {
        // Child process: set up signal handlers
        signal(SIGHUP, signal_handler);
        signal(SIGINT, signal_handler);
        signal(SIGQUIT, signal_handler);

        // Wait for signals
        while (1) {
            sleep(1); // Keep the child alive to handle signals
        }
    } else {
        // Parent process: send signals at intervals
        sleep(3); // Wait before sending first signal
        kill(pid, SIGHUP);
        sleep(3); // Wait before sending second signal
        kill(pid, SIGINT);
        sleep(5); // Wait before sending third signal
        kill(pid, SIGQUIT);
    }

    return 0;
}
```

Explanation:

1. **Signal Handler:** A signal handler is defined for `SIGHUP`, `SIGINT`, and `SIGQUIT`. When any of these signals is received, a message is printed.
2. **Child Process:** The child process sets up signal handlers using `signal()` for the three signals. It then enters an infinite loop to remain active and handle incoming signals.
3. **Parent Process:** The parent process sends `SIGHUP` and `SIGINT` signals at 3-second intervals and `SIGQUIT` after a 5-second delay, which causes the child to terminate.

---

4. Child writes to unnamed pipe (Slip 5, 18)

Question:
Write a C program to create an unnamed pipe where the child process writes multiple messages ("Hello World", "Hello SPPU", "Linux is Funny") and the parent process reads and displays them.
Example Slips: 5, 18

Answer:

```
c
Copy code
#include <stdio.h>
#include <unistd.h>
```

```
int main() {
    int pipefd[2];
    pid_t pid;

    if (pipe(pipefd) == -1) {
        perror("pipe");
        return 1;
    }

    pid = fork();
    if (pid == 0) {
        // Child process writes to pipe
        close(pipefd[0]); // Close the read end
        write(pipefd[1], "Hello World\n", 12);
        write(pipefd[1], "Hello SPPU\n", 11);
        write(pipefd[1], "Linux is Funny\n", 16);
        close(pipefd[1]); // Close write end after writing
    } else {
        // Parent process reads from pipe
        char buffer[100];
        close(pipefd[1]); // Close the write end
        while (read(pipefd[0], buffer, sizeof(buffer)) > 0) {
            printf("%s", buffer); // Display the message
        }
        close(pipefd[0]); // Close read end after reading
        wait(NULL); // Wait for the child to finish
    }

    return 0;
}
```

Explanation:

1. **Unnamed Pipe:** An unnamed pipe is created using `pipe(pipefd)`, which provides a two-way communication channel (read and write ends).
2. **Child Process:** The child writes multiple messages to the pipe using `write()`.
3. **Parent Process:** The parent reads the messages from the pipe using `read()` and prints them.

5. Sort files by size (Slip 12, 21)

Question:
Write a C program that receives filenames as command-line arguments and displays them in ascending order according to their sizes.
Example Slips: 12, 21

Answer:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

int compare(const void *a, const void *b) {
```

```
    struct stat stat_a, stat_b;
    stat(*(const char **)a, &stat_a);
    stat(*(const char **)b, &stat_b);
    return stat_a.st_size - stat_b.st_size;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <filenames>\n", argv[0]);
        return 1;
    }

    // Sort files by size
    qsort(argv + 1, argc - 1, sizeof(char *), compare);

    // Display sorted filenames
    for (int i = 1; i < argc; i++) {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```

Explanation:

1. **Compare Function:** `compare()` uses the `stat()` system call to retrieve the size of each file and returns the difference for sorting.
2. **Sorting:** The `qsort()` function is used to sort the filenames based on file size.
3. **Output:** After sorting, the program prints the filenames in ascending order of their sizes.

---

6. File type identification using stat() (Slip 9, 20, 23)

Question:
Write a C program to identify the type of a file (Directory, Regular file, Block device, Character device, FIFO, Symbolic link, or Socket) using the `stat()` system call.
Example Slips: 9, 20, 23

Answer:

```
c
Copy code
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

void print_file_type(struct stat st) {
    if (S_ISREG(st.st_mode)) {
        printf("Regular file\n");
    } else if (S_ISDIR(st.st_mode)) {
        printf("Directory\n");
    } else if (S_ISBLK(st.st_mode)) {
        printf("Block device\n");
    } else if (S_ISCHR(st.st_mode)) {
```

```
        printf("Character device\n");
    } else if (S_ISFIFO(st.st_mode)) {
        printf("FIFO\n");
    } else if (S_ISLNK(st.st_mode)) {
        printf("Symbolic link\n");
    } else if (S_ISSOCK(st.st_mode)) {
        printf("Socket\n");
    } else {
        printf("Unknown file type\n");
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    struct stat st;
    if (stat(argv[1], &st) == 0) {
        print_file_type(st);
    } else {
        perror("stat");
    }

    return 0;
}
```

Explanation:

1. **File Type Identification:** The `stat()` system call retrieves information about a file, which is then used to determine its type using macros like `S_ISREG()`, `S_ISDIR()`, etc.
2. **Output:** The program prints the file type (regular file, directory, etc.) based on the file's mode.

---

7. Parent-child communication with signals (Slip 2, 13)

Question:
Write a C program where a child process handles `SIGINT` and displays an appropriate message when caught. If `SIGINT` is received again, the program exits.
Example Slips: 2, 13

Answer:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void sigint_handler(int sig) {
    static int count = 0;
```

```
    count++;
    if (count == 1) {
        printf("Received SIGINT, but continuing...\n");
    } else {
        printf("Received SIGINT again, exiting...\n");
        exit(0);
    }
}

int main() {
    signal(SIGINT, sigint_handler);

    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        while (1) {
            sleep(1); // Keep the child alive to receive SIGINT
        }
    }

    // Parent process
    kill(pid, SIGINT); // Send SIGINT to child
    sleep(1); // Allow time for child to process the signal
    kill(pid, SIGINT); // Send SIGINT again to terminate the child

    return 0;
}
```

Explanation:

1. **SIGINT Handler:** A signal handler is set for `SIGINT`. The handler counts the number of times `SIGINT` is received. If it's the first signal, it prints a message and continues; if it's received again, the child exits.
2. **Parent-Child Interaction:** The parent sends `SIGINT` to the child process twice, and the child handles it based on the count.

---

8. Redirect output to output.txt using dup (Slip 8, 25)

Question:
Write a C program that redirects the standard output to a file (`output.txt`) using `dup` and `open` system calls.
Example Slips: 8, 25

Answer:

```
c
Copy code
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
```

```
    if (fd == -1) {
        perror("open");
        return 1;
    }

    // Redirect stdout to file
    dup2(fd, STDOUT_FILENO);
    close(fd);

    // Now, printf will write to output.txt
    printf("This will be written to output.txt\n");

    return 0;
}
```

Explanation:

1. **Redirect Output:** The `open()` system call opens (or creates) the `output.txt` file, and `dup2()` redirects the standard output (`stdout`) to this file.
2. **Writing to File:** Any `printf()` statement after the redirection will write to `output.txt`.

---

9. Parent-child process with unnamed pipe (Slip 10)

Question:
Write a C program where the parent process writes to an unnamed pipe and the child process reads from it.
Example Slips: 10

Answer:

```
c
Copy code
#include <stdio.h>
#include <unistd.h>

int main() {
    int pipefd[2];
    pid_t pid;

    if (pipe(pipefd) == -1) {
        perror("pipe");
        return 1;
    }

    pid = fork();
    if (pid == 0) {
        // Child process reads from pipe
        close(pipefd[1]); // Close the write end
        char buffer[100];
        read(pipefd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(pipefd[0]); // Close the read end
    } else {
        // Parent process writes to pipe
```

```
        close(pipefd[0]); // Close the read end
        write(pipefd[1], "Hello from Parent", 18);
        close(pipefd[1]); // Close the write end
    }

    return 0;
}
```

Explanation:

1. **Unnamed Pipe:** An unnamed pipe is created using `pipe(pipefd)`.
2. **Parent Process:** The parent writes a message to the pipe using `write()`.
3. **Child Process:** The child reads the message from the pipe using `read()` and prints it.

10. Cumulative time for child processes (Slip 6)

Question:
Write a C program to create multiple child processes. After all processes terminate, calculate the total cumulative time spent in user and kernel modes.
Example Slips: 6

Answer:

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/times.h>

int main() {
    struct tms start, end;
    clock_t start_time, end_time;

    start_time = times(&start);

    pid_t pid1 = fork();
    if (pid1 == 0) {
        // First child
        for (int i = 0; i < 1000000; i++); // Some work
        exit(0);
    }

    pid_t pid2 = fork();
    if (pid2 == 0) {
        // Second child
        for (int i = 0; i < 1000000; i++); // Some work
        exit(0);
    }

    // Parent process waits for children to terminate
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);
```

```
    end_time = times(&end);
    printf("Total User time: %ld\n", end.tms_utime - start.tms_utime);
    printf("Total Kernel time: %ld\n", end.tms_stime - start.tms_stime);

    return 0;
}
```

Explanation:

1. **Process Creation:** Two child processes are created using `fork()`. Each performs some dummy work.
2. **Time Calculation:** The `times()` function is used to record the process times. After both children terminate, the parent calculates and displays the cumulative user and kernel times.

1. What is an inode in a file system?

Answer: An inode is a data structure in a file system that stores metadata about a file, such as its size, ownership, permissions, and the locations of its data blocks.

2. What is the function of `fork()` in process creation?

Answer: `fork()` is a system call that creates a new process by duplicating the calling (parent) process. The new process is called the child process.

3. What is the `exec()` system call used for?

Answer: `exec()` replaces the current process image with a new process image. It loads a new program into the current process, replacing its code and data.

4. What does the `stat()` system call do?

Answer: The `stat()` system call retrieves information about a file, such as its size, permissions, inode number, and timestamps of last access and modification.

5. What does the `kill()` system call do?

Answer: `kill()` sends a signal to a process, which can be used to terminate the process or trigger a specific action (such as handling an interrupt or alarm).

6. Explain the purpose of `pipe()` in inter-process communication.

Answer: `pipe()` creates a unidirectional communication channel between processes, where data written to the pipe by one process can be read by another process.

7. What is the function of `sleep()` in a program?

Answer: `sleep()` causes the calling process to pause for a specified amount of time (in seconds), allowing other processes to execute.

8. What does the signal `SIGINT` represent?

Answer: `SIGINT` is a signal sent to a process when the user interrupts it, typically by pressing `Ctrl+C` on the terminal.

9. What is the use of `open()` system call?

Answer: `open()` is used to open a file or device for reading or writing, and it returns a file descriptor that can be used for subsequent file operations.

10. What is the purpose of using `dup()` system call?

Answer: `dup()` duplicates an existing file descriptor, allowing it to refer to the same underlying file or resource as the original file descriptor.

11. What does `wait()` do in a parent process?

Answer: `wait()` causes the parent process to wait until all of its child processes have terminated, allowing it to retrieve the child's exit status.

12. What does `SIGHUP` signal represent?

Answer: `SIGHUP` is a signal sent to a process when its controlling terminal is closed, typically indicating that the terminal or session is being closed.

13. What is the purpose of redirecting standard output?

Answer: Redirecting standard output allows the output of a command or program to be sent to a file or another destination instead of the terminal.

14. What is a `FIFO` in Unix/Linux?

Answer: A FIFO (First In, First Out) is a type of named pipe used for inter-process communication, where data written by one process is read by another in the same order.

15. Explain the role of `waitpid()` in process management.

Answer: `waitpid()` allows a parent process to wait for a specific child process to finish, and it can also retrieve the child's exit status.