## Q1a) Write a C program that creates a directory and then changes its permissions using the `chmod` system call.

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

int main() {
    // Create directory
    if (mkdir("new_dir", 0777) == -1) {
        perror("mkdir failed");
        exit(1);
    }

    // Change permissions to rwxr-xr-x (0755)
    if (chmod("new_dir", 0755) == -1) {
        perror("chmod failed");
        exit(1);
    }

    printf("Directory created and permissions changed.\n");
    return 0;
}
```

## Q1b) Write a C program that uses threads to sort an array of integers.

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

#define SIZE 10

int arr[SIZE] = {9, 2, 5, 1, 7, 4, 8, 3, 6, 0};

// Thread function to sort a portion of the array
void* sort_thread(void* arg) {
    int start = *((int*)arg);
    for (int i = start; i < start + SIZE/2; i++) {
        for (int j = start; j < start + SIZE/2 - 1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t t1, t2;
    int start1 = 0, start2 = SIZE/2;

    // Create two threads to sort halves of the array
    pthread_create(&t1, NULL, sort_thread, &start1);
    pthread_create(&t2, NULL, sort_thread, &start2);
    // Wait for threads to finish
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    // Merge the sorted halves (simplified for brevity)
    printf("Sorted array: ");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

**Q2a) Create two directories named ABC and XYZ. Inside ABC create a file `f1.txt`. Give the permissions `rw-r-x---` to `f1.txt`. Move the file `f1.txt` to XYZ directory and rename it as `f2`. Create a copy of file `f2` to ABC directory and name it as `f3`.**

```
mkdir ABC XYZ
touch ABC/f1.txt
chmod 650 ABC/f1.txt
mv ABC/f1.txt XYZ/f2
cp XYZ/f2 ABC/f3
```

**Q2b) Write a shell program to print the Fibonacci series up to n terms using recursion.**

```
#!/bin/bash

fib() {
    if [ $1 -le 1 ]; then
        echo $1
    else
        echo $(( $(fib $(( $1 - 1 )) ) + $(fib $(( $1 - 2 )) ) ))
    fi
}

echo "Enter number of terms:"
read n
for (( i=0; i<n; i++ )); do
    echo -n "$(fib $i) "
done
Echo
```

**Q3a) Write a C program that demonstrates a race condition using multiple threads.**

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

int counter = 0;

void* increment(void* arg) {
    for (int i = 0; i < 100000; i++) {
        counter++;
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    // Expected 200000, but actual value is less due to race condition
    printf("Final counter value: %d\n", counter);
    return 0;
}
```

### Q3b) Write a C program that reads user input from the console and writes it to a file named `input.txt`.

```
#include <stdio.h>

int main() {
    char input[100];
    FILE *fp = fopen("input.txt", "w");

    printf("Enter text: ");
    fgets(input, sizeof(input), stdin);
    fprintf(fp, "%s", input);

    fclose(fp);
    return 0;
}
```

### Q4a) Write a C program to create three processes P1, P2, and P3 in the hierarchy P1 → P2 → P3. Display the PID and PPID of all processes.

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

int main() {
    pid_t p1 = getpid();
    printf("P1 PID: %d, PPID: %d\n", p1, getppid());

    if (fork() == 0) {  // P2
        printf("P2 PID: %d, PPID: %d\n", getpid(), getppid());
        if (fork() == 0) {  // P3
            printf("P3 PID: %d, PPID: %d\n", getpid(), getppid());
        }
    }

    wait(NULL);  // Wait for child processes
    return 0;
}
```

## Q4b) Write a C program to read the last 10 characters of a file using `lseek()` and `read()`.

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

int main() {
    int fd = open("file.txt", O_RDONLY);
    lseek(fd, -10, SEEK_END);  // Move to 10th byte from end

    char buf[11];
    read(fd, buf, 10);
    buf[10] = '\0';

    printf("Last 10 chars: %s\n", buf);
    close(fd);
    return 0;
}
```

## Q5a) Write a C program to create a directory ABC and then create a file F1 inside it using system calls.

```
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    mkdir("ABC", 0755);
    int fd = creat("ABC/F1", 0644);
    close(fd);
    return 0;
}
```

## Q5b) Write a C program to create a process and print the first 10 odd numbers in the child process.

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

int main() {
    if (fork() == 0) {  // Child process
        printf("Child process (10 odd numbers): ");
        for (int i = 1; i <= 19; i += 2) {
            printf("%d ", i);
        }
        printf("\n");
    } else {  // Parent waits
        wait(NULL);
    }
    return 0;
}
```

## Q6a) Write a shell script to find the smallest of three numbers using nested `if-else`.

```
#!/bin/bash

echo "Enter three numbers:"
read a b c

if [ $a -lt $b ]; then
  if [ $a -lt $c ]; then
    echo "$a is smallest"
  else
    echo "$c is smallest"
  fi
else
  if [ $b -lt $c ]; then
    echo "$b is smallest"
  else
    echo "$c is smallest"
  fi
Fi
```

## Q6b) Write a C program that uses a semaphore to synchronize access to a shared variable.

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

sem_t sem;
int shared = 0;

void* thread_func(void* arg) {
  sem_wait(&sem);
  for (int i = 0; i < 100000; i++) {
    shared++;
  }
  sem_post(&sem);
  pthread_exit(NULL);
}

int main() {
  sem_init(&sem, 0, 1);  // Initialize semaphore
  pthread_t t1, t2;

  pthread_create(&t1, NULL, thread_func, NULL);
  pthread_create(&t2, NULL, thread_func, NULL);

  pthread_join(t1, NULL);
  pthread_join(t2, NULL);

  printf("Shared value: %d\n", shared);  // Correctly 200000
  sem_destroy(&sem);
  return 0;
}
```

## Q7a) Write a C program to create a file `abc.txt`, write text into it, close, reopen, and read characters 11–20.

```
#include <stdio.h>

int main() {
    FILE *fp = fopen("abc.txt", "w");
    fprintf(fp, "This is a sample text for testing the program.");
    fclose(fp);

    fp = fopen("abc.txt", "r");
    fseek(fp, 10, SEEK_SET);  // 11th character (index 10)
    char buf[11];
    fread(buf, 1, 10, fp);
    buf[10] = '\0';
    printf("11th to 20th chars: %s\n", buf);
    fclose(fp);
    return 0;
}
```

## Q7b) Write a shell script to reverse a number and check if it is a palindrome.

```
#!/bin/bash

echo "Enter a number:"
read num

reverse=$(echo $num | rev)
if [ $num -eq $reverse ]; then
    echo "$num is a palindrome"
else
    echo "$num is not a palindrome"
Fi
```

## Q8a) Write a C program to create a chain of 3 processes (each creates one child) and print their PIDs.

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

int main() {
    pid_t p1 = getpid();
    printf("P1 PID: %d\n", p1);

    if (fork() == 0) {  // P2
        printf("P2 PID: %d, Parent PID: %d\n", getpid(), getppid());
        if (fork() == 0) {  // P3
            printf("P3 PID: %d, Parent PID: %d\n", getpid(), getppid());
        }
    }

    wait(NULL);
    return 0;
}
```

## Q8b) Write a C program to print file statistics using system calls.

```
#include <sys/stat.h>
#include <stdio.h>

int main() {
   struct stat file_stat;
   stat("file.txt", &file_stat);

   printf("File Size: %ld bytes\n", file_stat.st_size);
   printf("File Permissions: %o\n", file_stat.st_mode & 0777);
   printf("File Inode: %lu\n", file_stat.st_ino);
   return 0;
}
```

## Q9a) Write a shell script to check if a number is prime.

```
#!/bin/bash

echo "Enter a number:"
read num

is_prime=1
for (( i=2; i*i<=$num; i++ )); do
   if [ $((num % i)) -eq 0 ]; then
      is_prime=0
      break
   fi
done

if [ $num -lt 2 ]; then
   is_prime=0
fi

if [ $is_prime -eq 1 ]; then
   echo "$num is prime"
else
   echo "$num is not prime"
Fi
```

## Q9b) Write a C program that eliminates a race condition using semaphores.

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

sem_t sem;
int counter = 0;

void* increment(void* arg) {
   sem_wait(&sem);
   for (int i = 0; i < 100000; i++) {
      counter++;
   }
```

```c
    sem_post(&sem);
    pthread_exit(NULL);
}

int main() {
    sem_init(&sem, 0, 1);
    pthread_t t1, t2;

    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Counter: %d\n", counter);  // Correctly 200000
    sem_destroy(&sem);
    return 0;
}
```

## Q10a) Write a C program to create a directory, print its contents, and remove it.

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

int main() {
    mkdir("temp_dir", 0755);
    system("ls temp_dir");  // Print contents
    rmdir("temp_dir");      // Remove directory
    return 0;
}
```

## Q10b) Write a C program to recursively traverse a directory and print filenames with serial numbering.

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

int count = 1;

void traverse(const char *dirpath) {
    DIR *dir = opendir(dirpath);
    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
            continue;
        }

        char path[1024];
        snprintf(path, sizeof(path), "%s/%s", dirpath, entry->d_name);

        printf("%d. %s\n", count++, path);
        struct stat statbuf;
        if (stat(path, &statbuf) continue;
        if (S_ISDIR(statbuf.st_mode)) {
            traverse(path);  // Recursive call for subdirectories
        }
    }
    closedir(dir);
}

int main() {
    traverse(".");  // Start from current directory
    return 0;
}
```