

WEEK 6

Program 1) Write a program to communicate parent and child process with each other in such a way that whenever child writes something, parent process can read it. Consider mode of communication is through.

a. pipe

Source Code:

```
#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main()
{
    int p[2];
    int returnstatus;
    char writing[2][25] = {"hello", "world"};
    char readmsg[25];
    returnstatus = pipe(p);
    if (returnstatus == -1)
    {
        printf("Pipe not created\n");
        return 1;
    }
    printf("Writing started: %s\n", writing[0]);
    write(p[1], writing[0], strlen(writing[0]) + 1);
    read(p[0], readmsg, sizeof(readmsg));
    printf("Reading from pipe - msg1: %s\n", readmsg);
    return 0;
}
```

OUTPUT

```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab$ cd
.git/ .vscode/ Manual/ WEEK1/ WEEK2/ WEEK3/ WEEK4/ WEEK5/ WEEK6/ WEEK7/ WEEK8/
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab$ cd WEEK6
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 1a.c -o 1a -lpthread
./1a
Writing started: hello
Reading from pipe - msg1: hello
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 1a.c -o 1a -lpthread
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./1a
Writing started: hello
Reading from pipe - msg1: hello
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$
```

Ln 21, Col 14 Spaces: 4 UTF-8 CRLF {} C Go Live Prettier

b) message passing

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <unistd.h>

struct msg_buffer

{   long msg_type;

    char msg_text[100];

} message;

void main()

{

    key_t key;

    int msgid;

    key = ftok("progfile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);

    message.msg_type = 1;

    if (fork() == 0)

    {   strcpy(message.msg_text, "Message from child");

        msgsnd(msgid, &message, sizeof(message), 0);

    }

    else

    {   msgrcv(msgid, &message, sizeof(message), 1, 0);

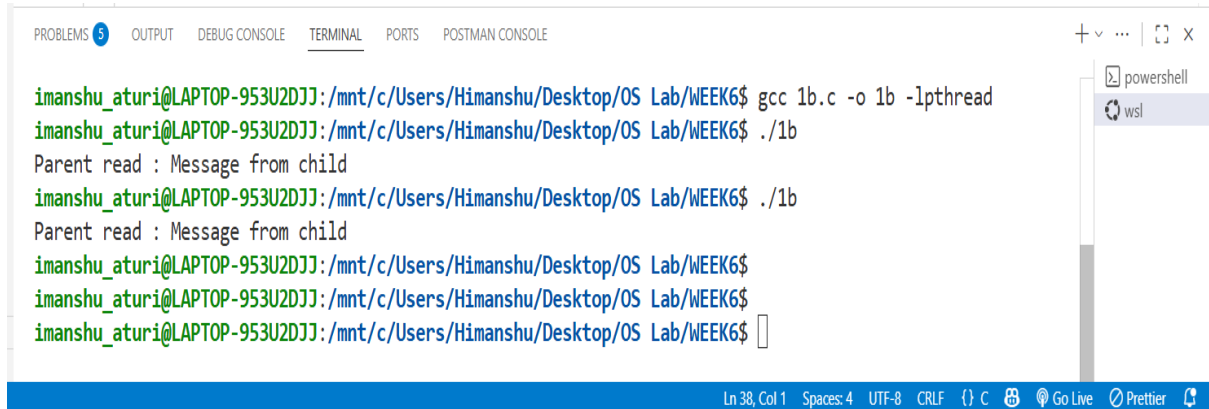
        printf("Parent read : %s\n", message.msg_text);

        msgctl(msgid, IPC_RMID, NULL);

    }

}
```

OUTPUT



```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 1b.c -o 1b -lpthread
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./1b
Parent read : Message from child
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./1b
Parent read : Message from child
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$
```

Ln 38, Col 1 Spaces: 4 UTF-8 CRLF {} C Go Live Prettier

c) shared memory

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    key_t key;
    int shmid;
    char *shared_memory;
    key = ftok(".", 65);
    if (key == -1)
    {
        perror("ftok");
        exit(1);
    }
    shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    if (shmid == -1)
    {
        perror("shmget");
        exit(1);
    }
    if (fork() == 0)
    {
        shared_memory = (char *)shmat(shmid, NULL, 0);
```

```

    if (shared_memory == (char *)(-1))
    {
        perror("shmat");
        exit(1);
    }

    strcpy(shared_memory, "Message from child process");
    printf("Child wrote: %s\n", shared_memory);

    shmdt(shared_memory);
}
else
{
    sleep(1);

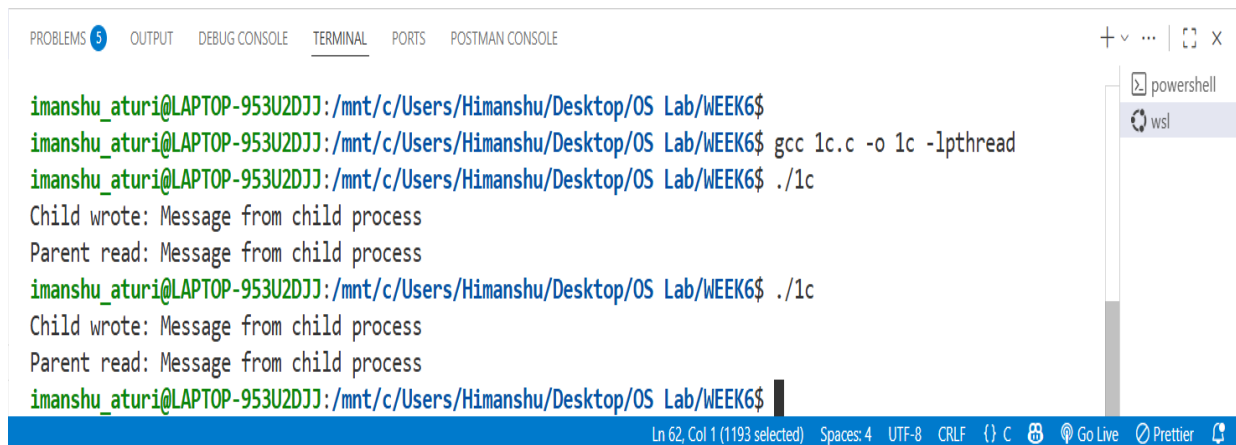
    shared_memory = (char *)shmat(shmid, NULL, 0);
    if (shared_memory == (char *)(-1))
    {
        perror("shmat");
        exit(1);
    }

    printf("Parent read: %s\n", shared_memory);

    shmdt(shared_memory);
    shmctl(shmid, IPC_RMID, NULL);
}
return 0;
}

```

OUTPUT:



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal displays the following commands and output:

```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$  
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 1c.c -o 1c -lpthread  
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./1c  
Child wrote: Message from child process  
Parent read: Message from child process  
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./1c  
Child wrote: Message from child process  
Parent read: Message from child process  
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$
```

The status bar at the bottom indicates: Ln 62, Col 1 (1193 selected) Spaces: 4 UTF-8 CRLF {} C Go Live Prettier.

Program2) Write a program to implement the concept of Producer-Consumer problem using semaphores.

Source Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];

int in = 0, out = 0;

sem_t empty;

sem_t full;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *producer(void *args)

{

    int item;

    for (int i = 0; i < 10; i++)

    {

        item = i + 1;

        sem_wait(&empty);

        pthread_mutex_lock(&mutex);

        buffer[in] = item;

        printf("Producer produced %d at index %d\n", item, in);

        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);

        sem_post(&full);

        sleep(1);

    }

    return NULL;

}
```

```

void *consumer(void *args)
{
    int item;

    for (int i = 0; i < 10; i++)
    {
        sem_wait(&full);

        pthread_mutex_lock(&mutex);

        item = buffer[out];

        printf("Consumer consumed %d from index %d\n", item, out);

        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);

        sem_post(&empty);

        sleep(2);
    }

    return NULL;
}

int main()
{
    pthread_t prod, cons;

    sem_init(&empty, 0, BUFFER_SIZE);

    sem_init(&full, 0, 0);

    pthread_create(&prod, NULL, producer, NULL);

    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);

    pthread_join(cons, NULL);

    sem_destroy(&empty);

    sem_destroy(&full);

    pthread_mutex_destroy(&mutex);

    return 0;
}

```

OUTPUT:

```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 2.c -o 2 -lpthread
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./2
Producer produced 1 at index 0
Consumer consumed 1 from index 0
Producer produced 2 at index 1
Consumer consumed 2 from index 1
Producer produced 3 at index 2
Producer produced 4 at index 3
Consumer consumed 3 from index 2
Producer produced 5 at index 4
Producer produced 6 at index 0
Consumer consumed 4 from index 3
Producer produced 7 at index 1
Producer produced 8 at index 2
Consumer consumed 5 from index 4
Producer produced 9 at index 3
Producer produced 10 at index 4
Consumer consumed 6 from index 0
Consumer consumed 7 from index 1
Consumer consumed 8 from index 2
Consumer consumed 9 from index 3
```

Program3) Write a program to implement the concept of Dining-Philosopher problem.

Source Code:

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <stdlib.h>

#include <time.h>

#define N 5

sem_t chopstick[N];

void *philosopher(void *num)

{

    int id = *(int *)num;

    while (1)

    {

        printf("Philosopher %d is thinking\n", id);

        sleep(rand() % 3 + 1);

        int left = id;

        int right = (id + 1) % N;

        if (sem_trywait(&chopstick[left]) == 0)

        {

            if (sem_trywait(&chopstick[right]) == 0)

            {

                printf("Philosopher %d is eating using chopsticks %d and %d\n", id, left, right);

                sleep(rand() % 2 + 1); // eating time

                sem_post(&chopstick[left]);

                sem_post(&chopstick[right]);

                printf("Philosopher %d has released chopsticks %d and %d\n", id, left, right);

            }

        }

    }

}
```

```

        else
        {
            sem_post(&chopstick[left]);

            printf("Philosopher %d released chopstick %d since right chopstick %d is unavailable\n",
                id, left, right);
        }
    }
else
{
    printf("Philosopher %d couldn't pick left chopstick %d, will try again\n", id, left);
}

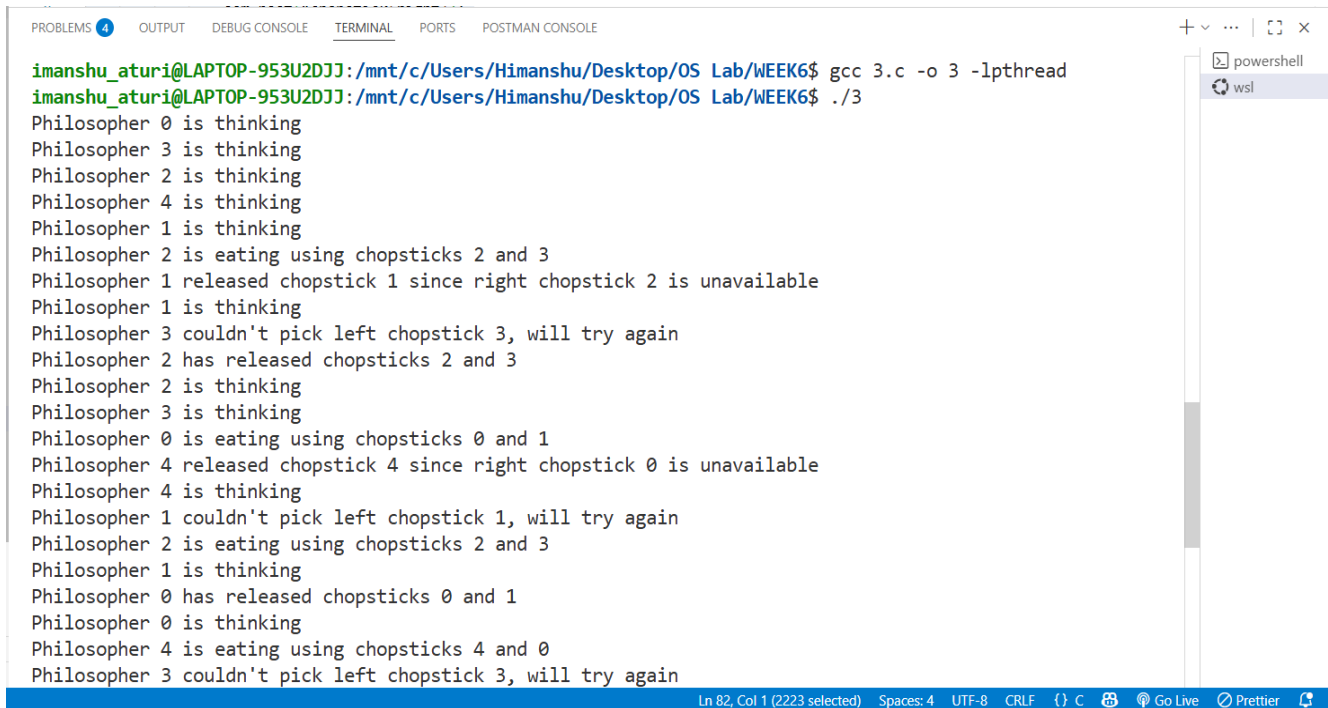
usleep(200); // small delay to reduce CPU usage
}

pthread_exit(NULL);
}

void main()
{
    pthread_t tid[N];
    int ids[N];
    srand(time(NULL));
    for (int i = 0; i < N; i++)
        sem_init(&chopstick[i], 0, 1);
    for (int i = 0; i < N; i++)
    {
        ids[i] = i;
        pthread_create(&tid[i], NULL, philosopher, &ids[i]);
    }
    for (int i = 0; i < N; i++)
        pthread_join(tid[i], NULL);
}

```

OUTPUT:



```
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ gcc 3.c -o 3 -lpthread
imanshu_aturi@LAPTOP-953U2DJJ:/mnt/c/Users/Himanshu/Desktop/OS Lab/WEEK6$ ./3
Philosopher 0 is thinking
Philosopher 3 is thinking
Philosopher 2 is thinking
Philosopher 4 is thinking
Philosopher 1 is thinking
Philosopher 2 is eating using chopsticks 2 and 3
Philosopher 1 released chopstick 1 since right chopstick 2 is unavailable
Philosopher 1 is thinking
Philosopher 3 couldn't pick left chopstick 3, will try again
Philosopher 2 has released chopsticks 2 and 3
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 0 is eating using chopsticks 0 and 1
Philosopher 4 released chopstick 4 since right chopstick 0 is unavailable
Philosopher 4 is thinking
Philosopher 1 couldn't pick left chopstick 1, will try again
Philosopher 2 is eating using chopsticks 2 and 3
Philosopher 1 is thinking
Philosopher 0 has released chopsticks 0 and 1
Philosopher 0 is thinking
Philosopher 4 is eating using chopsticks 4 and 0
Philosopher 3 couldn't pick left chopstick 3, will try again
```

WEEK 7

Program1) FIFO – First In First Out : page which came first (i.e. oldest page) need to be moved out.

Source Code:

```
#include <stdio.h>

void enqueue(int q[], int *front, int *rear, int v, int n)
{
    if (*rear - *front + 1 == n)
    {
        printf("Queue is full\n");
        return;
    }else
    {
        if (*front == -1)
            *front = 0;
        q[++(*rear)] = v;
    }
}

int dequeue(int q[], int *front, int *rear)
{
    int v;
    if (*front == -1)
    {
        printf("Queue is empty\n");
        return -1;
    }
    else
    {
        v = q[*front];
        (*front)++;
    }
}
```

```

        if (*front > *rear)
        {
            *front = -1;
            *rear = -1;
        } }

    return v;
}

int isPresent(int q[], int front, int rear, int page)
{
    if (front == -1)
        return 0;

    for (int i = front; i <= rear; i++)
    { if (q[i] == page)
        return 1; }

    return 0;
}

void display(int q[], int front, int rear)
{
    if (front == -1)
    {
        printf("Queue is empty\n");
        return;
    }

    for (int i = front; i <= rear; i++)
        printf("%d ", q[i]);

    printf("\n");
}

void main()
{
    int n, m;

```

```

printf("Enter number of frames: ");

scanf("%d", &n);

printf("Enter number of page requests: ");

scanf("%d", &m);

int pages[m];

for (int i = 0; i < m; i++)
    scanf("%d", &pages[i]);

int q[20];

int front = -1, rear = -1;

int pageFaults = 0;

for (int i = 0; i < m; i++)
{
    int page = pages[i];

    printf("Request for page %d -> ", page);

    if (!isPresent(q, front, rear, page))
    {
        pageFaults++;

        if (front == -1 || rear - front + 1 < n)
            enqueue(q, &front, &rear, page, n);

        else{

            dequeue(q, &front, &rear);

            enqueue(q, &front, &rear, page, n);

        }

        printf("Page Fault Frames: ");

    } else

        printf("Page Hit Frames: ");

    display(q, front, rear);

}

printf("\nTotal Page Faults = %d\n", pageFaults);

printf("Total Page Hits = %d\n", m - pageFaults);

}

```

OUTPUT:



```
PS C:\Users\Himanshu\Desktop\OS Lab> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK7\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }
Enter number of frames: 3
Enter number of page requests: 12
2 3 2 1 5 2 4 5 3 2 5 2
Request for page 2 -> Page Fault Frames: 2
Request for page 3 -> Page Fault Frames: 2 3
Request for page 2 -> Page Hit Frames: 2 3
Request for page 1 -> Page Fault Frames: 2 3 1
Request for page 5 -> Page Fault Frames: 3 1 5
Request for page 2 -> Page Fault Frames: 1 5 2
Request for page 4 -> Page Fault Frames: 5 2 4
Request for page 5 -> Page Hit Frames: 5 2 4
Request for page 3 -> Page Fault Frames: 2 4 3
Request for page 2 -> Page Hit Frames: 2 4 3
Request for page 5 -> Page Fault Frames: 4 3 5
Request for page 2 -> Page Fault Frames: 3 5 2

Total Page Faults = 9
Total Page Hits = 3
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK7>
```

Program2) LRU – Least Recently Used : page which is has not been used for longest Ame need to be moved out.

Source Code:

```
#include <stdio.h>

int isPresent(int frames[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frames[i] == page)
            return 1;
    }
    return 0;
}

int findLRU(int time[], int n) {
    int min = time[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i; }
    }
    return pos;
}

void display(int frames[], int n) {
    for (int i = 0; i < n; i++) {
        if (frames[i] != -1)
            printf("%d ", frames[i]);
        else
            printf("- ");
    }
    printf("\n");
}

void main()
```

```

{
    int n, m;

    printf("Enter number of frames: ");
    scanf("%d", &n);

    printf("Enter number of page requests: ");
    scanf("%d", &m);

    int pages[m];

    printf("Enter the page reference string: ");
    for (int i = 0; i < m; i++)
        scanf("%d", &pages[i]);

    int frames[n], time[n];

    int counter = 0, pageFaults = 0;
    for (int i = 0; i < n; i++)
    {
        frames[i] = -1;
        time[i] = 0;
    }

    for (int i = 0; i < m; i++)
    {
        int page = pages[i];

        printf("Request for page %d -> ", page);

        if (isPresent(frames, n, page))
        {
            for (int j = 0; j < n; j++)
            {
                if (frames[j] == page)
                    time[j] = ++counter;
            }

            printf("Page Hit Frames: ");

```

```

    }

    else
    {
        pageFaults++;

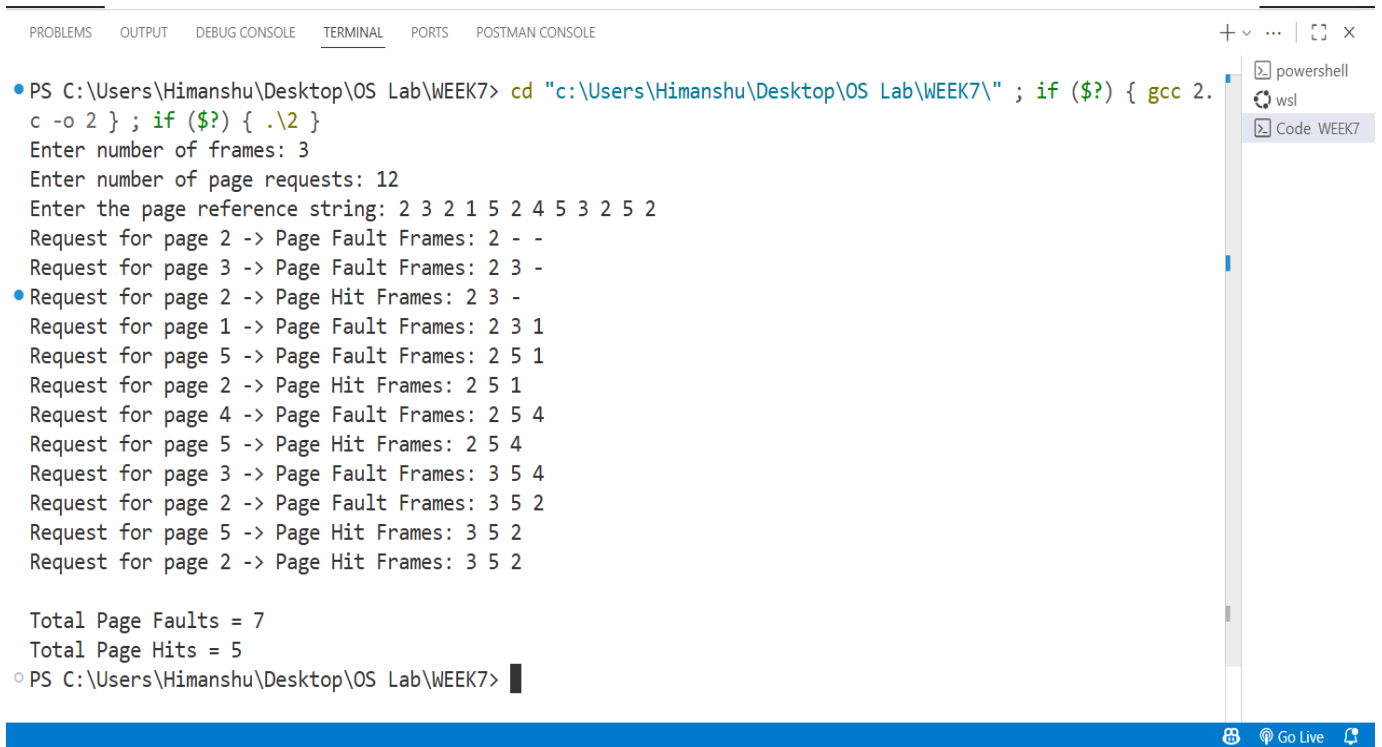
        int emptyPos = -1;
        for (int j = 0; j < n; j++)
        {
            if (frames[j] == -1)
            {
                emptyPos = j;
                break;
            }
        }
        if (emptyPos != -1)
        {
            frames[emptyPos] = page;
            time[emptyPos] = ++counter;
        }
        else {
            int pos = findLRU(time, n);
            frames[pos] = page;
            time[pos] = ++counter;
        }
        printf("Page Fault Frames: ");
    }

    display(frames, n);
}

printf("\nTotal Page Faults = %d\n", pageFaults);
printf("Total Page Hits = %d\n", m - pageFaults);
}

```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK7> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK7\" ; if ($?) { gcc 2.
c -o 2 } ; if ($?) { .\2 }
Enter number of frames: 3
Enter number of page requests: 12
Enter the page reference string: 2 3 2 1 5 2 4 5 3 2 5 2
Request for page 2 -> Page Fault Frames: 2 - -
Request for page 3 -> Page Fault Frames: 2 3 -
Request for page 2 -> Page Hit Frames: 2 3 -
Request for page 1 -> Page Fault Frames: 2 3 1
Request for page 5 -> Page Fault Frames: 2 5 1
Request for page 2 -> Page Hit Frames: 2 5 1
Request for page 4 -> Page Fault Frames: 2 5 4
Request for page 5 -> Page Hit Frames: 2 5 4
Request for page 3 -> Page Fault Frames: 3 5 4
Request for page 2 -> Page Fault Frames: 3 5 2
Request for page 5 -> Page Hit Frames: 3 5 2
Request for page 2 -> Page Hit Frames: 3 5 2

Total Page Faults = 7
Total Page Hits = 5
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK7>
```

WEEK 8

Program1) Best Fit – block which is closes to the size of request is allocated i.e. the smallest hole that is big enough to allocate to the requesting program.

Source Code:

```
#include <stdio.h>

int main()
{
    int b, p;

    printf("Enter number of free blocks available : ");
    scanf("%d", &b);

    int block[b];

    for (int i = 0; i < b; i++)
        scanf("%d", &block[i]);

    printf("Enter number of processes : ");
    scanf("%d", &p);

    int process[p];

    for (int i = 0; i < p; i++)
        scanf("%d", &process[i]);

    int allocated[p];

    for (int i = 0; i < p; i++)
        allocated[i] = -1;

    for (int i = 0; i < p; i++)
    {
        int bestIdx = -1;

        for (int j = 0; j < b; j++)
        {
            if (block[j] >= process[i])
            {
                if (bestIdx == -1 || block[j] < block[bestIdx])
                    bestIdx = j;
            }
        }
    }
}
```

```

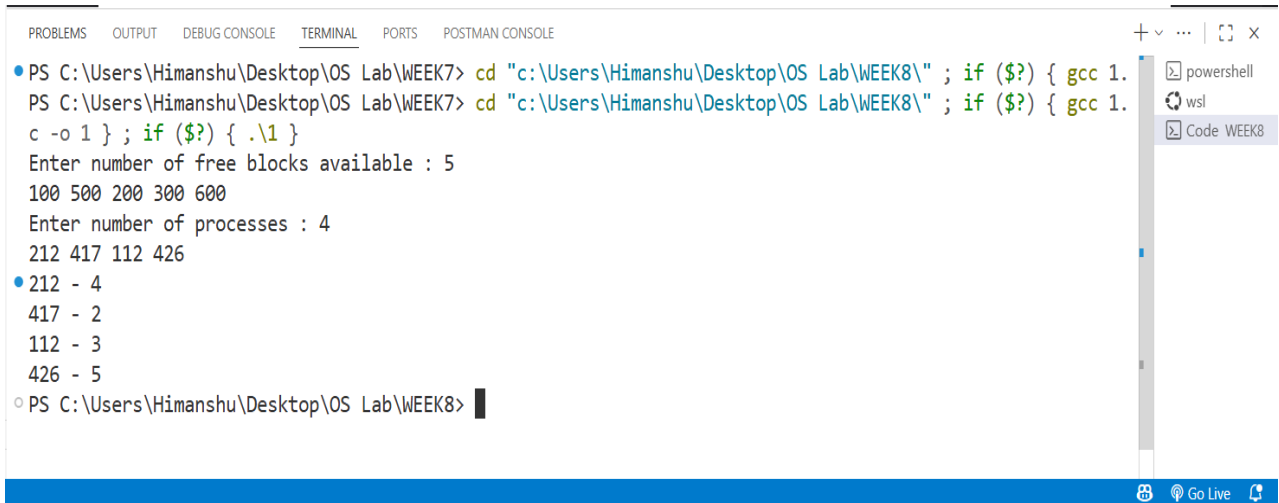
    }
}
if (bestIdx != -1)
{
    allocated[i] = bestIdx;
    block[bestIdx] -= process[i];
}
}

for (int i = 0; i < p; i++)
{
    if (allocated[i] != -1)
        printf("%d - %d\n", process[i], allocated[i] + 1);
    else
        printf("%d no free block allocated\n", process[i]);
}

return 0;
}

```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK7> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK8\" ; if ($?) { gcc 1.
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK7> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK8\" ; if ($?) { gcc 1.
c -o 1 } ; if ($?) { .\1 }
Enter number of free blocks available : 5
100 500 200 300 600
Enter number of processes : 4
212 417 112 426
212 - 4
417 - 2
112 - 3
426 - 5
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK8>
```

Program 2) First Fit – start searching the list from beginning, take the first block whose size is greater than or equal to the requesting program size and allocate it to program.

Source Code:

```
#include <stdio.h>

int main()
{
    int b, p;

    printf("Enter number of free blocks available : ");

    scanf("%d", &b);

    int block[b];

    for (int i = 0; i < b; i++)
        scanf("%d", &block[i]);

    printf("Enter number of processes : ");

    scanf("%d", &p);

    int process[p];

    for (int i = 0; i < p; i++)
        scanf("%d", &process[i]);

    int allocated[p];

    for (int i = 0; i < p; i++)
        allocated[i] = -1;

    for (int i = 0; i < p; i++)
    {
        for (int j = 0; j < b; j++)
        {
            if (block[j] >= process[i])
            {
                allocated[i] = j;
                block[j] -= process[i];
                break;
            }
        }
    }
}
```

```
    }  
}  
  
for (int i = 0; i < p; i++)  
{  
    if (allocated[i] != -1)  
        printf("%d - %d\n", process[i], allocated[i] + 1);  
    else  
        printf("%d no free block allocated\n", process[i]);  
}  
  
return 0;  
}
```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK8> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK8\" ; if ($?) { gcc 2.
c -o 2 } ; if ($?) { .\2 }
Enter number of free blocks available : 5
100 500 200 300 600
Enter number of processes : 4
212 417 112 426
212 - 2
417 - 5
112 - 2
426 no free block allocated
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK8>
```

Program3) Worst Fit – block which is largest among all is allocated for the program.

Source Code:

```
#include <stdio.h>

int main()
{
    int b, p;

    printf("Enter number of free blocks available : ");
    scanf("%d", &b);

    int block[b];

    for (int i = 0; i < b; i++)
        scanf("%d", &block[i]);

    printf("Enter number of processes : ");
    scanf("%d", &p);

    int process[p];

    for (int i = 0; i < p; i++)
        scanf("%d", &process[i]);

    int allocated[p];

    for (int i = 0; i < p; i++)
        allocated[i] = -1;

    for (int i = 0; i < p; i++)
    {
        int worstIdx = -1;

        for (int j = 0; j < b; j++)
        {
```

```

        if (block[j] >= process[i])
        {
            if (worstIdx == -1 || block[j] > block[worstIdx])
                worstIdx = j;
        }
    }
    if (worstIdx != -1)
    {
        allocated[i] = worstIdx;
        block[worstIdx] -= process[i];
    }
}

for (int i = 0; i < p; i++)
{
    if (allocated[i] != -1)
        printf("%d - %d\n", process[i], allocated[i] + 1);
    else
        printf("%d no free block allocated\n", process[i]);
}

return 0;
}

```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK8> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK8\" ; if ($?) { gcc 3.c -o 3 } ; if ($?) { .\3 }
Enter number of free blocks available : 5
100 500 200 300 600
Enter number of processes : 4
212 417 112 426
212 - 5
417 - 2
112 - 5
426 no free block allocated
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK8>
```

WEEK 9-10

Program1) Write a program to implement Sequential file allocation strategies.

Source Code:

```
#include <stdio.h>

#include <stdbool.h>

typedef struct file
{
    char name;

    int start_block;

    int no_of_blocks;
} file;

void main()
{
    bool blocks[1000] = {true};

    int n;

    printf("Enter the number of files:");

    scanf("%d", &n);

    file files[n];

    for (int i = 0; i < n; i++)
    {
        getchar();

        printf("Enter the name of file %d:", i + 1);

        scanf("%c", &files[i].name);

        printf("Enter the starting block of file %d:", i + 1);

        scanf("%d", &files[i].start_block);

        printf("Enter the no of blocks of file %d:", i + 1);

        scanf("%d", &files[i].no_of_blocks);

        int st = files[i].start_block;

        for (int j = 0; j < files[i].no_of_blocks; j++)
```

```

    {
        blocks[st++] = false;
    }
}
char ch;
getchar();
printf("Enter the name of file to be searched:");
scanf("%c", &ch);
bool found = false;
for (int i = 0; i < n; i++)
{
    if (files[i].name == ch)
    {
        printf("File Name    : %c\n", files[i].name);
        printf("Start Block  : %d\n", files[i].start_block);
        printf("No. of Blocks : %d\n", files[i].no_of_blocks);
        printf("Blocks Occupied: ");
        int st = files[i].start_block;
        for (int j = 0; j < files[i].no_of_blocks; j++)
        {
            printf("%d ", st++);
        }
        found = true;
        break;
    }
}
if (!found)
    printf("File not found");
}

```

OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
```

```
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK9-10> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK9-10\" ; if ($?) { gcc 1.c -o 1 } ; i  
f ($?) { .\1 }  
Enter the number of files:3  
Enter the name of file 1:A  
Enter the starting block of file 1:85  
• Enter the no of blocks of file 1:6  
Enter the name of file 2:B  
Enter the starting block of file 2:102  
Enter the no of blocks of file 2:4  
Enter the name of file 3:C  
Enter the starting block of file 3:60  
Enter the no of blocks of file 3:4  
Enter the name of file to be searched:B  
File Name      : B  
Start Block    : 102  
No. of Blocks  : 4  
Blocks Occupied: 102 103 104 105  
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK9-10>
```

Program2) Write a program to implement Linked file allocation strategies.

Source Code:

```
#include <stdio.h>

#include <stdbool.h>

typedef struct file{

    char name;

    int start_block;

    int blocks[100];

    int no_of_blocks;

} file;

void main(){

    bool blocks[1000];

    for (int i = 0; i < 1000; ++i)

        blocks[i] = true; // true means free

    int n;

    printf("Enter number of files: ");

    if (scanf("%d", &n) != 1 || n <= 0){

        printf("Invalid number of files.\n");

        return 1;

    }

    file files[n];

    for (int i = 0; i < n; i++) {

        printf("\nEnter file %d name: ", i + 1);

        if (scanf(" %c", &files[i].name) != 1) {

            printf("Invalid file name input.\n");

            return 1;

        }

        printf("Enter starting block of file %d: ", i + 1);

        if (scanf("%d", &files[i].start_block) != 1) {
```

```

    printf("Invalid start block input.\n");

    return 1;
}

printf("Enter no of blocks in file %d: ", i + 1);
if (scanf("%d", &files[i].no_of_blocks) != 1 ||
    files[i].no_of_blocks < 0 ||
    files[i].no_of_blocks > 100) {
    printf("Invalid number of blocks (0..%d).\n", 100);
    return 1;
}

if (files[i].no_of_blocks == 0) {
    printf("No blocks to read for file %c.\n", files[i].name);
    continue;
}

printf("Enter blocks for file %d: ", i + 1);
for (int j = 0; j < files[i].no_of_blocks; j++) {
    int b;
    while (1) {
        if (scanf("%d", &b) != 1) {
            int c;
            while ((c = getchar()) != EOF && c != '\n')
                printf("Invalid input. Enter a valid block number: ");
            continue;
        }
        if (b < 0 || b >= 1000) {
            printf("Block %d out of range (0..%d). Enter another block: ", b, 1000 - 1);
            continue;
        }
        if (blocks[b] == false) {
            printf("Block %d already occupied, enter another block: ", b);

```

```

        continue;
    }

    files[i].blocks[j] = b;

    blocks[b] = false; // mark occupied

    break;
} } }

char ch;

printf("\nEnter the file name to be searched: ");

if (scanf(" %c", &ch) != 1) {
    printf("Invalid input.\n");
    return 1;
}

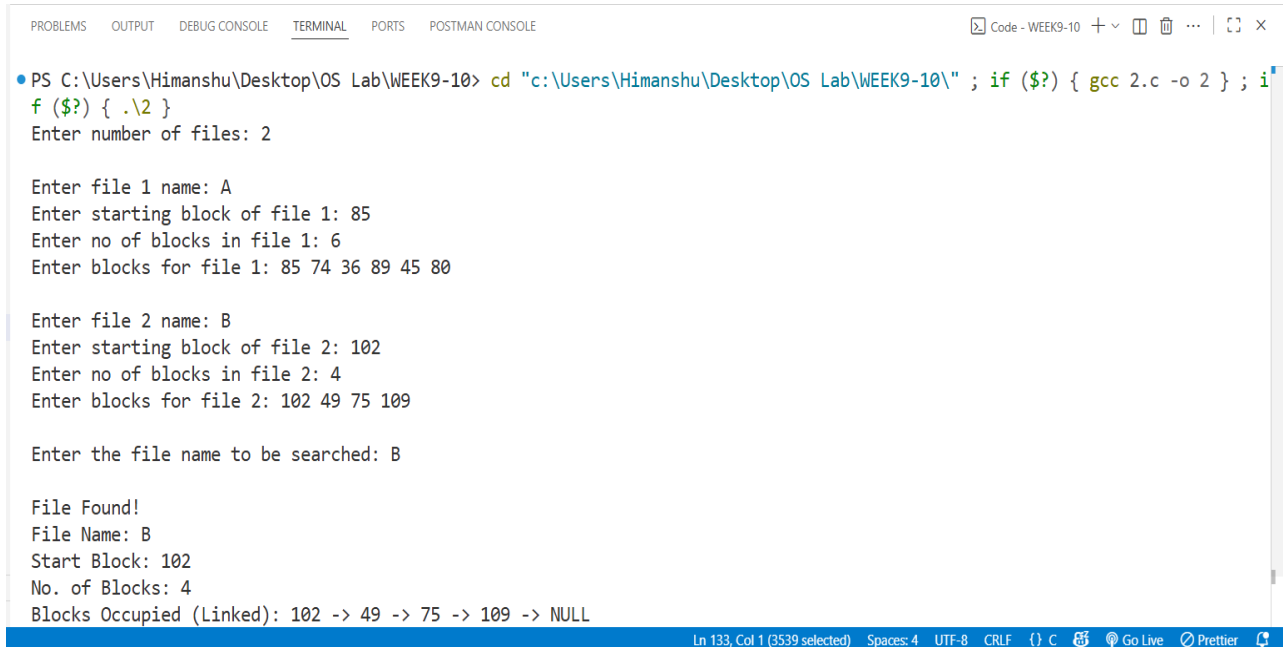
bool found = false;

for (int i = 0; i < n; i++) {
    if (files[i].name == ch) {
        printf("\nFile Found!\n");
        printf("File Name: %c\n", files[i].name);
        printf("Start Block: %d\n", files[i].start_block);
        printf("No. of Blocks: %d\n", files[i].no_of_blocks);
        printf("Blocks Occupied (Linked): ");
        for (int j = 0; j < files[i].no_of_blocks; j++) {
            printf("%d", files[i].blocks[j]);
        }
        printf(" -> NULL\n");
        found = true;
        break;
    } }

if (!found)
    printf("\nFile not found\n");
}

```

OUTPUT



```
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK9-10> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK9-10\" ; if ($?) { gcc 2.c -o 2 } ; if ($?) { .\2 }
Enter number of files: 2

Enter file 1 name: A
Enter starting block of file 1: 85
Enter no of blocks in file 1: 6
Enter blocks for file 1: 85 74 36 89 45 80

Enter file 2 name: B
Enter starting block of file 2: 102
Enter no of blocks in file 2: 4
Enter blocks for file 2: 102 49 75 109

Enter the file name to be searched: B

File Found!
File Name: B
Start Block: 102
No. of Blocks: 4
Blocks Occupied (Linked): 102 -> 49 -> 75 -> 109 -> NULL
```

Program3) Write a program to implement Indexed file allocation strategies.

Source Code:

```
#include <stdio.h>

#include <stdbool.h>

struct file {

    char name;

    int start_block;

    int no_of_blocks;

    int flag;

};

struct block {

    int current;

    int next;

};

void main() {

    int n;

    printf("Enter number of files: ");

    scanf("%d", &n);

    struct file files[n];

    struct block sector[1000];

    for (int i = 0; i < 1000; i++) {

        if (i % 2 == 0) {

            sector[i].current = 100;

            sector[i].next = 100;

        } else {

            sector[i].current = -1;

            sector[i].next = -1;

        }

    }

    for (int i = 0; i < n; i++) {
```

```

    getchar();

    printf("\nEnter file %d name: ", i + 1);

    scanf("%c", &files[i].name);

    printf("Enter starting block of file %c: ", files[i].name);

    scanf("%d", &files[i].start_block);

    printf("Enter number of blocks for file %c: ", files[i].name);

    scanf("%d", &files[i].no_of_blocks);

}

int p = 0;

for (int i = 0; i < n; i++) {

    int req = files[i].no_of_blocks;

    int start_block = files[i].start_block;

    int index = -1;

    if (sector[start_block].current == -1) {

        files[i].flag = 1;

        for (int j = start_block; j < 1000 && req > 0; j++) {

            if (sector[j].current == -1) {

                sector[j].current = p++; // assign a block number

                req--;

                if (index == -1)

                    files[i].start_block = j;

                else

                    sector[index].next = j;

                index = j;

            }

        }

    }

    else {

        printf("enter correct starting block\n");

        files[i].flag = 0;
    }
}

```

```

    }
}
printf("\n\nFile Allocation Table:\n");
printf("File\tStart\tBlocks Linked\n");
printf("-----\n");
for (int i = 0; i < n; i++) {
    if (!files[i].flag)
        continue;
    printf("%c\t%d\t", files[i].name, files[i].start_block);
    int j = files[i].start_block;
    while (j != -1) {
        printf("%d ", j);
        j = sector[j].next;
    }
    printf("\n");
}
}

```

OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Code - WEEK9-10 + - [ ] ... [ ] [ ] x

• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK9-10> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK9-10\" ; if ($?) { gcc 3.c -o 3 } ; if ($?) { .\3 }
Enter number of files: 2

Enter file 1 name: A
Enter starting block of file A: 85
Enter number of blocks for file A: 6
Enter blocks for file A: 85 74 36 89 45 80

Enter file 2 name: B
Enter starting block of file B: 102
Enter number of blocks for file B: 4
Enter blocks for file B: 102 49 75 109

File Allocation Table:
File Name   Start block  No. of blocks  Blocks occupied
-----
A           85           6             85, 74, 36, 89, 45, 80
B          102           4             102, 49, 75, 109

Enter the file name to be searched : B

File Found!
File Name: B
Start Block: 102
No. of Blocks: 4
Blocks Occupied: 102, 49, 75, 109
○ PS C:\Users\Himanshu\Desktop\OS Lab\WEEK9-10> [ ]
```

WEEK 11-12

Program1) Write a program to implement FCFS Disc Scheduling Algorithms.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

int main()
{
    int total_head_movement = 0, initial_pos, n;

    printf("\n Enter the no. of cylinders in Request queue:\n");

    scanf("%d", &n);

    int request_queue[n];

    printf("\n Enter the cylinders no. in Request queue :\n");

    for (int i = 0; i < n; i++)

        scanf("%d", &request_queue[i]);

    printf("\n Enter the initial Position of RW head: ");

    scanf("%d", &initial_pos);

    for (int i = 0; i < n; i++)

    {

        total_head_movement += abs(initial_pos - request_queue[i]);

        initial_pos = request_queue[i];

    }

    printf("\nTotal No. of Head Movements = %d\n", total_head_movement);

    printf("\nAverage head movements = %.2f\n", (float)total_head_movement / n);

    return 0;

}
```

OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Code - WEEK11-12 + - [ ] [ ] ... | [ ] x
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK11-12\" ; if ($?) { gcc 1.c -
  o 1 } ; if ($?) { .\1 }

  Enter the no. of cylinders in Request queue:
9

  Enter the cylinders no. in Request queue :
55 58 60 70 18 90 150 160 184

  Enter the initial Position of RW head: 55

Total No. of Head Movements = 233

Average head movements = 25.89
• PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> █
```

Program2) Write a program to implement SCAN Disc Scheduling Algorithms.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#define LOW 0
#define HIGH 199

Void main() {

    int queue[20];

    int head, max, q_size, temp, sum;

    int dloc; // location of disk (head) arr

    printf("%s\t", "Input no of disk locations");

    scanf("%d", &q_size);

    printf("%s\t", "Enter head position");

    scanf("%d", &head);

    printf("%s\n", "Input elements into disk queue");

    for (int i = 0; i < q_size; i++)

        scanf("%d", &queue[i]);

    queue[q_size] = head; // add RW head into queue

    q_size++;

    for (int i = 0; i < q_size; i++) {

        for (int j = i; j < q_size; j++) {

            if (queue[i] > queue[j]) {

                temp = queue[i];

                queue[i] = queue[j];

                queue[j] = temp;

            }

        }

    }

    max = queue[q_size - 1];
```

```

for (int i = 0; i < q_size; i++) {
    if (head == queue[i]) {
        dloc = i;
        break;
    }
}

if (abs(head - LOW) <= abs(head - HIGH)) {
    for (int j = dloc; j >= 0; j--)
        printf("%d --> ", queue[j]);
    for (int j = dloc + 1; j < q_size; j++)
        printf("%d --> ", queue[j]);
} else
{
    for (int j = dloc + 1; j < q_size; j++)
    {
        printf("%d --> ", queue[j]);
    }
    for (int j = dloc; j >= 0; j--)
    {
        printf("%d --> ", queue[j]);
    }
}

sum = head + max;
printf("\nmovement of total cylinders %d", sum);
return 0;
}

```

OUTPUT

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Code - WEEK11-12 + v [ ] [ ] ... [ ] [ ] X
```

- PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK11-12\" ; if (\$?) { gcc 2.c -o 2 } ; if (\$?) { .\2 }

```
Input no of disk locations      9
Enter head position           55
Input elements into disk queue
55 58 60 70 18 90 150 160 184
55 --> 18 --> 55 --> 58 --> 60 --> 70 --> 90 --> 150 --> 160 --> 184 -->
```

- movement of total cylinders 239

- PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> []

Program3) Write a program to implement C-SCAN Disc Scheduling Algorithms.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

int cmp(const void *a, const void *b) {

    int x = *(int *)a, y = *(int *)b;

    return (x > y) - (x < y);

}

void main() {

    const int START = 0, END = 199;

    int n;

    printf("Enter number of disk requests : ");

    if (scanf("%d", &n) != 1 || n < 0)

        return 0;

    int *req = (int *)malloc(sizeof(int) * n);

    for (int i = 0; i < n; ++i)

        scanf("%d", &req[i]);

    int head;

    printf("Enter initial head position : ");

    if (scanf("%d", &head) != 1)

        head = 0;

    char dir[16];

    printf("Enter initial direction (left/right) : ");

    scanf("%15s", dir);

    qsort(req, n, sizeof(int), cmp);

    int leftCount = 0, rightCount = 0;

    for (int i = 0; i < n; ++i) {

        if (req[i] < head)

            leftCount++;

    }
```

```

else

    rightCount++;

} int *left = (int *)malloc(sizeof(int) * leftCount);

int *right = (int *)malloc(sizeof(int) * rightCount);

int li = 0, ri = 0;

for (int i = 0; i < n; ++i) {

    if (req[i] < head)

        left[li++] = req[i];

    else

        right[ri++] = req[i];

}

long total_seek = 0;

int current = head;

if (dir[0] == 'I' || dir[0] == 'L') {

    for (int i = leftCount - 1; i >= 0; --i) {

        total_seek += abs(current - left[i]);

        current = left[i];

    }

    if (current != START) {

        total_seek += abs(current - START);

        current = START;

    }

    total_seek += abs(END - START);

    current = END;

    for (int i = rightCount - 1; i >= 0; --i) {

        total_seek += abs(current - right[i]);

        current = right[i];

    }

} else {

    for (int i = 0; i < rightCount; ++i) {

```

```

        total_seek += abs(current - right[i]);
        current = right[i];
    }
    if (current != END)
    {
        total_seek += abs(current - END);
        current = END;
    }
    // jump from end to start (counted)
    total_seek += abs(END - START);
    current = START;
    // service left in ascending order
    for (int i = 0; i < leftCount; ++i)
    {
        total_seek += abs(current - left[i]);
        current = left[i];
    }
}

printf("Total seek movement : %ld\n", total_seek);
free(req);
free(left);
free(right);
return 0;
}

```

OUTPUT

```
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> cd "c:\Users\Himanshu\Desktop\OS Lab\WEEK11-12\" ; if ($?) { gcc 3.c -  
o 3 } ; if ($?) { .\3 }  
Enter number of disk requests : 9  
55 58 60 70 18 90 150 160 184  
Enter initial head position : 55  
Enter initial direction (left/right) : right  
Total seek movement : 361  
PS C:\Users\Himanshu\Desktop\OS Lab\WEEK11-12> 
```