q1.
```c
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;
int shared_variable = 0;

void *increment(void *arg) {
    for (int i = 0; i < 5; ++i) {
        pthread_mutex_lock(&mutex);
        shared_variable++;
        printf("Thread %s: %d\n", (char *)arg, shared_variable);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    pthread_mutex_init(&mutex, NULL);

    pthread_create(&thread1, NULL, increment, "A");
    pthread_create(&thread2, NULL, increment, "B");

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Q2.Open File in Read-Only Mode and Read the Last 5 Characters

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "r");

    if (file != NULL) {
        fseek(file, -5, SEEK_END);

        char buffer[6];  // 5 characters + null terminator
        fread(buffer, sizeof(char), 5, file);
        buffer[5] = '\0';

        printf("Last 5 characters: %s\n", buffer);

        fclose(file);
    } else {
        printf("Error opening the file.\n");
    }

    return 0;
}
```

Q3.Create a File and Write "Hello" after 4 Characters

```
#include <fcntl.h>
#include <unistd.h>

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

    if (fd != -1) {
        lseek(fd, 4, SEEK_SET);  // Move the cursor to the 4th position
        write(fd, "hello", 5);  // Write "hello"

        close(fd);
    }

    return 0;
}
```

4.Calculate Addition in Parent Process, Display Result in Child Process

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

int main() {
    int fd[2];
    pipe(fd);

    pid_t pid = fork();

    if (pid == 0) { // Child process
        close(fd[1]); // Close write end
        int result;
        read(fd[0], &result, sizeof(result));
        close(fd[0]); // Close read end

        printf("Child process: Sum is %d\n", result);
    } else if (pid > 0) { // Parent process
        close(fd[0]); // Close read end
        int num1 = 10, num2 = 20, sum;

        sum = num1 + num2;
        write(fd[1], &sum, sizeof(sum));
        close(fd[1]); // Close write end
    }

    return 0;
}
```

Q5.Write into a Pipe using popen() and pclose()

```
#include <stdio.h>

int main() {
```

```
    FILE *pipe_fp;
    char buffer[20];

    pipe_fp = popen("echo 'Hello, Pipe!'", "r");

    if (pipe_fp != NULL) {
        fread(buffer, sizeof(char), sizeof(buffer), pipe_fp);
        printf("Received from pipe: %s\n", buffer);
        pclose(pipe_fp);
    } else {
        printf("Error opening pipe.\n");
    }

    return 0;
}
```

Q6.Create Two Threads - Print Numbers and Check Even/Odd

```
#include <stdio.h>
#include <pthread.h>

void *printNumbers(void *arg) {
    for (int i = 1; i <= 10; ++i) {
        printf("Thread 1: %d\n", i);
    }
    return NULL;
}

void *checkEvenOdd(void *arg) {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("Thread 2: Even\n");
    } else {
        printf("Thread 2: Odd\n");
    }

    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, printNumbers, NULL);
    pthread_create(&thread2, NULL, checkEvenOdd, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Q7.Read from 3rd to 10th Character using System Calls

```
#include <fcntl.h>
#include <unistd.h>

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

    if (fd != -1) {
        lseek(fd, 2, SEEK_SET);  // Move the cursor to the 3rd character
        char buffer[9]; // 8 characters + null terminator
        read(fd, buffer, sizeof(buffer) - 1);
        buffer[8] = '\0';

        printf("Characters 3 to 10: %s\n", buffer);

        close(fd);
    } else {
        printf("Error opening the file.\n");
    }

    return 0;
}
```

Q8.Process Hierarchy - P1 has children P2 and P3

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

int main() {
    pid_t pid1, pid2, pid3;

    pid1 = getpid();

    if ((pid2 = fork()) == 0) {
        // Child process P2
        printf("P2: PID=%d, Parent PID=%d\n", getpid(), getppid());
    } else if (pid2 > 0) {
        // Parent process P1
        if ((pid3 = fork()) == 0) {
            // Child process P3
            printf("P3: PID=%d, Parent PID=%d\n", getpid(), getppid());
        } else if (pid3 > 0) {
            // Parent process P1
            printf("P1: PID=%d\n", pid1);
            sleep(2); // Ensure P2 and P3 print before P1
        } else {
            perror("Error creating P3");
            exit(EXIT_FAILURE);
        }
    } else {
```

```
        perror("Error creating P2");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

Q9.Hierarchy with P1 -> P2 ->

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

int main() {
    pid_t pid1, pid2, pid3;

    pid1 = getpid();

    if ((pid2 = fork()) == 0) {
        // Child process P2
        printf("P2: PID=%d, Parent PID=%d)
```