Process Management

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

int main() {
    pid_t child_pid;
    int status;

    // Fork a child process
    child_pid = fork();

    if (child_pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // This is the child process
        printf("Child process: PID = %d\n", getpid());
        printf("Child process: PPID = %d\n", getppid());
        printf("Child process: Executing ls command...\n");
        execlp("ls", "ls", "-l", NULL);
        // execlp() will only return if there's an error
        perror("execlp failed");
        exit(EXIT_FAILURE);
    } else {
        // This is the parent process
        printf("Parent process: PID = %d\n", getpid());
        printf("Parent process: Waiting for child process to terminate...\n");
        wait(&status);
        if (WIFEXITED(status)) {
            printf("Parent process: Child process exited with status %d\n",
WEXITSTATUS(status));
        } else {
            printf("Parent process: Child process exited abnormally\n");
        }
    }

    return 0;
}
```

Information Maintenance

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

int main() {
    // Get and print the process ID (PID) of the current process
    pid_t pid = getpid();
    printf("Process ID (PID): %d\n", pid);

    // Get and print the parent process ID (PPID) of the current process
    pid_t ppid = getppid();
    printf("Parent Process ID (PPID): %d\n", ppid);

    // Get and print the user ID (UID) of the current user executing the program
    uid_t uid = getuid();
    printf("User ID (UID): %d\n", uid);

    return 0;
}
```

File Management

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

int main() {
    int fd1, fd2;
    ssize_t bytes_read, bytes_written;
    char buffer[1024];

    // Open input file "input.txt" for reading
    fd1 = open("input.txt", O_RDONLY);
    if (fd1 == -1) {
        perror("Failed to open input file");
        exit(EXIT_FAILURE);
    }
    // Open output file "output.txt" for writing (create if not exists, truncate
if exists)
    fd2 = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (fd2 == -1) {
        perror("Failed to open output file");
        exit(EXIT_FAILURE);
    }

    // Read from input file and write to output file
    while ((bytes_read = read(fd1, buffer, sizeof(buffer))) > 0) {
        bytes_written = write(fd2, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Write error");
            exit(EXIT_FAILURE);
        }
    }

    if (bytes_read == -1) {
        perror("Read error");
        exit(EXIT_FAILURE);
    }

    // Close input and output files
    if (close(fd1) == -1) {
        perror("Failed to close input file");
        exit(EXIT_FAILURE);
    }
    if (close(fd2) == -1) {
        perror("Failed to close output file");
        exit(EXIT_FAILURE);
    }

    printf("File copied successfully.\n");

    return 0;
}
```