Week 6: Process Synchronization

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

14. Write a C program to communicate parent and child process with each other in such a way that whenever child writes something, parent process can read it. Conside mode of communication is through- a)pipe b)message passing c)shared memory

a) **Communication via Pipe**

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

int main() {
   int pipefd[2];
   pid_t pid;
   char buffer[100];

   if (pipe(pipefd) == -1) {
      perror("Pipe failed");
      return 1;
   }

   pid = fork();

   if (pid < 0) {
      perror("Fork failed");
      return 1;
   }

   if (pid == 0) {  // Child process
      close(pipefd[0]);
```

```
        const char *message = "Hello from Child!";
        write(pipefd[1], message, strlen(message) + 1);
        close(pipefd[1]);
    } else {  // Parent process
        close(pipefd[1]);
        read(pipefd[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
        close(pipefd[0]);
    }

    return 0;
}
```

Output:

```
Parent received: Hello from Child!
kartikeyaswarupsharma@Kartikeyas-MacBook-Air Operating System Lab %
```

b) Communication via Message Passing (Message Queue)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>

#define MAX 100

struct msg_buffer {
    long msg_type;
    char msg_text[MAX];
};

int main() {
    key_t key;
    int msgid;
    struct msg_buffer message;

    key = ftok("progfile", 65);
```

```
    msgid = msgget(key, 0666 | IPC_CREAT);

    if (fork() == 0) {
        message.msg_type = 1;
        strcpy(message.msg_text, "Hello from Child!");
        msgsnd(msgid, &message, sizeof(message), 0);
    } else {
        msgrcv(msgid, &message, sizeof(message), 1, 0);
        printf("Parent received: %s\n", message.msg_text);
        msgctl(msgid, IPC_RMID, NULL);
    }

    return 0;
}
```

Output:

```
Parent received: Hello from Child!


=== Code Execution Successful ===
```

c) Communication via Shared Memory

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

#define SHM_SIZE 1024

int main() {
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, SHM_SIZE, 0666 | IPC_CREAT);
    char *str = (char *)shmat(shmid, NULL, 0

    if (fork() == 0) {
        strcpy(str, "Hello from Child!");
        shmdt(str);
```

```
    } else {
        sleep(1);
        printf("Parent received: %s\n", str);
        shmdt(str);
        shmctl(shmid, IPC_RMID, NULL);
    }

    return 0;
}
```

Output:

```
Parent received: Hello from Child!
kartikeyaswarupsharma@Kartikeyas-MacBook-Air Operating System Lab %
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

15. Write a C program to implement the concept of Producer-Consumer problem using semaphores.

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int in = 0, out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void *producer(void *arg) {
   int item;
   for (int i = 0; i < 10; i++) {
      item = rand() % 100;
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("Producer produced: %d at position %d\n", item, in);
      in = (in + 1) % BUFFER_SIZE;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
      sleep(1);
   }
   return NULL;
}

void *consumer(void *arg) {
   int item;
   for (int i = 0; i < 10; i++) {
```

```
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        item = buffer[out];
        printf("Consumer consumed: %d from position %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_mutex_init(&mutex, NULL);
    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:

```
Producer produced: 42 at position 0
Consumer consumed: 42 from position 0
Producer produced: 87 at position 1
Consumer consumed: 87 from position 1
Producer produced: 13 at position 2
Producer produced: 26 at position 3
Consumer consumed: 13 from position 2
Producer produced: 99 at position 4
Consumer consumed: 26 from position 3
Consumer consumed: 99 from position 4
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

16. Write a C program to implement the concept of Dining-Philosopher problem.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define NUM_PHILOSOPHERS 5
pthread_mutex_t chopsticks[NUM_PHILOSOPHERS];
void *philosopher(void *arg) {
   int id = *(int *)arg;

   while (1) {
      printf("Philosopher %d is thinking.\n", id);
      sleep(rand() % 3 + 1);
      printf("Philosopher %d is hungry.\n", id);
      pthread_mutex_lock(&chopsticks[id]); // Pick up left chopstick
      printf("Philosopher %d picked up left chopstick %d.\n", id, id);
      pthread_mutex_lock(&chopsticks[(id + 1) % NUM_PHILOSOPHERS]); //
Pick up right chopstick
      printf("Philosopher %d picked up right chopstick %d.\n", id, (id + 1) %
NUM_PHILOSOPHERS);
      printf("Philosopher %d is eating.\n", id);
      sleep(rand() % 2 + 1);
      pthread_mutex_unlock(&chopsticks[(id + 1) %
NUM_PHILOSOPHERS]); // Put down right chopstick
      printf("Philosopher %d put down right chopstick %d.\n", id, (id + 1) %
NUM_PHILOSOPHERS);
      pthread_mutex_unlock(&chopsticks[id]); // Put down left chopstick
      printf("Philosopher %d put down left chopstick %d.\n", id, id);
   }
   return NULL;
}
int main() {
   pthread_t threads[NUM_PHILOSOPHERS];
   int ids[NUM_PHILOSOPHERS];
   for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
```

```
        pthread_mutex_init(&chopsticks[i], NULL);
        ids[i] = i;
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_create(&threads[i], NULL, philosopher, &ids[i]);
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(threads[i], NULL);
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_destroy(&chopsticks[i]);
    }
    return 0;
}
```

Output:

```
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 2 is hungry.
Philosopher 2 picked up left chopstick 2.
Philosopher 2 picked up right chopstick 3.
Philosopher 2 is eating.
Philosopher 3 is hungry.
Philosopher 1 is hungry.
Philosopher 2 put down right chopstick 3.
Philosopher 2 put down left chopstick 2.
Philosopher 2 is thinking.
Philosopher 3 picked up left chopstick 3.
Philosopher 3 picked up right chopstick 4.
Philosopher 3 is eating.
```

Week 7: Page Replacement Algorithms

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

17. FIFO Page Replacement Program

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

void fifo(int frames, int requests, int pages[]) {
    int frame[frames];
    for (int i = 0; i < frames; i++)
        frame[i] = -1;

    int page_faults = 0, index = 0;

    for (int i = 0; i < requests; i++) {
        int found = 0;

        for (int j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        if (!found) {
            frame[index] = pages[i];
            index = (index + 1) % frames;
            page_faults++;
        }
        printf("Frame after accessing page %d: ", pages[i]);
        for (int k = 0; k < frames; k++) {
            if (frame[k] != -1)
                printf("%d ", frame[k]);
            else
                printf("- ");
        }
```

```
        printf("\n");
    }

    printf("Total number of page faults (FIFO): %d\n", page_faults);
}

int main() {
    int frames, requests;

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

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

    int pages[requests];
    printf("Enter the requested page numbers: ");
    for (int i = 0; i < requests; i++) {
        scanf("%d", &pages[i]);
    }

    fifo(frames, requests, pages);

    return 0;
}
```

Output:

```
Enter number of frames available: 3
Enter number of requests: 12
Enter the requested page numbers: 2 3 2 1 5 2 4 5 3 2 5 2
Frame after accessing page 2: 2 - -
Frame after accessing page 3: 2 3 -
Frame after accessing page 2: 2 3 -
Frame after accessing page 1: 2 3 1
Frame after accessing page 5: 5 3 1
Frame after accessing page 2: 5 2 1
Frame after accessing page 4: 5 2 4
Frame after accessing page 5: 5 2 4
Frame after accessing page 3: 3 2 4
Frame after accessing page 2: 3 2 4
Frame after accessing page 5: 3 5 4
Frame after accessing page 2: 3 5 2
Total number of page faults (FIFO): 9
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

18. LRU Page Replacement Program

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

void lru(int frames, int requests, int pages[]) {
    int frame[frames], age[frames];
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
        age[i] = 0;
    }

    int page_faults = 0;

    for (int i = 0; i < requests; i++) {
        int found = 0;

        for (int j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                found = 1;
                age[j] = i;
                break;
            }
        }

        if (!found) {
            int lru_index = 0;
            for (int j = 1; j < frames; j++) {
                if (age[j] < age[lru_index]) {
                    lru_index = j;
                }
            }

            frame[lru_index] = pages[i];
            age[lru_index] = i;
            page_faults++;
```

```
        }

        // Print the current state of the frame
        printf("Frame after accessing page %d: ", pages[i]);
        for (int k = 0; k < frames; k++) {
            if (frame[k] != -1)
                printf("%d ", frame[k]);
            else
                printf("- ");
        }
        printf("\n");
    }

    printf("Total number of page faults (LRU): %d\n", page_faults);
}

int main() {
    int frames, requests;

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

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

    int pages[requests];
    printf("Enter the requested page numbers: ");
    for (int i = 0; i < requests; i++) {
        scanf("%d", &pages[i]);
    }

    lru(frames, requests, pages);

    return 0;
}
```

Output:

```
Enter number of frames available: 3
Enter number of requests: 12
Enter the requested page numbers: 2 3 2 1 5 2 4 5 3 2 5 2
Frame after accessing page 2: 2 - -
Frame after accessing page 3: 3 - -
Frame after accessing page 2: 3 2 -
Frame after accessing page 1: 3 2 1
Frame after accessing page 5: 5 2 1
Frame after accessing page 2: 5 2 1
Frame after accessing page 4: 5 2 4
Frame after accessing page 5: 5 2 4
Frame after accessing page 3: 5 3 4
Frame after accessing page 2: 5 3 2
Frame after accessing page 5: 5 3 2
Frame after accessing page 2: 5 3 2
Total number of page faults (LRU): 8
```

Week 8: Memory Allocation Techniques

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

19. C Code for Best Fit

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

void bestFit(int blocks[], int b, int processes[], int p) {
    int allocation[p];
    for (int i = 0; i < p; i++) allocation[i] = -1;

    for (int i = 0; i < p; i++) {
        int bestIdx = -1;
        for (int j = 0; j < b; j++) {
            if (blocks[j] >= processes[i]) {
                if (bestIdx == -1 || blocks[j] < blocks[bestIdx]) {
                    bestIdx = j;
                }
            }
        }

        if (bestIdx != -1) {
            allocation[i] = bestIdx + 1; // Store block index (1-based)
            blocks[bestIdx] -= processes[i];
        }
    }

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

```
int main() {
    int b, p;
    printf("Enter number of free blocks available: ");
    scanf("%d", &b);
    int blocks[b];
    printf("Enter sizes of %d blocks: ", b);
    for (int i = 0; i < b; i++) scanf("%d", &blocks[i]);

    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processes[p];
    printf("Enter memory requirements of %d processes: ", p);
    for (int i = 0; i < p; i++) scanf("%d", &processes[i]);

    printf("\nBest Fit Allocation:\n");
    bestFit(blocks, b, processes, p);

    return 0;
}
```

Output:

```
Enter number of free blocks available: 5
Enter sizes of 5 blocks: 100 500 200 300 600
Enter number of processes: 4
Enter memory requirements of 4 processes: 212 417 112 426

Best Fit Allocation:
Process 1 (212) -> Block 4
Process 2 (417) -> Block 2
Process 3 (112) -> Block 3
Process 4 (426) -> Block 5
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

20. C Code for First Fit

```
#include <stdio.h>

void firstFit(int blocks[], int b, int processes[], int p) {
   int allocation[p];
   for (int i = 0; i < p; i++) allocation[i] = -1;

   for (int i = 0; i < p; i++) {
      for (int j = 0; j < b; j++) {
         if (blocks[j] >= processes[i]) {
            allocation[i] = j + 1; // Store block index (1-based)
            blocks[j] -= processes[i];
            break;
         }
      }
   }

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

int main() {
   int b, p;
   printf("Enter number of free blocks available: ");
   scanf("%d", &b);
   int blocks[b];
   printf("Enter sizes of %d blocks: ", b);
   for (int i = 0; i < b; i++) scanf("%d", &blocks[i]);
```

```
    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processes[p];
    printf("Enter memory requirements of %d processes: ", p);
    for (int i = 0; i < p; i++) scanf("%d", &processes[i]);

    printf("\nFirst Fit Allocation:\n");
    firstFit(blocks, b, processes, p);

    return 0;
}
```

Output:

```
Enter number of free blocks available: 5
Enter sizes of 5 blocks: 100 500 200 300 600
Enter number of processes: 4
Enter memory requirements of 4 processes: 212 417 112 426

First Fit Allocation:
Process 1 (212) -> Block 2
Process 2 (417) -> Block 5
Process 3 (112) -> Block 2
Process 4 (426) -> no free block allocated
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

21. C Code for Worst Fit

```
#include <stdio.h>

void worstFit(int blocks[], int b, int processes[], int p) {
    int allocation[p];
    for (int i = 0; i < p; i++) allocation[i] = -1;

    for (int i = 0; i < p; i++) {
        int worstIdx = -1;
        for (int j = 0; j < b; j++) {
            if (blocks[j] >= processes[i]) {
                if (worstIdx == -1 || blocks[j] > blocks[worstIdx]) {
                    worstIdx = j;
                }
            }
        }

        if (worstIdx != -1) {
            allocation[i] = worstIdx + 1; // Store block index (1-based)
            blocks[worstIdx] -= processes[i];
        }
    }

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

int main() {
    int b, p;
```

```
    printf("Enter number of free blocks available: ");
    scanf("%d", &b);
    int blocks[b];
    printf("Enter sizes of %d blocks: ", b);
    for (int i = 0; i < b; i++) scanf("%d", &blocks[i]);

    printf("Enter number of processes: ");
    scanf("%d", &p);
    int processes[p];
    printf("Enter memory requirements of %d processes: ", p);
    for (int i = 0; i < p; i++) scanf("%d", &processes[i]);

    printf("\nWorst Fit Allocation:\n");
    worstFit(blocks, b, processes, p);

    return 0;
}
```

Output:

```
Enter number of free blocks available: 5
Enter sizes of 5 blocks: 100 500 200 300 600
Enter number of processes: 4
Enter memory requirements of 4 processes: 212 417 112 426

Worst Fit Allocation:
Process 1 (212) -> Block 5
Process 2 (417) -> Block 2
Process 3 (112) -> Block 5
Process 4 (426) -> no free block allocated
```

Week 9-10: File Allocation Strategies

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

22. C Code for Contiguous File Allocation Strategy

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

struct File {
   char name[20];
   int startBlock;
   int numBlocks;
};

void contiguousAllocation() {
   int n;
   printf("Enter number of files: ");
   scanf("%d", &n);
   struct File files[n];

   for (int i = 0; i < n; i++) {
      printf("Enter file %d name: ", i + 1);
      scanf("%s", files[i].name);
      printf("Enter starting block of file %d: ", i + 1);
      scanf("%d", &files[i].startBlock);
      printf("Enter number of blocks in file %d: ", i + 1);
      scanf("%d", &files[i].numBlocks);
   }

   char searchFile[20];
   printf("Enter the file name to be searched: ");
   scanf("%s", searchFile);

   for (int i = 0; i < n; i++) {
      if (strcmp(files[i].name, searchFile) == 0) {
         printf("\nFile Name: %s\n", files[i].name);
         printf("Start block: %d\n", files[i].startBlock);
```

```
            printf("Number of blocks: %d\n", files[i].numBlocks);
            printf("Blocks occupied: ");
            for (int j = 0; j < files[i].numBlocks; j++) {
                printf("%d ", files[i].startBlock + j);
            }
            printf("\n");
            return;
        }
    }

    printf("File not found!\n");
}

int main() {
    contiguousAllocation();
    return 0;
}
```

Output:

```
Enter number of files: 3
Enter file 1 name: A
Enter starting block of file 1: 85
Enter no of blocks in file 1: 6
Enter file 2 name: B
Enter starting block of file 2: 102
Enter no of blocks in file 2: 4
Enter file 3 name: C
Enter starting block of file 3: 60
Enter no of blocks in file 3: 4
Enter the file name to be searched: B
```

```
File Name: B
Start block: 102
Number of blocks: 4
Blocks occupied: 102 103 104 105
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

23. C Code for Linked File Allocation Strategy

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

struct File {
    char name[20];
    int startBlock;
    int numBlocks;
    int blocks[50];
};

void linkedAllocation() {
    int n;
    printf("Enter number of files: ");
    scanf("%d", &n);
    struct File files[n];

    for (int i = 0; i < n; i++) {
        printf("Enter file %d name: ", i + 1);
        scanf("%s", files[i].name);
        printf("Enter starting block of file %d: ", i + 1);
        scanf("%d", &files[i].startBlock);
        printf("Enter number of blocks in file %d: ", i + 1);
        scanf("%d", &files[i].numBlocks);
        printf("Enter blocks for file %d: ", i + 1);
        for (int j = 0; j < files[i].numBlocks; j++) {
            scanf("%d", &files[i].blocks[j]);
        }
    }

    char searchFile[20];
    printf("Enter the file name to be searched: ");
    scanf("%s", searchFile);

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

```
        if (strcmp(files[i].name, searchFile) == 0) {
            printf("\nFile Name: %s\n", files[i].name);
            printf("Start block: %d\n", files[i].startBlock);
            printf("Number of blocks: %d\n", files[i].numBlocks);
            printf("Blocks occupied: ");
            for (int j = 0; j < files[i].numBlocks; j++) {
                printf("%d ", files[i].blocks[j]);
            }
            printf("\n");
            return;
        }
    }

    printf("File not found!\n");
}

int main() {
    linkedAllocation();
    return 0;
}
```

Ouptut:

```
Enter number of files: 3
Enter file 1 name: A
Enter starting block of file 1: 85
Enter no of blocks in file 1: 6
Enter file 2 name: B
Enter starting block of file 2: 102
Enter no of blocks in file 2: 4
Enter file 3 name: C
Enter starting block of file 3: 60
Enter no of blocks in file 3: 4
Enter the file name to be searched: B

File Name: B
Start block: 102
Number of blocks: 4
Blocks occupied: 102 49 75 109
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

24. C Code for Indexed File Allocation Strategy

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

struct File {
    char name[20];
    int startBlock;
    int numBlocks;
    int blocks[50];
};

void indexedAllocation() {
    int n;
    printf("Enter number of files: ");
    scanf("%d", &n);
    struct File files[n];

    for (int i = 0; i < n; i++) {
        printf("Enter file %d name: ", i + 1);
        scanf("%s", files[i].name);
        printf("Enter starting block of file %d: ", i + 1);
        scanf("%d", &files[i].startBlock);
        printf("Enter number of blocks in file %d: ", i + 1);
        scanf("%d", &files[i].numBlocks);
        printf("Enter blocks for file %d: ", i + 1);
        for (int j = 0; j < files[i].numBlocks; j++) {
            scanf("%d", &files[i].blocks[j]);
        }
    }

    char searchFile[20];
    printf("Enter the file name to be searched: ");
    scanf("%s", searchFile);

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

```
        if (strcmp(files[i].name, searchFile) == 0) {
            printf("\nFile Name: %s\n", files[i].name);
            printf("Start block: %d\n", files[i].startBlock);
            printf("Number of blocks: %d\n", files[i].numBlocks);
            printf("Blocks occupied: ");
            for (int j = 0; j < files[i].numBlocks; j++) {
                printf("%d ", files[i].blocks[j]);
            }
            printf("\n");
            return;
        }
    }

    printf("File not found!\n");
}

int main() {
    indexedAllocation();
    return 0;
}
```

Output:

```
Enter number of files: 3
Enter file 1 name: A
Enter starting block of file 1: 85
Enter no of blocks in file 1: 6
Enter file 2 name: B
Enter starting block of file 2: 102
Enter no of blocks in file 2: 4
Enter file 3 name: C
Enter starting block of file 3: 60
Enter no of blocks in file 3: 4
Enter the file name to be searched: B

File Name: B
Start block: 102
Number of blocks: 4
Blocks occupied: 102 49 75 109
```

Week 11-12: Disc Scheduling Algorithms

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

25. C Code for FCFS (First Come First Serve)

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

void fcfs(int requests[], int n, int head) {
    int totalSeek = 0;
    int current = head;

    for (int i = 0; i < n; i++) {
        totalSeek += abs(requests[i] - current);
        current = requests[i];
    }

    printf("Total seek movement: %d\n", totalSeek);
}

int main() {
    int n, head;
    printf("Enter number of disk requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the track numbers: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter initial head position: ");
    scanf("%d", &head);

    fcfs(requests, n, head);

    return 0;
}
```

Output:

```
Enter number of disk requests: 9
Enter the track numbers: 55 58 60 70 18 90 150 160 184
Enter initial head position: 50
Total seek movement: 238
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

25. C Code for SCAN (Elevator Algorithm)

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

void scan(int requests[], int n, int head, int direction) {
    int totalSeek = 0, current = head;
    int tracks = 200;


    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (requests[j] > requests[j + 1]) {
                int temp = requests[j];
                requests[j] = requests[j + 1];
                requests[j + 1] = temp;
            }
        }
    }

    if (direction == 1) {
        for (int i = 0; i < n; i++) {
            if (requests[i] >= head) {
                totalSeek += abs(requests[i] - current);
                current = requests[i];
            }
        }
        totalSeek += abs(tracks - 1 - current);
        current = tracks - 1;

        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] < head) {
                totalSeek += abs(current - requests[i]);
                current = requests[i];
            }
        }
```

```
    } else {
        for (int i = n - 1; i >= 0; i--) {
            if (requests[i] <= head) {
                totalSeek += abs(requests[i] - current);
                current = requests[i];
            }
        }
        totalSeek += abs(current - 0);
        current = 0;

        for (int i = 0; i < n; i++) {
            if (requests[i] > head) {
                totalSeek += abs(current - requests[i]);
                current = requests[i];
            }
        }
    }

    printf("Total seek movement: %d\n", totalSeek);
}

int main() {
    int n, head, direction;
    printf("Enter number of disk requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the track numbers: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter initial head position: ");
    scanf("%d", &head);
    printf("Enter direction (1 for high, 0 for low): ");
    scanf("%d", &direction);

    scan(requests, n, head, direction);

    return 0;
}
```

Output:

```
Enter number of disk requests: 9
Enter the track numbers: 55 58 60 70 18 90 150 160 184
Enter initial head position: 50
Enter direction (1 for high, 0 for low): 1
Total seek movement: 330
```

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

26. C Code for C-SCAN (Circular SCAN)

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

void cscan(int requests[], int n, int head) {
    int totalSeek = 0, current = head;
    int tracks = 200;  // Total tracks: 0 to 199


    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (requests[j] > requests[j + 1]) {
                int temp = requests[j];
                requests[j] = requests[j + 1];
                requests[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (requests[i] >= head) {
            totalSeek += abs(requests[i] - current);
            current = requests[i];
        }
    }

    totalSeek += abs(tracks - 1 - current);
    current = 0;
    totalSeek += abs(tracks - 1);

    for (int i = 0; i < n; i++) {
        if (requests[i] < head) {
            totalSeek += abs(current - requests[i]);
            current = requests[i];
        }
```

```
    }

    printf("Total seek movement: %d\n", totalSeek);
}

int main() {
    int n, head;
    printf("Enter number of disk requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the track numbers: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }
    printf("Enter initial head position: ");
    scanf("%d", &head);

    cscan(requests, n, head);

    return 0;
}
```

Output:

```
Enter number of disk requests: 9
Enter the track numbers: 55 58 60 70 18 90 150 160 184
Enter initial head position: 50
Total seek movement: 366
```

Week 13-14: Writing a Shell

Name: Mohit Bisht
Section: J1
Roll No: 37
Course: B.tech 5th Sem
Branch: CSE

27. Code for Simple Shell

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

#define MAX_INPUT_SIZE 1024
#define MAX_ARG_SIZE 100

void tokenize(char *input, char **args) {
    char *token = strtok(input, " \t\n");
    int i = 0;
    while (token != NULL) {
        args[i++] = token;
        token = strtok(NULL, " \t\n");
    }
    args[i] = NULL;
}

void builtin_ls() {
    pid_t pid = fork();
    if (pid == 0) {
        execlp("ls", "ls", NULL);
        perror("ls");
        exit(1);
    } else if (pid > 0) {
        wait(NULL);
    } else {
        perror("Fork failed");
    }
}
```

```
void builtin_mv(char **args) {
    if (args[1] == NULL || args[2] == NULL) {
        fprintf(stderr, "mv: missing file operand\n");
        return;
    }
    if (rename(args[1], args[2]) != 0) {
        perror("mv");
    }
}

void builtin_cat(char **args) {
    if (args[1] == NULL) {
        fprintf(stderr, "cat: missing file operand\n");
        return;
    }
    FILE *src = fopen(args[1], "r");
    if (src == NULL) {
        perror("cat");
        return;
    }

    if (args[2] == NULL) {
        char ch;
        while ((ch = fgetc(src)) != EOF) {
            putchar(ch);
        }
        fclose(src);
    } else {
        FILE *dest = fopen(args[2], "w");
        if (dest == NULL) {
            perror("cat");
            fclose(src);
            return;
        }
        char ch;
        while ((ch = fgetc(src)) != EOF) {
            fputc(ch, dest);
        }
        fclose(src);
        fclose(dest);
    }
}
```

```
void builtin_cd(char **args) {
    if (args[1] == NULL) {
        fprintf(stderr, "cd: missing operand\n");
        return;
    }
    if (chdir(args[1]) != 0) {
        perror("cd");
    }
}

int main() {
    char input[MAX_INPUT_SIZE];
    char *args[MAX_ARG_SIZE];

    while (1) {
        printf("myshell> ");
        fflush(stdout);

        if (fgets(input, MAX_INPUT_SIZE, stdin) == NULL) {
            printf("\n");
            break; // Exit shell on EOF
        }

        tokenize(input, args);

        if (args[0] == NULL) {
            continue; // No command entered
        }

        if (strcmp(args[0], "exit") == 0) {
            break;
        }

        if (strcmp(args[0], "ls") == 0) {
            builtin_ls();
        } else if (strcmp(args[0], "mv") == 0) {
            builtin_mv(args);
        } else if (strcmp(args[0], "cat") == 0) {
            builtin_cat(args);
        } else if (strcmp(args[0], "cd") == 0) {
            builtin_cd(args);
        } else {
```

```
            pid_t pid = fork();
            if (pid == 0) {
                if (execvp(args[0], args) == -1) {
                    perror("Command not found");
                }
                exit(1);
            } else if (pid > 0) {
                wait(NULL);
            } else {
                perror("Fork failed");
            }
        }
    }

    printf("Exiting shell...\n");
    return 0;
}
```

Output:

```
myshell> ls
file1.txt  file2.txt  myprogram.c
```

```
myshell> mv file1.txt newfile.txt
```

```
myshell> cat file2.txt
This is the content of file2.txt.
myshell> cat file2.txt file3.txt
```

```
myshell> pwd
/home/user
myshell> echo Hello, World!
Hello, World!
```

```
myshell> cd /home/user/documents
myshell> cd invalid_directory
cd: No such file or directory
```

```
myshell> exit
Exiting shell...
```

```
myshell> invalid_command
Command not found: No such file or directory
```