1. Develop a C program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process).

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

int main() {
    pid_t pid;
    int status;

    pid = fork();
    if (pid < 0) {
        fprintf(stderr, "Fork Failed\n");
        return 1;
    } else if (pid == 0) {
        printf("This is the child process with pid=%d\n", getpid());
        execl("/bin/ls", "ls", NULL);
        perror("execlfailed");
        _exit(1);
    } else {
        printf("Parent process,PID=%u\n", getpid());
        waitpid(pid, &status, 0);
        printf("Child completed with pid=%d\n", pid);
    }

    return 0;
}
```

3. Develop a C program to simulate Dining Philosophers problem using semaphores.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t room;
sem_t chopstick[5];

void *philosopher(void *);
void eat(int);

int main() {
    int i, a[5];
    pthread_t tid[5];

    sem_init(&room, 0, 4); // Allow only 4 philosophers in the room
    for (i = 0; i < 5; i++)
        sem_init(&chopstick[i], 0, 1); // Each chopstick starts as available

    for (i = 0; i < 5; i++) {
        a[i] = i;
        pthread_create(&tid[i], NULL, philosopher, (void *)&a[i]);
    }

    for (i = 0; i < 5; i++)
        pthread_join(tid[i], NULL);

    return 0;
}

void *philosopher(void *num) {
    int phil = *(int *)num;

    sem_wait(&room); // Enter the dining room (limit 4 philosophers)
    printf("\nPhilosopher %d has entered the room", phil);

    sem_wait(&chopstick[phil]); // Pick up left chopstick
    sem_wait(&chopstick[(phil + 1) % 5]); // Pick up right chopstick

    eat(phil);
    sleep(2);

    printf("\nPhilosopher %d has finished eating", phil);

    sem_post(&chopstick[(phil + 1) % 5]); // Release right chopstick
    sem_post(&chopstick[phil]); // Release left chopstick

    sem_post(&room); // Leave the room
}

void eat(int phil) {
    printf("\nPhilosopher %d is eating", phil);
}
```

10. Develop a C program to simulate SCAN disk scheduling algorithm.

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

int main() {
    int queue[20], n, head, i, j, seek = 0, max, diff, temp;
    int queue1[20], queue2[20], temp1 = 0, temp2 = 0;
    float avg;

    printf("Enter the max range of disk: ");
    scanf("%d", &max);

    printf("Enter the initial head position: ");
    scanf("%d", &head);

    printf("Enter the size of queue request: ");
    scanf("%d", &n);

    printf("Enter the queue of disk positions to be read:\n");
    for (i = 1; i <= n; i++) {
        scanf("%d", &temp);
        if (temp >= head) {
            queue1[temp1] = temp;
            temp1++;
        } else {
            queue2[temp2] = temp;
            temp2++;
        }
    }

    for (i = 0; i < temp1 - 1; i++) {
        for (j = i + 1; j < temp1; j++) {
            if (queue1[i] > queue1[j]) {
                temp = queue1[i];
                queue1[i] = queue1[j];
                queue1[j] = temp;
            }
        }
    }

    for (i = 0; i < temp2 - 1; i++) {
        for (j = i + 1; j < temp2; j++) {
            if (queue2[i] > queue2[j]) {
                temp = queue2[i];
                queue2[i] = queue2[j];
                queue2[j] = temp;
            }
        }
    }
    for (i = 1, j = 0; j < temp1; i++, j++)
        queue[i] = queue1[j];

    queue[i] = max;
    queue[i + 1] = 0;

    for (i = temp1 + 3, j = 0; j < temp2; i++, j++)
        queue[i] = queue2[j];

    queue[0] = head;

    printf("\nDisk movement sequence:\n");
    for (j = 0; j <= n + 1; j++) {
        diff = abs(queue[j + 1] - queue[j]);
        seek += diff;
        printf("Disk head moves from %d to %d with seek %d\n", queue[j], queue[j + 1], diff);
    }

    printf("\nTotal seek time is %d\n", seek);
    avg = seek / (float)n;
    printf("Average seek time is %.2f\n", avg);

    return 0;
}
```

9. Develop a C program to simulate the Linked file allocation strategies.

```
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
```

```
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
}
```

4. Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

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

#define FIFO_NAME "myfifo"

int main() {
    int fd;
    char buffer[BUFSIZ];

    // Create the FIFO (named pipe) if it doesn't already exist
    if (mkfifo(FIFO_NAME, 0666) == -1) {
        perror("mkfifo");
        exit(EXIT_FAILURE);
    }
    // Create a child process
    if (fork() == 0) {
        // Writer process
        printf("Writer process is running. Enter data to write (type 'exit' to quit):\n");

        fd = open(FIFO_NAME, O_WRONLY); // Open FIFO in write-only mode
        if (fd == -1) {
            perror("open");
            exit(EXIT_FAILURE);
        }

        while (1) {
            fgets(buffer, BUFSIZ, stdin);
            if (strcmp(buffer, "exit\n") == 0) {
                break;
            }
            write(fd, buffer, strlen(buffer) + 1); // Write input to FIFO
        }
        close(fd); // Close FIFO after writing
    }
    else {
        // Reader process
        printf("Reader process is running. Reading data from the FIFO:\n");

        fd = open(FIFO_NAME, O_RDONLY); // Open FIFO in read-only mode
        if (fd == -1) {
            perror("open");
            exit(EXIT_FAILURE);
        }

        while (1) {
            if (read(fd, buffer, BUFSIZ) == 0) {
                break;
            }
            printf("Received: %s", buffer); // Print received message
        }
        close(fd); // Close FIFO after reading
    }
    return 0;
}
```

8. Simulate following File Organization Techniques:

a) Single level directory

b) Two level directory.

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

```
struct
{
   char dname[10], fname[10][10];
   int fcnt;
} dir;

void main()
{
   int i, ch;
   char f[30];
   dir.fcnt = 0;

   printf("\nEnter name of directory -- ");
   scanf("%s", dir.dname);

   while(1)
   {
      printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter
your choice -- ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            printf("\n Enter the name of the file -- ");
            scanf("%s", dir.fname[dir.fcnt]);
            dir.fcnt++;
            break;

         case 2:
            printf("\n Enter the name of the file -- ");
            scanf("%s", f);
            for(i = 0; i < dir.fcnt; i++)
            {
               if(strcmp(f, dir.fname[i]) == 0)
               {
                  printf("File %s is deleted ", f);
                  strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);
                  break;
               }
            }
            if(i == dir.fcnt)
               printf("File %s not found", f);
            else
               dir.fcnt--;
            break;

         case 3:
            printf("\n Enter the name of the file -- ");
```

```
                scanf("%s", f);
                for(i = 0; i < dir.fcnt; i++)
                {
                    if(strcmp(f, dir.fname[i]) == 0)
                    {
                        printf("File %s is found ", f);
                        break;
                    }
                }
                if(i == dir.fcnt)
                    printf("File %s not found", f);
                break;

            case 4:
                if(dir.fcnt == 0)
                    printf("\n Directory Empty");
                else
                {
                    printf("\n The Files are -- ");
                    for(i = 0; i < dir.fcnt; i++)
                        printf("\t%s", dir.fname[i]);
                }
                break;

            default:
                exit(0);
        }
    }
}
```

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

struct
{
    char dname[10], fname[10][10];
    int fcnt;
} dir[10]; // Array for multiple users (sub-directories)

int dcnt = 0; // Directory count

void main()
{
    int i, j, ch;
    char d[30], f[30];

    while (1)
```

```
{
    printf("\n1. Create Directory  2. Create File  3. Delete File\n");
    printf("4. Search File  5. Display Files  6. Exit\nEnter your choice: ");
    scanf("%d", &ch);

    switch (ch)
    {
    case 1: // Create Directory (User)
        printf("\nEnter directory name: ");
        scanf("%s", dir[dcnt].dname);
        dir[dcnt].fcnt = 0;
        dcnt++;
        break;

    case 2: // Create File in a Directory
        printf("\nEnter directory name: ");
        scanf("%s", d);
        for (i = 0; i < dcnt; i++)
        {
            if (strcmp(dir[i].dname, d) == 0)
            {
                printf("Enter file name: ");
                scanf("%s", dir[i].fname[dir[i].fcnt]);
                dir[i].fcnt++;
                break;
            }
        }
        if (i == dcnt)
            printf("Directory not found!\n");
        break;

    case 3: // Delete File
        printf("\nEnter directory name: ");
        scanf("%s", d);
        for (i = 0; i < dcnt; i++)
        {
            if (strcmp(dir[i].dname, d) == 0)
            {
                printf("Enter file name to delete: ");
                scanf("%s", f);
                for (j = 0; j < dir[i].fcnt; j++)
                {
                    if (strcmp(dir[i].fname[j], f) == 0)
                    {
                        printf("File %s deleted\n", f);
                        strcpy(dir[i].fname[j], dir[i].fname[--dir[i].fcnt]);
                        break;
                    }
                }
                if (j == dir[i].fcnt)
```

```
                    printf("File not found!\n");
                break;
            }
        }
        if (i == dcnt)
            printf("Directory not found!\n");
        break;

    case 4: // Search File
        printf("\nEnter directory name: ");
        scanf("%s", d);
        for (i = 0; i < dcnt; i++)
        {
            if (strcmp(dir[i].dname, d) == 0)
            {
                printf("Enter file name to search: ");
                scanf("%s", f);
                for (j = 0; j < dir[i].fcnt; j++)
                {
                    if (strcmp(dir[i].fname[j], f) == 0)
                    {
                        printf("File %s found\n", f);
                        break;
                    }
                }
                if (j == dir[i].fcnt)
                    printf("File not found!\n");
                break;
            }
        }
        if (i == dcnt)
            printf("Directory not found!\n");
        break;

    case 5: // Display Files
        printf("\nEnter directory name: ");
        scanf("%s", d);
        for (i = 0; i < dcnt; i++)
        {
            if (strcmp(dir[i].dname, d) == 0)
            {
                if (dir[i].fcnt == 0)
                    printf("No files found!\n");
                else
                {
                    printf("Files in %s: ", dir[i].dname);
                    for (j = 0; j < dir[i].fcnt; j++)
                        printf("%s ", dir[i].fname[j]);
                    printf("\n");
                }
```

```
                    break;
                }
            }
            if (i == dcnt)
                printf("Directory not found!\n");
            break;

        case 6:
            exit(0);

        default:
            printf("Invalid choice! Try again.\n");
        }
    }
}
```

5. Develop a C program to simulate the following contiguous memory allocation Techniques:

First-Fit:

```
#include <stdio.h>
#define max 25

void main() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max] = {0}, ff[max] = {0};  // Initialize bf[] and ff[] to 0

    printf("\n\tMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);

    printf("\nEnter the size of the blocks:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }

    printf("Enter the size of the files:\n");
    for (i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }

    // First Fit Allocation
    for (i = 0; i < nf; i++) {
        for (j = 0; j < nb; j++) {
            if (bf[j] == 0) {  // Check if the block is free
```

```
                temp = b[j] - f[i];
                if (temp >= 0) {
                    ff[i] = j;
                    frag[i] = temp;  // Corrected placement of fragmentation
                    bf[j] = 1;  // Mark block as allocated
                    break;
                }
            }
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (i = 0; i < nf; i++) {
        if (ff[i] != 0) {  // If file was allocated
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        } else {
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
        }
    }
}
```

Best-Fit:

```
#include <stdio.h>
#define MAX 25

void main() {
    int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, lowest;
    int bf[MAX] = {0}, ff[MAX] = {-1};  // Initialize allocation tracking arrays

    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);

    printf("\nEnter the size of the blocks:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }

    printf("Enter the size of the files:\n");
    for (i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }

    // Best Fit Allocation
    for (i = 0; i < nf; i++) {
        lowest = 10000;  // Reset lowest for each file
        int bestIndex = -1;

        for (j = 0; j < nb; j++) {
            if (bf[j] == 0) { // Check if block is free
                temp = b[j] - f[i];
                if (temp >= 0 && temp < lowest) {  // Finding the best fit block
                    bestIndex = j;
                    lowest = temp;
                }
            }
        }

        if (bestIndex != -1) { // If a suitable block was found
            ff[i] = bestIndex;
            frag[i] = lowest;
            bf[bestIndex] = 1; // Mark block as allocated
        } else {
            ff[i] = -1; // Indicate file not allocated
            frag[i] = -1;
        }
    }

    // Print the allocation details
    printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");
    for (i = 0; i < nf; i++) {
        if (ff[i] != -1) {
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        } else {
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
        }
    }
}
```

Worst-Fit:

```
#include <stdio.h>
#define MAX 25

void main() {
    int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, highest;
    int bf[MAX] = {0}, ff[MAX] = {0};  // Initialize allocation tracking arrays

    printf("\n\tMemory Management Scheme - Worst Fit");
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);

    printf("\nEnter the size of the blocks:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }

    printf("Enter the size of the files:\n");
    for (i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }

    // Worst Fit Allocation
    for (i = 0; i < nf; i++) {
        highest = -1;
        int bestIndex = -1;

        for (j = 0; j < nb; j++) {
            if (bf[j] == 0) { // Check if block is free
                temp = b[j] - f[i];
                if (temp >= 0 && temp > highest) {  // Finding the worst fit block
                    bestIndex = j;
                    highest = temp;
                }
            }
        }

        if (bestIndex != -1) { // If a suitable block was found
            ff[i] = bestIndex;
            frag[i] = highest;
            bf[bestIndex] = 1; // Mark block as allocated
        } else {
            ff[i] = -1; // Indicate file not allocated
            frag[i] = -1;
        }
    }

    // Print the allocation details
    printf("\nFile_no\tFile_size\tBlock_no\tBlock_size\tFragment");
    for (i = 0; i < nf; i++) {
        if (ff[i] != -1) {
            printf("\n%d\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        } else {
            printf("\n%d\t%d\t\tNot Allocated", i + 1, f[i]);
        }
    }
}
```