Operating System Assignments (I3106)

Code Listing Collection

Ayush Kothadia / Ayush Karanjkhele

November 5, 2025

Contents

1 Assignment 1(A): Address Book Management

File: I3118_1a.txt

```
echo "ENTER ADDRESS BOOK NAME:    "
read fname
touch $fname
echo -e "ADDRESS BOOK CREATED\n"
echo -e "NAME\t\t ID \t DOB\t\tADDRESS\t MOB_NO\t\t SALARY">>$fname
ch=0
while [ $ch -lt '7' ]
do
echo -e "ADDRESS BOOK :\n"
echo -e "1.CREATE ADDRESS BOOK"
 echo -e "2.VIEW ADDRESS BOOK"
echo -e "3.INSERT A RECORD"
echo -e "4.DELETE A RECORD"
echo -e "5.MODIFY A RECORD"
echo -e "6.SEARCH A RECORD"
echo -e "7.EXIT FROM ADDRESS BOOK"
echo -e "ENTER YOUR CHOICE"
read ch
case $ch in
    1)echo -e "enter number of records you want to enter \t"
      read n
      for((i=0;i<$n;i++))
      do
        echo -e "ENTER NAME OF EMPLOYEE\t"
        read ename
        echo -e "ENTER ID OF EMPLOYEE\t"
        read eid
        echo -e "ENTER DOB OF EMPLOYEE\t"
        read edob
        echo -e "ENTER ADDRESS OF EMPLOYEE\t"
        read eadd
        echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
        read emob
        echo -e "ENTER SALARY OF EMPLOYEE\t"
        read esal
        echo -e " $ename \t $eid \t $edob \t $eadd \t $emob \t $esal \n">>$fname
      done
      ;;
      2)
        cat $fname
      ;;
      3)
        echo -e "enter new record"
        echo -e "ENTER NAME OF EMPLOYEE\t"
        read ename
        echo -e "ENTER ID OF EMPLOYEE\t"
        read eid
        echo -e "ENTER DOB OF EMPLOYEE\t"
        read edob
        echo -e "ENTER ADDRESS OF EMPLOYEE\t"
        read eadd
        echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
        read emob
        echo -e "ENTER SALARY OF EMPLOYEE\t"
        read esal
        echo -e " $ename \t $eid \t $edob\t\t $eadd \t $emob \t $esal \n">>$fname
      ;;
      4)
        echo -e "ENTER EMPLOYEE ID TO BE DELETED "
        read eid
        if  grep -w $eid $fname
        then
                grep -wv $eid $fname >>temp
                rm $fname
                mv temp $fname
                echo "RECORD DELETED"

        else
```

```
            echo "RECORD DOES NOT EXIST "
        fi
      ;;
      5)
        echo "ENTER EMPLOYE  ID TO BE MODIFIY"
        read eid
        if  grep -w $eid $fname
            then
                grep -wv $eid $fname >>temp
                rm $fname
                mv temp $fname
                echo -e "enter modified record"
                echo -e "ENTER NAME OF EMPLOYEE\t"
                read ename
                echo -e "ENTER ID OF EMPLOYEE\t"
                read eid
                echo -e "ENTER DOB OF EMPLOYEE\t"
                read edob
                echo -e "ENTER ADDRESS OF EMPLOYEE\t"
                read eadd
                echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
                read emob
                echo -e "ENTER SALARY OF EMPLOYEE\t"
                read esal
                echo -e " $ename \t $eid \t $edob\t\t $eadd \t $emob \t $esal \n">>$fname
            else
                echo "RECORD DOES NOT EXIST "
        fi
      ;;
      6)
        echo -e "ENTER EMPLOYEE ID TO BE SEARCHED "
        read eid
        if  grep  $eid $fname
        then
            grep -w $eid $fname
            echo "RECORD FOUND...!!!"
        else
            echo "RECORD DOES NOT EXIST "
        fi
      ;;
      esac
done
```

2 Assignment 2(A): Process Control (Fork, Wait, Zombie, Orphan)

File: `ass2.a-ayush.txt`

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
void bubble_acs(int a[50],int n)
{
    int i,j,temp;
    for(i=n-1;i>0;i--)
    {
        for(j=0;j<i;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
void bubble_dcs(int a[50],int n)
{
    int i,j,temp;
    for(i=n-1;i>0;i--)
    {
        for(j=0;j<i;j++)
        {
            if(a[j]<a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}

int main(){
int pid;
int i,n,a[50],j;
    printf("\nEnter the Number of elements:");
    scanf("%d",&n);
    printf("\nEnter List of Numbers:\n");
    for(j=0;j<n;j++)
    {
        scanf("%d",&a[j]);
    }
pid = fork();
if(pid==0)
{

        printf("\n");


 printf("I am child and my id is %d \n",getpid());
 printf("I am parent and my id is %d\n",getppid());
}
else
{
 system("ps -el|grep Z");
   sleep(5);
   int i= wait(0);

   printf("The terminated child's pid is %d \n",i);
   printf("I am parent and my id is %d \n",getpid());
   printf("I am parent's parent and my id is %d",getppid());
   bubble_dcs(a,n);
        printf("\nList of Numbers in Descending Order:\n");
        for(j=0;j<n;j++)
        {
            printf("%d\n",a[j]);
        }
    }

return 0;
}
```

Listing 1:

3 Assignment 2(B): Process Control (Fork, Execve, Sort/Search)

File: ass2.b-ayush.txt - Parent Process

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

void sort_ascending(int *arr, int size) {
    int temp;
    for (int pass = 0; pass < size - 1; pass++) {
        for (int i = 0; i < size - pass - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
    }

    printf("\n[Parent] Sorted array in ascending order:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Usage: %s <child_program_path> <num1> <num2> ...\n", argv[0]);
        return 1;
    }

    int count = argc - 2;
    int values[count];


    for (int i = 0; i < count; i++) {
        values[i] = atoi(argv[i + 2]);
    }


    sort_ascending(values, count);


    char *child_args[count + 2];
    child_args[0] = argv[1];

    for (int i = 0; i < count; i++) {
        child_args[i + 1] = malloc(12);
        sprintf(child_args[i + 1], "%d", values[i]);
    }
    child_args[count + 1] = NULL;


    pid_t pid = fork();

    if (pid == 0) {

        printf("\n[Child] PID: %d | PPID: %d\n", getpid(), getppid());
        execve(argv[1], child_args, NULL);
        perror("execve failed");
        exit(EXIT_FAILURE);
    } else if (pid > 0) {

        wait(NULL);
        printf("\n[Parent] Child finished. PID: %d\n", pid);
    } else {
        perror("fork failed");
        return 1;
    }


    for (int i = 1; i <= count; i++) {
        free(child_args[i]);
    }

    return 0;
}
```

Listing 2:

File: ass2.b-ayush.txt - Child Process

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

void sort_descending(int *arr, int size) {
    int temp;
    for (int pass = 0; pass < size - 1; pass++) {
        for (int i = 0; i < size - pass - 1; i++) {
            if (arr[i] < arr[i + 1]) {
                temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
    }

    printf("\n[Child] Sorted array in descending order:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <sorted_numbers>\n", argv[0]);
        return 1;
    }

    int size = argc - 1;
    int numbers[size];

    for (int i = 0; i < size; i++) {
        numbers[i] = atoi(argv[i + 1]);
    }

    printf("\n[Child] Executing child process with PID: %d\n", getpid());

    sort_descending(numbers, size);

    return 0;
}
```

Listing 3:

4 Assignment 3: CPU Scheduling (SRTF, Round Robin)

File: `Ass3.c`

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

#define MAX 20
#define QSIZE (MAX * 10)


typedef struct {
    char name[16];
    int AT;
    int BT;
    int rem;
    int WT;
    int TAT;
    int PID;
} process;


typedef struct {
    char name[16];
    int start;
    int end;
} gantt_event;

/* prototypes */
void get_PCB(process p[], int *n);
void disp_table(process p[], int n);
float cal_avgwt(process p[], int n);
float cal_avgtat(process p[], int n);

void SJF_Preemptive(process p[], int n);
void RoundRobin(process p[], int n, int tq);

void menu() {
    printf("\n\t**** CPU SCHEDULING MENU ****\n");
    printf("1. SJF (Preemptive - SRTF)\n");
    printf("2. Round Robin\n");
    printf("3. Exit\n");
    printf("Enter choice: ");
}

int main() {
    process P[MAX];
    int n, ch, tq;


    get_PCB(P, &n);


    while (1) {
        menu();
        if (scanf("%d", &ch) != 1) {

            break;
        }


        process P_copy[MAX];
        memcpy(P_copy, P, sizeof(process) * n);

        if (ch == 1) {
            SJF_Preemptive(P_copy, n);
            disp_table(P_copy, n);
            printf("\nAverage WT : %.2f\n", cal_avgwt(P_copy, n));
            printf("Average TAT: %.2f\n", cal_avgtat(P_copy, n));
        } else if (ch == 2) {
            printf("Enter Time Quantum: ");
            if (scanf("%d", &tq) != 1) break;
            if (tq <= 0) {
                printf("Time quantum must be >= 1\n");
                continue;
            }
            RoundRobin(P_copy, n, tq);
            disp_table(P_copy, n);
            printf("\nAverage WT : %.2f\n", cal_avgwt(P_copy, n));
            printf("Average TAT: %.2f\n", cal_avgtat(P_copy, n));
        } else {
            break;
        }
    }

    return 0;
}


void get_PCB(process p[], int *n) {
    int i;
    printf("Enter total no of processes: ");
    if (scanf("%d", n) != 1) *n = 0;

    for (i = 0; i < *n; i++) {
        printf("\nProcess %d name (e.g., P%d): ", i + 1, i + 1);
        scanf("%s", p[i].name);
        printf("Arrival Time       : ");
        scanf("%d", &p[i].AT);
        printf("Burst Time         : ");
        scanf("%d", &p[i].BT);
        p[i].rem = p[i].BT;
        p[i].WT = p[i].TAT = 0;
        p[i].PID = i;
    }
}

void disp_table(process p[], int n) {
    int i;
    printf("\n\n P_NAME \t AT \t BT \t WT \t TAT\n");
    for (i = 0; i < n; i++) {
        printf(" %-8s \t %d \t %d \t %d \t %d\n",
               p[i].name, p[i].AT, p[i].BT, p[i].WT, p[i].TAT);
    }
}

float cal_avgwt(process p[], int n) {
    float s = 0.0f;
    int i;
    for (i = 0; i < n; i++) s += p[i].WT;
    return s / n;
}

float cal_avgtat(process p[], int n) {
    float s = 0.0f;
    int i;
    for (i = 0; i < n; i++) s += p[i].TAT;
    return s / n;
}


void SJF_Preemptive(process p[], int n) {
    int i, completed = 0, time = 0;
    int min_idx;
    int prev_idx = -1;
    gantt_event events[1000];
    int evcount = 0;


    for (i = 0; i < n; i++) {
        p[i].rem = p[i].BT;
        p[i].WT = p[i].TAT = 0;
    }


    int earliest = p[0].AT;
    for (i = 1; i < n; i++) if (p[i].AT < earliest) earliest = p[i].AT;
    time = earliest;

    while (completed < n) {

        min_idx = -1;
        for (i = 0; i < n; i++) {
            if (p[i].rem > 0 && p[i].AT <= time) {
                if (min_idx == -1 || p[i].rem < p[min_idx].rem ||
                    (p[i].rem == p[min_idx].rem && p[i].AT < p[min_idx].AT)) {
                    min_idx = i;
                }
            }
        }

        if (min_idx == -1) {

            int next_arr = -1;
            for (i = 0; i < n; i++) {

                if (p[i].rem > 0) {
                    if (next_arr == -1 || (p[i].AT > time && p[i].AT < next_arr) || next_arr == -1)
                        {
                        next_arr = p[i].AT;
                    }
                }
            }


            if (next_arr != -1 && next_arr > time) {

                if (evcount > 0 && strcmp(events[evcount - 1].name, "IDLE") == 0) {
                    events[evcount - 1].end = next_arr;
                } else {
                    strcpy(events[evcount].name, "IDLE");
                    events[evcount].start = time;
                    events[evcount].end = next_arr;
                    evcount++;
                }
                time = next_arr;
            } else {

                break;
            }
            continue;
        }


        if (prev_idx != min_idx) {

            strcpy(events[evcount].name, p[min_idx].name);
            events[evcount].start = time;
            events[evcount].end = time;
            evcount++;
        }


        p[min_idx].rem--;
        time++;
        events[evcount - 1].end = time;

        if (p[min_idx].rem == 0) {
            completed++;

            p[min_idx].TAT = time - p[min_idx].AT;
            p[min_idx].WT = p[min_idx].TAT - p[min_idx].BT;
        }
        prev_idx = min_idx;
    }


    printf("\n\nGANTT CHART (SJF Preemptive):\n|");
    for (i = 0; i < evcount; i++) {
        printf(" %s |", events[i].name);
    }
    printf("\n%d", events[0].start);
    for (i = 0; i < evcount; i++) {
        printf("     %d", events[i].end);
    }
    printf("\n");
}


void RoundRobin(process p[], int n, int tq) {
    int i;

    for (i = 0; i < n; i++) {
        p[i].rem = p[i].BT;
        p[i].WT = p[i].TAT = 0;
    }

    int time = 0, completed = 0;
    int queue[QSIZE];
    int head = 0, tail = 0, countq = 0;

    int in_queue[MAX];
    memset(in_queue, 0, sizeof(in_queue));
    gantt_event events[1000];
    int evcount = 0;


    int earliest = p[0].AT;
    for (i = 1; i < n; i++) if (p[i].AT < earliest) earliest = p[i].AT;
    time = earliest;


    for (i = 0; i < n; i++) {
        if (p[i].AT <= time && !in_queue[i]) {
            queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
        }
    }

    while (completed < n) {
        if (countq == 0) {

            int next_arr = -1;
            for (i = 0; i < n; i++) {

                if (p[i].rem > 0) {

                    if (p[i].AT > time) {
                        if (next_arr == -1 || p[i].AT < next_arr) next_arr = p[i].AT;
                    }
                }
            }

            if (next_arr != -1 && next_arr > time) {

                if (evcount > 0 && strcmp(events[evcount - 1].name, "IDLE") == 0) {
                    events[evcount - 1].end = next_arr;
                } else {
                    strcpy(events[evcount].name, "IDLE");
                    events[evcount].start = time;
                    events[evcount].end = next_arr;
                    evcount++;
                }
                time = next_arr;
            } else if (next_arr == -1) {

                break;
            }


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

                if (!in_queue[i] && p[i].rem > 0 && p[i].AT <= time) {
                    queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
                }
            }
            continue;
        }


        int idx = queue[head]; head = (head + 1) % QSIZE; countq--; in_queue[idx] = 0;


        strcpy(events[evcount].name, p[idx].name);
        events[evcount].start = time;
        events[evcount].end = time;
        evcount++;


        int run = (p[idx].rem < tq) ? p[idx].rem : tq;


        int t;
        for (t = 0; t < run; t++) {
            p[idx].rem--;
            time++;
            events[evcount - 1].end = time;


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

                if (!in_queue[i] && p[i].rem > 0 && p[i].AT == time) {
                    queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
                }
            }
        }

        if (p[idx].rem == 0) {

            completed++;
            p[idx].TAT = time - p[idx].AT;
            p[idx].WT  = p[idx].TAT - p[idx].BT;
        } else {

            queue[tail] = idx; tail = (tail + 1) % QSIZE; countq++; in_queue[idx] = 1;
        }
    }


    printf("\n\nGANTT CHART (Round Robin, TQ=%d):\n|", tq);
    for (i = 0; i < evcount; i++) {
        printf(" %s |", events[i].name);
    }
    printf("\n%d", events[0].start);
    for (i = 0; i < evcount; i++) {
        printf("     %d", events[i].end);
    }
    printf("\n");
}
```

Listing 4:

5 Assignment 4(A): Producer-Consumer (Counting Semaphores)

File: `Ass4_pro.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <pthread.h>
#include <semaphore.h>

#define BUFFER_SIZE 20

void *producer(void *arg);
void *consumer(void *arg);

typedef struct {
    int buff[BUFFER_SIZE];
    sem_t full;
    sem_t empty;
} shared;

shared sh;
int in = 0;
int out = 0;
sem_t mutex;
int next_item = 0;

int main()
{
    pthread_t ptid1, ptid2, ctid1;




    sem_init(&sh.empty, 0, BUFFER_SIZE);
    sem_init(&sh.full, 0, 0);


    sem_init(&mutex, 0, 1);


    pthread_create(&ptid1, NULL, producer, NULL);
    pthread_create(&ptid2, NULL, producer, NULL);
    pthread_create(&ctid1, NULL, consumer, NULL);


    pthread_join(ptid1, NULL);
    pthread_join(ptid2, NULL);
    pthread_join(ctid1, NULL);

    return 0;
}

void *producer(void *arg)
{
    (void)arg;

    while (1)
    {

        int item = next_item++;


        sem_wait(&sh.empty);


        sem_wait(&mutex);


        sh.buff[in] = item;

        in = (in + 1) % BUFFER_SIZE;


        pid_t tid = (pid_t) syscall(SYS_gettid);
        printf("producer thread id: %d, produced item: %d\n", (int)tid, item);


        sem_post(&mutex);


        sem_post(&sh.full);


        sleep(2);
    }

    return NULL;
}

void *consumer(void *arg)
{
    (void)arg;

    while (1)
    {

        sem_wait(&sh.full);


        sem_wait(&mutex);


        int item = sh.buff[out];

        out = (out + 1) % BUFFER_SIZE;


        pid_t tid = (pid_t) syscall(SYS_gettid);
        printf("consumer thread id: %d, consumed item: %d\n", (int)tid, item);


        sem_post(&mutex);


        sem_post(&sh.empty);


        sleep(2);
    }

    return NULL;
}
```

Listing 5:

6 Assignment 4(B): Reader-Writer Problem

File: Ass4_read.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<sys/types.h>

void *reader(void *argp);
void *writer(void *argp);

int buf;

int getbuff(){
    int temp;
    printf("\nEnter the item in buffer :- ");
    scanf("%d",&temp);
    return temp;
}

void readbuff(int buf){
    printf("\nItem READ from buffer : %d\n",buf);
}

pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t wrt=PTHREAD_MUTEX_INITIALIZER;

int read_count=0;
int flag=0;

void *writer(void *argp)
{
    while(1)
    {
        pthread_mutex_lock(&wrt);
        if(flag==0)
        {
            buf=getbuff();
            flag=1;
        }
        pthread_mutex_unlock(&wrt);
    }
}

void *reader(void *argp)
{
    while(1)
    {

        pthread_mutex_lock(&mutex1);
        read_count++;
        if(read_count==1)
        {
            pthread_mutex_lock(&wrt);
        }
        pthread_mutex_unlock(&mutex1);


        if(flag==1)
        {
            readbuff(buf);
            sleep(2);
            flag=0;
        }


        pthread_mutex_lock(&mutex1);
        read_count--;
        if(read_count==0)
        {
            pthread_mutex_unlock(&wrt);
        }
        pthread_mutex_unlock(&mutex1);
    }
}

int main()
{
    pthread_t tid1,tid2,tid3;


    pthread_create(&tid1,NULL,writer,NULL);
    pthread_create(&tid2,NULL,reader,NULL);
    pthread_create(&tid3,NULL,reader,NULL);


    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    pthread_join(tid3,NULL);

    return 0;
}
```

Listing 6:

7 Assignment 5: Deadlock Avoidance (Banker’s Algorithm)

File: Ass5.c

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

struct process {
    int all[6], max[6], need[6], finished, request[6];
} p[10];

int avail[6], sseq[10], ss = 0, check1 = 0, check2 = 0;
int n, pid, work[6];
int nor, nori;

int safeseq(void);

void wait_for_enter(void) {
    int c;
    printf("\n\nPress Enter to continue...");

    do { c = getchar(); } while (c != '\n' && c != EOF);

    getchar();
}

int main(void)
{
    int ch, i = 0, j = 0, k, pid, ch1;
    int violationcheck = 0, waitcheck = 0;

    do {

        printf("\n\n\t 1. Input");
        printf("\n\n\t 2. New Request");
        printf("\n\n\t 3. Safe State or Not");
        printf("\n\n\t 4. Print");
        printf("\n\n\t 5. Exit");
        printf("\n\n\t Enter your choice : ");

        if (scanf("%d", &ch) != 1) {
            printf("\nInvalid input. Exiting.\n");
            return 0;
        }

        switch (ch) {
            case 1:
                printf("\n\n\t Enter number of processes : ");
                scanf("%d", &n);

                printf("\n\n\t Enter the Number of Resources : ");
                scanf("%d", &nor);

                printf("\n\n\t Enter the Available Resources : ");
                for (k = 0; k < nor; k++) {
                    printf("\n\n\t For Resource type %d : ", k);
                    scanf("%d", &avail[k]);
                }


                for (i = 0; i < n; i++) {
                    for (k = 0; k < nor; k++) {
                        p[i].max[k] = 0;
                        p[i].all[k] = 0;
                        p[i].need[k] = 0;
                        p[i].request[k] = 0;
                    }
                    p[i].finished = 0;
                }


                for (i = 0; i < n; i++) {
                    printf("\n\n\t Enter Max and Allocated resources for P%d : ", i);
                    for (j = 0; j < nor; j++) {
                        printf("\n\n\t Enter the Max of resource %d : ", j);
                        scanf("%d", &p[i].max[j]);
                        printf("\n\n\t Allocation of resource %d    : ", j);
                        scanf("%d", &p[i].all[j]);
                        if (p[i].all[j] > p[i].max[j]) {
                            printf("\n\n\t Allocation should be <= Max, please re-enter this resource.");
                            j--;
                        } else {
                            p[i].need[j] = p[i].max[j] - p[i].all[j];
                            avail[j] -= p[i].all[j];
                        }
                    }
                }
                break;

            case 2:
                violationcheck = 0;
                waitcheck = 0;

                printf("\n\n\t Requesting process id (0..%d): ", n-1);
                scanf("%d", &pid);

                if (pid < 0 || pid >= n) {
                    printf("\n\n\t Invalid PID.");
                    break;
                }

                for (j = 0; j < nor; j++) {
                    printf("\n\n\t Number of Request for resource %d : ", j);
                    scanf("%d", &p[pid].request[j]);
                    if (p[pid].request[j] > p[pid].need[j])
                        violationcheck = 1;
                    if (p[pid].request[j] > avail[j])
                        waitcheck = 1;
                }

                if (violationcheck == 1) {
                    printf("\n\n\t The Process Exceeds its Max Need: Terminated");
                } else if (waitcheck == 1) {
                    printf("\n\n\t Lack of Resources : Process State - Wait");
                } else {

                    for (j = 0; j < nor; j++) {
                        avail[j]    -= p[pid].request[j];
                        p[pid].all[j]  += p[pid].request[j];
                        p[pid].need[j] -= p[pid].request[j];
                    }
                    ch1 = safeseq();
                    if (ch1 == 0) {
                        printf("\n\n\t Granting leads to Unsafe state: Request Denied");

                        for (j = 0; j < nor; j++) {
                            avail[j]    += p[pid].request[j];
                            p[pid].all[j]  -= p[pid].request[j];
                            p[pid].need[j] += p[pid].request[j];
                        }
                    } else {
                        printf("\n\n\t Request Committed ");
                    }
                }
                break;

            case 3:
                if (safeseq() == 1)
                    printf("\n\n\t The System is in SAFE state ");
                else
                    printf("\n\n\t The System is NOT in safe state ");
                break;

            case 4:
```

```c
                printf("\n\n\t Number of processes : %d", n);
                printf("\n\n\t Number of Resources : %d", nor);

                printf("\n\n\t Pid \t   Max \t   Allocated \t Need ");
                for (i = 0; i < n; i++) {
                    int r;
                    printf("\n\n\t  P%d : ", i);

                    for (r = 0; r < nor; r++) printf(" %d ", p[i].max[r]);
                    printf("\t");
                    for (r = 0; r < nor; r++) printf(" %d ", p[i].all[r]);
                    printf("\t");
                    for (r = 0; r < nor; r++) printf(" %d ", p[i].need[r]);
                }

                printf("\n\n\t Available : ");
                for (i = 0; i < nor; i++) printf(" %d ", avail[i]);
                break;

            case 5:
                printf("\n\nExiting...\n");
                return 0;

            default:
                printf("\n\n\t Invalid choice.");
        }


        wait_for_enter();

    } while (ch != 5);

    return 0;
}

int safeseq(void)
{
    int i, j, t;
    ss = 0;


    for (j = 0; j < nor; j++)
        work[j] = avail[j];

    for (j = 0; j < n; j++)
        p[j].finished = 0;


    for (t = 0; t < n; t++) {
        for (j = 0; j < n; j++) {
            if (p[j].finished == 0) {
                check1 = 0;
                for (i = 0; i < nor; i++)
                    if (p[j].need[i] <= work[i])
                        check1++;
                if (check1 == nor) {
                    for (i = 0; i < nor; i++)
                        work[i] += p[j].all[i];
                    p[j].finished = 1;
                    sseq[ss++] = j;
                }
            }
        }
    }

    check2 = 0;
    for (i = 0; i < n; i++)
        if (p[i].finished == 1)
            check2++;

    if (check2 >= n) {
        printf("\n\n\t The system is in SAFE state\n\t Safe sequence : ");
        for (i = 0; i < n; i++)
```

```
            printf("P%d%s", sseq[i], (i < n - 1) ? " -> " : "");
        return 1;
    } else {
        printf("\n\n\t The system is NOT in safe state");
        return 0;
    }
}
```

Listing 7:

8 Assignment 6: Page Replacement Algorithms (FIFO, LRU, OPTIMAL)

File: `Ass_6.c`

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

#define MAX_PAGES  200
#define MAX_FRAME  20

static int in_frames(char frames[], int fsz, char p) {
    for (int i = 0; i < fsz; i++) if (frames[i] == p) return i;
    return -1;
}

static void print_row(char req, char frames[], int used, int hit) {
    printf("\n\t%c\t\t", req);
    for (int i = 0; i < used; i++) {
        printf("%c   ", frames[i] ? frames[i] : '-');
    }
    printf("\t\t%s", hit ? "HIT" : "FAULT");
}

void fifo(char pages[], int n, int frameSize) {
    char frames[MAX_FRAME] = {0};
    int used = 0, next = 0, faults = 0;

    printf("\nData Requested\tFrame contents\t    Status\n
        ===============================================");
    for (int i = 0; i < n; i++) {
        char p = pages[i];
        int hit = (in_frames(frames, used, p) != -1);

        if (!hit) {
            faults++;
            if (used < frameSize) {
                frames[used++] = p;
            } else {
                frames[next] = p;
                next = (next + 1) % frameSize;
            }
        }
        print_row(p, frames, used, hit);
    }
    printf("\n\n===============================================\n");
    printf("\nTotal no. of Page Faults: %d\n\n", faults);
}

void optimal(char pages[], int n, int frameSize) {
    char frames[MAX_FRAME] = {0};
    int used = 0, faults = 0;

    printf("\nData Requested\tFrame contents\t    Status\n
        ===============================================");
    for (int i = 0; i < n; i++) {
        char p = pages[i];
        int pos = in_frames(frames, used, p);
        int hit = (pos != -1);

        if (!hit) {
            faults++;
            if (used < frameSize) {
                frames[used++] = p;
            } else {

                int far_idx = 0, far_dist = -1;
                for (int f = 0; f < used; f++) {
                    int dist = 1000000;
                    for (int j = i + 1; j < n; j++) {
                        if (pages[j] == frames[f]) {
                            dist = j - i;
                            break;
                        }
                    }
                    if (dist > far_dist) {
                        far_dist = dist;
                        far_idx = f;
                    }
                }
                frames[far_idx] = p;
            }
        }
        print_row(p, frames, used, hit);
    }
    printf("\n\n=============================================\n");
    printf("\nTotal no. of Page Faults: %d\n\n", faults);
}

void lru(char pages[], int n, int frameSize) {
    char frames[MAX_FRAME] = {0};
    int last_used[MAX_FRAME] = {0};
    int used = 0, faults = 0;

    printf("\nData Requested\tFrame contents\t    Status\n=============================================");
    for (int time = 0; time < n; time++) {
        char p = pages[time];
        int idx = in_frames(frames, used, p);
        int hit = (idx != -1);

        if (hit) {
            last_used[idx] = time;
        } else {
            faults++;
            if (used < frameSize) {
                frames[used] = p;
                last_used[used] = time;
                used++;
            } else {

                int lru_idx = 0, lru_time = last_used[0];
                for (int f = 1; f < used; f++) {
                    if (last_used[f] < lru_time) {
                        lru_time = last_used[f];
                        lru_idx = f;
                    }
                }
                frames[lru_idx] = p;
                last_used[lru_idx] = time;
            }
        }
        print_row(p, frames, used, hit);
    }
    printf("\n\n=============================================\n");
    printf("\nTotal no. of Page Faults: %d\n\n", faults);
}

int main(void) {
    char line[512];
    char pages[MAX_PAGES];
    int n = 0, frameSize, ch;

    printf("Enter the reference string (chars, spaces allowed): ");
    if (!fgets(line, sizeof(line), stdin)) return 0;


    for (int i = 0; line[i] && n < MAX_PAGES; i++) {
        if (!isspace((unsigned char)line[i])) {
            pages[n++] = line[i];
        }
    }

    printf("Enter the size of the frame: ");
    if (scanf("%d", &frameSize) != 1 || frameSize <= 0 || frameSize > MAX_FRAME) {
        printf("Invalid frame size.\n");
        return 0;
    }

    do {
        printf("\nMENU\n====\n1. FIFO\n2. Least Recently Used (LRU)\n3. Optimal\n4. Exit\n\nYour
            Choice: ");
        if (scanf("%d", &ch) != 1) break;
        switch (ch) {
            case 1: fifo(pages, n, frameSize); break;
            case 2: lru(pages, n, frameSize); break;
            case 3: optimal(pages, n, frameSize); break;
            case 4: return 0;
            default: printf("\nInvalid choice! Please try again!\n");
        }
    } while (ch != 4);

    return 0;
}
```

Listing 8:

9 Assignment 7(A): Full-Duplex IPC (Named Pipes/FIFO)

File: Ass7_sender.c - Sender Process (Process 1)

```c
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#define Max_Buff 1024

int main()
{
    int fd1, fd2, c = 0;
    char *myfifo1 = "myfifo1";
    char *myfifo2 = "myfifo2";
    char buff[Max_Buff], ch;

    mkfifo(myfifo1, 0777);
    mkfifo(myfifo2, 0777);

    printf("\nEnter the string (end with #):\n");

    while((ch = getchar()) != '#')
        buff[c++] = ch;
    buff[c] = '\0';

    fd1 = open(myfifo1, O_WRONLY);
    write(fd1, buff, c + 1);
    close(fd1);

    fd2 = open(myfifo2, O_RDONLY);
    read(fd2, buff, Max_Buff);
    printf("\nContents of file:\n%s\n", buff);
    close(fd2);

    return 0;
}
```

Listing 9:

File: `Ass7_sender.c` - Receiver Process (Process 2)

```c
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#define MAX_BUF 1024

int main()
{
    int words = 0, lines = 0, chars = 0, i = 0;
    char buf[MAX_BUF];
    int fd, fd1;
    FILE *fp;

    char *myfifo1 = "myfifo1";
    char *myfifo2 = "myfifo2";

    mkfifo(myfifo1, 0777);
    mkfifo(myfifo2, 0777);

    fd = open(myfifo1, O_RDONLY);
    read(fd, buf, MAX_BUF);
    close(fd);

    printf("\nMessage received:\n%s\n", buf);

    while (buf[i] != '\0')
    {
        if (buf[i] == ' ' || buf[i] == '\n')
            words++;
        else
            chars++;

        if (buf[i] == '\n')
            lines++;

        i++;
    }

    words++;
    lines++;

    fp = fopen("a.txt", "w");
    fprintf(fp, "\nNo. of lines are : %d\n", lines);
    fprintf(fp, "\nNo. of words are : %d\n", words);
    fprintf(fp, "\nNo. of chars are : %d\n", chars);
    fclose(fp);

    fd1 = open(myfifo2, O_WRONLY);
    dprintf(fd1, "\nNo. of lines are : %d\nNo. of words are : %d\nNo. of chars are : %d\n",
            lines, words, chars);
    close(fd1);

    return 0;
}
```

Listing 10:

10 Assignment 7(B): Inter-process Communication (Shared Memory)

File: `Ass_7B_shared_ayush.c`

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

#define SHM_SIZE 4096

int main(void) {

    int shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid < 0) {
        perror("shmget");
        return 1;
    }
    printf("Shared memory created. ID = %d\n", shmid);


    void *addr = shmat(shmid, NULL, 0);
    if (addr == (void *)-1) {
        perror("shmat");

        shmctl(shmid, IPC_RMID, NULL);
        return 1;
    }
    char *shm_buf = (char *)addr;


    char input[SHM_SIZE];
    printf("[Server] Enter the message to write: ");
    fflush(stdout);

    if (!fgets(input, sizeof(input), stdin)) {
        fprintf(stderr, "Failed to read input.\n");
        shmdt(addr);
        shmctl(shmid, IPC_RMID, NULL);
        return 1;
    }

    input[strcspn(input, "\n")] = '\0';


    strncpy(shm_buf, input, SHM_SIZE - 1);
    shm_buf[SHM_SIZE - 1] = '\0';
    printf("[Server] Message written to shared memory.\n");


    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        shmdt(addr);
        shmctl(shmid, IPC_RMID, NULL);
        return 1;
    }

    if (pid == 0) {

        printf("\n[Client] Reading from shared memory...\n");
        printf("[Client] Message: %s\n", shm_buf);

        if (shmdt(addr) == -1) {
            perror("shmdt (client)");
            return 1;
        }
        return 0;
    } else {

        wait(NULL);

        if (shmdt(addr) == -1) {
            perror("shmdt (server)");

        }

        if (shmctl(shmid, IPC_RMID, NULL) == -1) {
            perror("shmctl IPC_RMID");
            return 1;
        }

        printf("[Server] Shared memory detached and removed. Done.\n");
        return 0;
    }
}
```

Listing 11:

11 Assignment 8: Disk Scheduling (SSTF, SCAN, C-LOOK)

File: `Ass8.c`

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

void run_SSTF(void);
void run_SCAN(void);
void run_CLOOK(void);

static void sort(int a[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (a[j] > a[j + 1]) {
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
            }
        }
    }
}

int main(void) {
    while (1) {
        int ch;
        printf("\n=========== DISK SCHEDULING ===========\n");
        printf("1) SSTF\n");
        printf("2) SCAN\n");
        printf("3) C-LOOK\n");
        printf("4) EXIT\n");
        printf("Enter choice: ");
        if (scanf("%d", &ch) != 1) return 0;

        switch (ch) {
            case 1: run_SSTF();  break;
            case 2: run_SCAN();  break;
            case 3: run_CLOOK(); break;
            case 4: return 0;
            default: printf("Invalid choice.\n"); break;
        }
    }
}

void run_SSTF(void) {
    int n, initial;
    int RQ[100];
    int used[100] = {0};
    long long total = 0;

    printf("\nSSTF\nEnter number of requests: ");
    scanf("%d", &n);
    if (n < 1 || n > 100) { printf("Bad n.\n"); return; }

    printf("Enter %d requests: ", n);
    for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);

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

    int head = initial;
    printf("Service order: %d", head);

    for (int served = 0; served < n; ++served) {
        int best = -1;
        int bestDist = 1e9;
        for (int i = 0; i < n; ++i) {
            if (!used[i]) {
                int d = RQ[i] - head;
                if (d < 0) d = -d;
                if (d < bestDist) {
                    bestDist = d;
                    best = i;
                }
            }
        }
        total += bestDist;
        head = RQ[best];
        used[best] = 1;
        printf(" -> %d", head);
    }
    printf("\nTotal head movement: %lld\n", total);
}

void run_SCAN(void) {
    int n, initial, size, move;
    int RQ[100];

    printf("\nSCAN\nEnter number of requests: ");
    scanf("%d", &n);
    if (n < 1 || n > 100) { printf("Bad n.\n"); return; }

    printf("Enter %d requests: ", n);
    for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);

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

    printf("Enter total disk size (number of cylinders): ");
    scanf("%d", &size);

    printf("Head direction (1 = towards higher, 0 = towards lower): ");
    scanf("%d", &move);

    sort(RQ, n);

    int head = initial;
    long long total = 0;
    printf("Service order: %d", head);


    int idx = 0;
    while (idx < n && RQ[idx] < head) idx++;

    if (move == 1) {

        for (int i = idx; i < n; ++i) {
            total += (RQ[i] >= head) ? (RQ[i] - head) : (head - RQ[i]);
            head = RQ[i];
            printf(" -> %d", head);
        }

        if (idx > 0) {
            total += (size - 1 - head);
            head = size - 1;
            printf(" -> %d", head);
            total += (head - RQ[idx - 1]);
            head = RQ[idx - 1];
            printf(" -> %d", head);
            for (int i = idx - 2; i >= 0; --i) {
                total += (head - RQ[i]);
                head = RQ[i];
                printf(" -> %d", head);
            }
        }
    } else {

        for (int i = idx - 1; i >= 0; --i) {
            total += (head >= RQ[i]) ? (head - RQ[i]) : (RQ[i] - head);
            head = RQ[i];
            printf(" -> %d", head);
        }

        if (idx < n) {
            total += head;
            head = 0;
            printf(" -> %d", head);
            total += (RQ[idx] - head);
            head = RQ[idx];
            printf(" -> %d", head);
            for (int i = idx + 1; i < n; ++i) {
                total += (RQ[i] - head);
                head = RQ[i];
                printf(" -> %d", head);
            }
        }
    }

    printf("\nTotal head movement: %lld\n", total);
}

void run_CLOOK(void) {
    int n, initial, size, move;
    int RQ[100];

    printf("\nC-LOOK\nEnter number of requests: ");
    scanf("%d", &n);
    if (n < 1 || n > 100) { printf("Bad n.\n"); return; }

    printf("Enter %d requests: ", n);
    for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);

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

    printf("Enter total disk size (ignored by C-LOOK logic but kept for parity): ");
    scanf("%d", &size);

    printf("Head direction (1 = towards higher, 0 = towards lower): ");
    scanf("%d", &move);

    sort(RQ, n);

    int head = initial;
    long long total = 0;
    printf("Service order: %d", head);

    int idx = 0;
    while (idx < n && RQ[idx] < head) idx++;

    if (move == 1) {

        for (int i = idx; i < n; ++i) {
            total += (RQ[i] - head >= 0) ? (RQ[i] - head) : (head - RQ[i]);
            head = RQ[i];
            printf(" -> %d", head);
        }
        if (idx > 0) {

            total += (RQ[n - 1] >= RQ[0]) ? (RQ[n - 1] - RQ[0]) : (RQ[0] - RQ[n - 1]);
            head = RQ[0];
            printf(" -> %d", head);
            for (int i = 1; i < idx; ++i) {
                total += (RQ[i] - head);
                head = RQ[i];
                printf(" -> %d", head);
            }
        }
    } else {

        for (int i = idx - 1; i >= 0; --i) {
            total += (head - RQ[i] >= 0) ? (head - RQ[i]) : (RQ[i] - head);
            head = RQ[i];
            printf(" -> %d", head);
        }
        if (idx < n) {

            total += (RQ[n - 1] >= RQ[0]) ? (RQ[n - 1] - RQ[0]) : (RQ[0] - RQ[n - 1]);
```

```c
            head = RQ[n - 1];
            printf(" -> %d", head);
            for (int i = n - 2; i >= idx; --i) {
                total += (head - RQ[i]);
                head = RQ[i];
                printf(" -> %d", head);
            }
        }
    }

    printf("\nTotal head movement: %lld\n", total);
}
```

Listing 12: