

# **Operating System Assignments (I3106)**

Code Listing Collection

Ayush Kothadia / Ayush Karanjkhele

November 5, 2025

## **Contents**

<b>1 Assignment 1(A): Address Book Management</b>	<b>2</b>
<b>2 Assignment 2(A): Process Control (Fork, Wait, Zombie, Orphan)</b>	<b>4</b>
<b>3 Assignment 2(B): Process Control (Fork, Execve, Sort/Search)</b>	<b>6</b>
<b>4 Assignment 3: CPU Scheduling (SRTF, Round Robin)</b>	<b>9</b>
<b>5 Assignment 4(A): Producer-Consumer (Counting Semaphores)</b>	<b>14</b>
<b>6 Assignment 4(B): Reader-Writer Problem</b>	<b>16</b>
<b>7 Assignment 5: Deadlock Avoidance (Banker's Algorithm)</b>	<b>18</b>
<b>8 Assignment 6: Page Replacement Algorithms (FIFO, LRU, OPTIMAL)</b>	<b>22</b>
<b>9 Assignment 7(A): Full-Duplex IPC (Named Pipes/FIFO)</b>	<b>25</b>
<b>10 Assignment 7(B): Inter-process Communication (Shared Memory)</b>	<b>27</b>
<b>11 Assignment 8: Disk Scheduling (SSTF, SCAN, C-LOOK)</b>	<b>29</b>

# 1 Assignment 1(A): Address Book Management

File: I3118\_1a.txt

```
1 echo "ENTER ADDRESS BOOK NAME:      "
2 read fname
3 touch $fname
4 echo -e "ADDRESS BOOK CREATED\n"
5 echo -e "NAME\t\t ID \t DOB\t\tADDRESS\t MOB_NO\t\t SALARY">>>$fname
6 ch=0
7 while [ $ch -lt '7' ]
8 do
9 echo -e "ADDRESS BOOK :\n"
10 echo -e "1.CREATE ADDRESS BOOK"
11 echo -e "2.VIEW ADDRESS BOOK"
12 echo -e "3.INSERT A RECORD"
13 echo -e "4.DELETE A RECORD"
14 echo -e "5.MODIFY A RECORD"
15 echo -e "6.SEARCH A RECORD"
16 echo -e "7.EXIT FROM ADDRESS BOOK"
17 echo -e "ENTER YOUR CHOICE"
18 read ch
19 case $ch in
20 1)echo -e "enter number of records you want to enter \t"
21     read n
22     for((i=0;i<$n;i++))
23     do
24         echo -e "ENTER NAME OF EMPLOYEE\t"
25         read ename
26         echo -e "ENTER ID OF EMPLOYEE\t"
27         read eid
28         echo -e "ENTER DOB OF EMPLOYEE\t"
29         read edob
30         echo -e "ENTER ADDRESS OF EMPLOYEE\t"
31         read eadd
32         echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
33         read emob
34         echo -e "ENTER SALARY OF EMPLOYEE\t"
35         read esal
36         echo -e " $ename \t $eid \t $edob \t $eadd \t $emob \t $esal \n">>>$fname
37     done
38     ;;
39 2)
40     cat $fname
41     ;;
42 3)
43     echo -e "enter new record"
44     echo -e "ENTER NAME OF EMPLOYEE\t"
45     read ename
46     echo -e "ENTER ID OF EMPLOYEE\t"
47     read eid
48     echo -e "ENTER DOB OF EMPLOYEE\t"
49     read edob
50     echo -e "ENTER ADDRESS OF EMPLOYEE\t"
51     read eadd
52     echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
53     read emob
54     echo -e "ENTER SALARY OF EMPLOYEE\t"
55     read esal
56     echo -e " $ename \t $eid \t $edob\t\t $eadd \t $emob \t $esal \n">>>$fname
57     ;;
58 4)
59     echo -e "ENTER EMPLOYEE ID TO BE DELETED "
60     read eid
61     if grep -w $eid $fname
62     then
63         grep -wv $eid $fname >>temp
64         rm $fname
65         mv temp $fname
66         echo "RECORD DELETED"
67     else
68     fi
```

```

69          echo "RECORD DOES NOT EXIST "
70      fi
71  ;;
72 5)
73      echo "ENTER EMPLOYEE ID TO BE MODIFIY"
74      read eid
75      if grep -w $eid $fname
76          then
77              grep -wv $eid $fname >>temp
78              rm $fname
79              mv temp $fname
80              echo -e "enter modified record"
81              echo -e "ENTER NAME OF EMPLOYEE\t"
82              read ename
83              echo -e "ENTER ID OF EMPLOYEE\t"
84              read eid
85              echo -e "ENTER DOB OF EMPLOYEE\t"
86              read edob
87              echo -e "ENTER ADDRESS OF EMPLOYEE\t"
88              read eadd
89              echo -e "ENTER MOBILE NO. OF EMPLOYEE\t"
90              read emob
91              echo -e "ENTER SALARY OF EMPLOYEE\t"
92              read esal
93              echo -e " $ename \t$eid \t$edob\t\t$eadd \t$emob \t$esal \n">>$fname
94      else
95          echo "RECORD DOES NOT EXIST "
96      fi
97  ;;
98 6)
99      echo -e "ENTER EMPLOYEE ID TO BE SEARCHED "
100     read eid
101     if grep -w $eid $fname
102         then
103             grep -w $eid $fname
104             echo "RECORD FOUND...!!!"
105         else
106             echo "RECORD DOES NOT EXIST "
107         fi
108  ;;
109     esac
110 done

```

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

File: ass2.a-ayush.txt

```
1 #include<sys/types.h>
2 #include<stdio.h>
3 #include<unistd.h>
4 void bubble_acs(int a[50],int n)
5 {
6     int i,j,temp;
7     for(i=n-1;i>0;i--)
8     {
9         for(j=0;j<i;j++)
10        {
11            if(a[j]>a[j+1])
12            {
13                temp=a[j];
14                a[j]=a[j+1];
15                a[j+1]=temp;
16            }
17        }
18    }
19 }
20 void bubble_dcs(int a[50],int n)
21 {
22     int i,j,temp;
23     for(i=n-1;i>0;i--)
24     {
25         for(j=0;j<i;j++)
26        {
27            if(a[j]<a[j+1])
28            {
29                temp=a[j];
30                a[j]=a[j+1];
31                a[j+1]=temp;
32            }
33        }
34    }
35 }
36
37 int main(){
38     int pid;
39     int i,n,a[50],j;
40     printf("\nEnter the Number of elements:");
41     scanf("%d",&n);
42     printf("\nEnter List of Numbers:\n");
43     for(j=0;j<n;j++)
44     {
45         scanf("%d",&a[j]);
46     }
47     pid = fork();
48     if(pid==0)
49     {
50         printf("\n");
51
52
53         printf("I am child and my id is %d \n",getpid());
54         printf("I am parent and my id is %d\n",getppid());
55     }
56     else
57     {
58         system("ps -el|grep Z");
59         sleep(5);
60         int i= wait(0);
61
62         printf("The terminated child's pid is %d \n",i);
63         printf("I am parent and my id is %d \n",getpid());
64         printf("I am parent's parent and my id is %d",getppid());
65         bubble_dcs(a,n);
66         printf("\nList of Numbers in Descending Order:\n");
67         for(j=0;j<n;j++)
68     }
```

```
69     {
70         printf("%d\n",a[j]);
71     }
72 }
73
74 return 0;
75 }
```

Listing 1:

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

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

```
1 // main_process.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6
7 void sort_descending(int *arr, int size) {
8     int temp;
9     for (int pass = 0; pass < size - 1; pass++) {
10         for (int i = 0; i < size - pass - 1; i++) {
11             if (arr[i] < arr[i + 1]) {
12                 temp = arr[i];
13                 arr[i] = arr[i + 1];
14                 arr[i + 1] = temp;
15             }
16         }
17     }
18
19     printf("\n[Parent] Sorted array in descending order:\n");
20     for (int i = 0; i < size; i++) {
21         printf("%d ", arr[i]);
22     }
23     printf("\n");
24 }
25
26 int main(int argc, char *argv[]) {
27     if (argc < 3) {
28         fprintf(stderr, "Usage: %s <child_program_path> <num1> <num2> ... \n", argv[0]);
29         return 1;
30     }
31
32     int count = argc - 2;
33     int values[count];
34
35
36     for (int i = 0; i < count; i++) {
37         values[i] = atoi(argv[i + 2]);
38     }
39
40     sort_descending(values, count);
41
42
43     char *child_args[count + 2];
44     child_args[0] = argv[1];
45
46     for (int i = 0; i < count; i++) {
47         child_args[i + 1] = malloc(12);
48         sprintf(child_args[i + 1], "%d", values[i]);
49     }
50     child_args[count + 1] = NULL;
51
52
53     pid_t pid = fork();
54
55     if (pid == 0) {
56
57         printf("\n[Child] PID: %d | PPID: %d\n", getpid(), getppid());
58         execve(argv[1], child_args, NULL);
59         perror("execve failed");
60         exit(EXIT_FAILURE);
61     } else if (pid > 0) {
62
63         wait(NULL);
64         printf("\n[Parent] Child finished. PID: %d\n", pid);
65     } else {
66         perror("fork failed");
67         return 1;
68     }
69 }
```

```
69
70 }
71
72 for (int i = 1; i <= count; i++) {
73     free(child_args[i]);
74 }
75
76 return 0;
77 }
```

Listing 2:

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

```
1 // child_process.c
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 void sort_descending(int *arr, int size) {
7     int temp;
8     for (int pass = 0; pass < size - 1; pass++) {
9         for (int i = 0; i < size - pass - 1; i++) {
10             if (arr[i] < arr[i + 1]) {
11                 temp = arr[i];
12                 arr[i] = arr[i + 1];
13                 arr[i + 1] = temp;
14             }
15         }
16     }
17
18     printf("\n[Child] Sorted array in descending order:\n");
19     for (int i = 0; i < size; i++) {
20         printf("%d ", arr[i]);
21     }
22     printf("\n");
23 }
24
25 int main(int argc, char *argv[]) {
26     if (argc < 2) {
27         fprintf(stderr, "Usage: %s <sorted_numbers>\n", argv[0]);
28         return 1;
29     }
30
31     int size = argc - 1;
32     int numbers[size];
33
34     for (int i = 0; i < size; i++) {
35         numbers[i] = atoi(argv[i + 1]);
36     }
37
38     printf("\n[Child] Executing child process with PID: %d\n", getpid());
39
40     sort_descending(numbers, size);
41
42     return 0;
43 }
```

Listing 3:

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

File: Ass3.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX 20
6 #define QSIZE (MAX * 10)
7
8
9 typedef struct {
10     char name[16];
11     int AT;
12     int BT;
13     int rem;
14     int WT;
15     int TAT;
16     int PID;
17 } process;
18
19
20 typedef struct {
21     char name[16];
22     int start;
23     int end;
24 } gantt_event;
25
26 /* prototypes */
27 void get_PCB(process p[], int *n);
28 void disp_table(process p[], int n);
29 float cal_avgwt(process p[], int n);
30 float cal_avgtat(process p[], int n);
31
32 void SJF_Preemptive(process p[], int n);
33 void RoundRobin(process p[], int n, int tq);
34
35 void menu() {
36     printf("\n\t**** CPU SCHEDULING MENU ****\n");
37     printf("1. SJF (Preemptive - SRTF)\n");
38     printf("2. Round Robin\n");
39     printf("3. Exit\n");
40     printf("Enter choice: ");
41 }
42
43 int main() {
44     process P[MAX];
45     int n, ch, tq;
46
47     get_PCB(P, &n);
48
49
50     while (1) {
51         menu();
52         if (scanf("%d", &ch) != 1) {
53
54             break;
55         }
56
57
58         process P_copy[MAX];
59         memcpy(P_copy, P, sizeof(process) * n);
60
61         if (ch == 1) {
62             SJF_Preemptive(P_copy, n);
63             disp_table(P_copy, n);
64             printf("\nAverage WT : %.2f\n", cal_avgwt(P_copy, n));
65             printf("Average TAT: %.2f\n", cal_avgtat(P_copy, n));
66         } else if (ch == 2) {
67             printf("Enter Time Quantum: ");
```

```

69         if (scanf("%d", &tq) != 1) break;
70         if (tq <= 0) {
71             printf("Time quantum must be >= 1\n");
72             continue;
73         }
74         RoundRobin(P_copy, n, tq);
75         disp_table(P_copy, n);
76         printf("\nAverage WT : %.2f\n", cal_avgwt(P_copy, n));
77         printf("Average TAT: %.2f\n", cal_avgtat(P_copy, n));
78     } else {
79         break;
80     }
81 }
82
83 return 0;
84 }

85
86
87 void get_PCB(process p[], int *n) {
88     int i;
89     printf("Enter total no of processes: ");
90     if (scanf("%d", n) != 1) *n = 0;
91
92     for (i = 0; i < *n; i++) {
93         printf("\nProcess %d name (e.g., P%d): ", i + 1, i + 1);
94         scanf("%s", p[i].name);
95         printf("Arrival Time      : ");
96         scanf("%d", &p[i].AT);
97         printf("Burst Time       : ");
98         scanf("%d", &p[i].BT);
99         p[i].rem = p[i].BT;
100        p[i].WT = p[i].TAT = 0;
101        p[i].PID = i;
102    }
103 }
104
105 void disp_table(process p[], int n) {
106     int i;
107     printf("\n\n P_NAME \t AT \t BT \t WT \t TAT\n");
108     for (i = 0; i < n; i++) {
109         printf(" %-8s \t %d \t %d \t %d \t %d\n",
110               p[i].name, p[i].AT, p[i].BT, p[i].WT, p[i].TAT);
111     }
112 }
113
114 float cal_avgwt(process p[], int n) {
115     float s = 0.0f;
116     int i;
117     for (i = 0; i < n; i++) s += p[i].WT;
118     return s / n;
119 }
120
121 float cal_avgtat(process p[], int n) {
122     float s = 0.0f;
123     int i;
124     for (i = 0; i < n; i++) s += p[i].TAT;
125     return s / n;
126 }
127
128
129 void SJF_Preemptive(process p[], int n) {
130     int i, completed = 0, time = 0;
131     int min_idx;
132     int prev_idx = -1;
133     gantt_event events[1000];
134     int evcount = 0;
135
136
137     for (i = 0; i < n; i++) {
138         p[i].rem = p[i].BT;
139         p[i].WT = p[i].TAT = 0;
140     }
141

```

```

142
143     int earliest = p[0].AT;
144     for (i = 1; i < n; i++) if (p[i].AT < earliest) earliest = p[i].AT;
145     time = earliest;
146
147     while (completed < n) {
148
149         min_idx = -1;
150         for (i = 0; i < n; i++) {
151             if (p[i].rem > 0 && p[i].AT <= time) {
152                 if (min_idx == -1 || p[i].rem < p[min_idx].rem ||
153                     (p[i].rem == p[min_idx].rem && p[i].AT < p[min_idx].AT)) {
154                     min_idx = i;
155                 }
156             }
157         }
158
159         if (min_idx == -1) {
160
161             int next_arr = -1;
162             for (i = 0; i < n; i++) {
163
164                 if (p[i].rem > 0) {
165                     if (next_arr == -1 || (p[i].AT > time && p[i].AT < next_arr) || next_arr == -1)
166                         {
167                             next_arr = p[i].AT;
168                         }
169                 }
170             }
171
172             if (next_arr != -1 && next_arr > time) {
173
174                 if (evcount > 0 && strcmp(events[evcount - 1].name, "IDLE") == 0) {
175                     events[evcount - 1].end = next_arr;
176                 } else {
177                     strcpy(events[evcount].name, "IDLE");
178                     events[evcount].start = time;
179                     events[evcount].end = next_arr;
180                     evcount++;
181                 }
182                 time = next_arr;
183             } else {
184
185                 break;
186             }
187             continue;
188         }
189
190         if (prev_idx != min_idx) {
191
192             strcpy(events[evcount].name, p[min_idx].name);
193             events[evcount].start = time;
194             events[evcount].end = time;
195             evcount++;
196         }
197
198
199         p[min_idx].rem--;
200         time++;
201         events[evcount - 1].end = time;
202
203         if (p[min_idx].rem == 0) {
204             completed++;
205
206             p[min_idx].TAT = time - p[min_idx].AT;
207             p[min_idx].WT = p[min_idx].TAT - p[min_idx].BT;
208         }
209         prev_idx = min_idx;
210     }
211
212
213

```

```

214 printf("\n\nGANTT CHART (SJF Preemptive):\n");
215 for (i = 0; i < evcount; i++) {
216     printf(" %s |", events[i].name);
217 }
218 printf("\n%d", events[0].start);
219 for (i = 0; i < evcount; i++) {
220     printf("      %d", events[i].end);
221 }
222 printf("\n");
223 }

225 void RoundRobin(process p[], int n, int tq) {
226     int i;
227
228     for (i = 0; i < n; i++) {
229         p[i].rem = p[i].BT;
230         p[i].WT = p[i].TAT = 0;
231     }
232
233     int time = 0, completed = 0;
234     int queue[QSIZE];
235     int head = 0, tail = 0, countq = 0;
236
237     int in_queue[MAX];
238     memset(in_queue, 0, sizeof(in_queue));
239     gantt_event events[1000];
240     int evcount = 0;
241
242
243     int earliest = p[0].AT;
244     for (i = 1; i < n; i++) if (p[i].AT < earliest) earliest = p[i].AT;
245     time = earliest;
246
247
248     for (i = 0; i < n; i++) {
249         if (p[i].AT <= time && !in_queue[i]) {
250             queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
251         }
252     }
253
254     while (completed < n) {
255         if (countq == 0) {
256
257             int next_arr = -1;
258             for (i = 0; i < n; i++) {
259
260                 if (p[i].rem > 0) {
261
262                     if (p[i].AT > time) {
263                         if (next_arr == -1 || p[i].AT < next_arr) next_arr = p[i].AT;
264                     }
265                 }
266             }
267
268             if (next_arr != -1 && next_arr > time) {
269
270                 if (evcount > 0 && strcmp(events[evcount - 1].name, "IDLE") == 0) {
271                     events[evcount - 1].end = next_arr;
272                 } else {
273                     strcpy(events[evcount].name, "IDLE");
274                     events[evcount].start = time;
275                     events[evcount].end = next_arr;
276                     evcount++;
277                 }
278                 time = next_arr;
279             } else if (next_arr == -1) {
280
281                 break;
282             }
283
284             for (i = 0; i < n; i++) {
285
286

```

```

287     if (!in_queue[i] && p[i].rem > 0 && p[i].AT <= time) {
288         queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
289     }
290     continue;
291 }
292
293
294
295 int idx = queue[head]; head = (head + 1) % QSIZE; countq--; in_queue[idx] = 0;
296
297
298 strcpy(events[evcount].name, p[idx].name);
299 events[evcount].start = time;
300 events[evcount].end = time;
301 evcount++;
302
303
304 int run = (p[idx].rem < tq) ? p[idx].rem : tq;
305
306
307 int t;
308 for (t = 0; t < run; t++) {
309     p[idx].rem--;
310     time++;
311     events[evcount - 1].end = time;
312
313
314     for (i = 0; i < n; i++) {
315
316         if (!in_queue[i] && p[i].rem > 0 && p[i].AT == time) {
317             queue[tail] = i; tail = (tail + 1) % QSIZE; countq++; in_queue[i] = 1;
318         }
319     }
320 }
321
322
323 if (p[idx].rem == 0) {
324
325     completed++;
326     p[idx].TAT = time - p[idx].AT;
327     p[idx].WT = p[idx].TAT - p[idx].BT;
328 } else {
329
330     queue[tail] = idx; tail = (tail + 1) % QSIZE; countq++; in_queue[idx] = 1;
331 }
332
333
334 printf("\n\nGANTT CHART (Round Robin, TQ=%d):\n| ", tq);
335 for (i = 0; i < evcount; i++) {
336     printf(" %s | ", events[i].name);
337 }
338
339 printf("\n%d", events[0].start);
340 for (i = 0; i < evcount; i++) {
341     printf("      %d", events[i].end);
342 }
343
344 printf("\n");
}

```

Listing 4:

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

File: Ass4\_pro.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/syscall.h>
5 #include <pthread.h>
6 #include <semaphore.h>
7
8 #define BUFFER_SIZE 20
9
10 void *producer(void *arg);
11 void *consumer(void *arg);
12
13 typedef struct {
14     int buff[BUFFER_SIZE];
15     sem_t full;
16     sem_t empty;
17 } shared;
18
19 shared sh;
20 int in = 0;
21 int out = 0;
22 sem_t mutex;
23 int next_item = 0;
24
25 int main()
26 {
27     pthread_t ptid1, ptid2, ctid1;
28
29
30
31     sem_init(&sh.empty, 0, BUFFER_SIZE);
32     sem_init(&sh.full, 0, 0);
33
34
35     sem_init(&mutex, 0, 1);
36
37
38     pthread_create(&ptid1, NULL, producer, NULL);
39     pthread_create(&ptid2, NULL, producer, NULL);
40     pthread_create(&ctid1, NULL, consumer, NULL);
41
42
43     pthread_join(ptid1, NULL);
44     pthread_join(ptid2, NULL);
45     pthread_join(ctid1, NULL);
46
47
48     return 0;
49 }
50
51 void *producer(void *arg)
52 {
53     (void)arg;
54
55     while (1)
56     {
57
58         int item = next_item++;
59
60         sem_wait(&sh.empty);
61
62
63         sem_wait(&mutex);
64
65
66         sh.buff[in] = item;
67
68 }
```

```

69     in = (in + 1) % BUFFER_SIZE;
70
71     pid_t tid = (pid_t) syscall(SYS_gettid);
72     printf("producer thread id: %d, produced item: %d\n", (int)tid, item);
73
74     sem_post(&mutex);
75
76     sem_post(&sh.full);
77
78     sleep(2);
79 }
80
81     return NULL;
82 }
83
84 void *consumer(void *arg)
85 {
86     (void)arg;
87
88     while (1)
89     {
90
91         sem_wait(&sh.full);
92
93         sem_wait(&mutex);
94
95         int item = sh.buff[out];
96
97         out = (out + 1) % BUFFER_SIZE;
98
99         pid_t tid = (pid_t) syscall(SYS_gettid);
100        printf("consumer thread id: %d, consumed item: %d\n", (int)tid, item);
101
102        sem_post(&mutex);
103
104        sem_post(&sh.empty);
105
106        sleep(2);
107    }
108
109    return NULL;
110 }

```

Listing 5:

## 6 Assignment 4(B): Reader-Writer Problem

File: Ass4\_read.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<pthread.h>
4 #include<sys/types.h>
5
6 void *reader(void *argp);
7 void *writer(void *argp);
8
9 int buf;
10
11 int getbuff(){
12     int temp;
13     printf("\nEnter the item in buffer :- ");
14     scanf("%d",&temp);
15     return temp;
16 }
17
18 void readbuff(int buf){
19     printf("\nItem READ from buffer : %d\n",buf);
20 }
21
22 pthread_mutex_t mutex1=PTHREAD_MUTEX_INITIALIZER;
23 pthread_mutex_t wrt=PTHREAD_MUTEX_INITIALIZER;
24
25 int read_count=0;
26 int flag=0;
27
28 void *writer(void *argp)
29 {
30     while(1)
31     {
32         pthread_mutex_lock(&wrt);
33         if(flag==0)
34         {
35             buf=getbuff();
36             flag=1;
37         }
38         pthread_mutex_unlock(&wrt);
39     }
40 }
41
42 void *reader(void *argp)
43 {
44     while(1)
45     {
46
47         pthread_mutex_lock(&mutex1);
48         read_count++;
49         if(read_count==1)
50         {
51             pthread_mutex_lock(&wrt);
52         }
53         pthread_mutex_unlock(&mutex1);
54
55         if(flag==1)
56         {
57             readbuff(buf);
58             sleep(2);
59             flag=0;
60         }
61
62
63         pthread_mutex_lock(&mutex1);
64         read_count--;
65         if(read_count==0)
66         {
67             pthread_mutex_unlock(&wrt);
68         }
69     }
70 }
```

```
69     }
70     pthread_mutex_unlock(&mutex1);
71 }
73
74 int main()
75 {
76     pthread_t tid1,tid2,tid3;
77
78     pthread_create(&tid1,NULL,writer,NULL);
79     pthread_create(&tid2,NULL,reader,NULL);
80     pthread_create(&tid3,NULL,reader,NULL);
82
83
84     pthread_join(tid1,NULL);
85     pthread_join(tid2,NULL);
86     pthread_join(tid3,NULL);
87
88     return 0;
89 }
```

Listing 6:

## 7 Assignment 5: Deadlock Avoidance (Banker's Algorithm)

File: Ass5.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct process {
5     int all[6], max[6], need[6], finished, request[6];
6 } p[10];
7
8 int avail[6], sseq[10], ss = 0, check1 = 0, check2 = 0;
9 int n, pid, work[6];
10 int nor, nori;
11
12 int safeseq(void);
13
14 void wait_for_enter(void) {
15     int c;
16     printf("\n\nPress Enter to continue...");
17
18     do { c = getchar(); } while (c != '\n' && c != EOF);
19
20     getchar();
21 }
22
23 int main(void)
24 {
25     int ch, i = 0, j = 0, k, pid, ch1;
26     int violationcheck = 0, waitcheck = 0;
27
28     do {
29
30         printf("\n\n\t 1. Input");
31         printf("\n\n\t 2. New Request");
32         printf("\n\n\t 3. Safe State or Not");
33         printf("\n\n\t 4. Print");
34         printf("\n\n\t 5. Exit");
35         printf("\n\n\t Enter your choice : ");
36
37         if (scanf("%d", &ch) != 1) {
38             printf("\nInvalid input. Exiting.\n");
39             return 0;
40         }
41
42         switch (ch) {
43             case 1:
44                 printf("\n\n\t Enter number of processes : ");
45                 scanf("%d", &n);
46
47                 printf("\n\n\t Enter the Number of Resources : ");
48                 scanf("%d", &nor);
49
50                 printf("\n\n\t Enter the Available Resources : ");
51                 for (k = 0; k < nor; k++) {
52                     printf("\n\n\t For Resource type %d : ", k);
53                     scanf("%d", &avail[k]);
54                 }
55
56
57                 for (i = 0; i < n; i++) {
58                     for (k = 0; k < nor; k++) {
59                         p[i].max[k] = 0;
60                         p[i].all[k] = 0;
61                         p[i].need[k] = 0;
62                         p[i].request[k] = 0;
63                     }
64                     p[i].finished = 0;
65                 }
66
67
68                 for (i = 0; i < n; i++) {
```

```

69         printf("\n\n\t Enter Max and Allocated resources for P%d : ", i);
70         for (j = 0; j < nor; j++) {
71             printf("\n\n\t Enter the Max of resource %d : ", j);
72             scanf("%d", &p[i].max[j]);
73             printf("\n\n\t Allocation of resource %d : ", j);
74             scanf("%d", &p[i].all[j]);
75             if (p[i].all[j] > p[i].max[j]) {
76                 printf("\n\n\t Allocation should be <= Max, please re-enter this
77                     resource.");
78                 j--;
79             } else {
80                 p[i].need[j] = p[i].max[j] - p[i].all[j];
81                 avail[j] -= p[i].all[j];
82             }
83         }
84     break;

85 case 2:
86     violationcheck = 0;
87     waitcheck = 0;
88
89     printf("\n\n\t Requesting process id (0..%d) : ", n-1);
90     scanf("%d", &pid);
91
92     if (pid < 0 || pid >= n) {
93         printf("\n\n\t Invalid PID.");
94         break;
95     }
96
97     for (j = 0; j < nor; j++) {
98         printf("\n\n\t Number of Request for resource %d : ", j);
99         scanf("%d", &p[pid].request[j]);
100        if (p[pid].request[j] > p[pid].need[j])
101            violationcheck = 1;
102        if (p[pid].request[j] > avail[j])
103            waitcheck = 1;
104    }
105
106    if (violationcheck == 1) {
107        printf("\n\n\t The Process Exceeds its Max Need: Terminated");
108    } else if (waitcheck == 1) {
109        printf("\n\n\t Lack of Resources : Process State - Wait");
110    } else {
111
112        for (j = 0; j < nor; j++) {
113            avail[j] -= p[pid].request[j];
114            p[pid].all[j] += p[pid].request[j];
115            p[pid].need[j] -= p[pid].request[j];
116        }
117        ch1 = safeseq();
118        if (ch1 == 0) {
119            printf("\n\n\t Granting leads to Unsafe state: Request Denied");
120
121            for (j = 0; j < nor; j++) {
122                avail[j] += p[pid].request[j];
123                p[pid].all[j] -= p[pid].request[j];
124                p[pid].need[j] += p[pid].request[j];
125            }
126        } else {
127            printf("\n\n\t Request Committed ");
128        }
129    }
130
131 break;

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

138 case 4:

```

```

141     printf("\n\n\t Number of processes : %d", n);
142     printf("\n\n\t Number of Resources : %d", nor);
143
144     printf("\n\n\t Pid \t Max \t Allocated \t Need ");
145     for (i = 0; i < n; i++) {
146         int r;
147         printf("\n\n\t P%d : ", i);
148
149         for (r = 0; r < nor; r++) printf(" %d ", p[i].max[r]);
150         printf("\t");
151         for (r = 0; r < nor; r++) printf(" %d ", p[i].all[r]);
152         printf("\t");
153         for (r = 0; r < nor; r++) printf(" %d ", p[i].need[r]);
154     }
155
156     printf("\n\n\t Available : ");
157     for (i = 0; i < nor; i++) printf(" %d ", avail[i]);
158     break;
159
160     case 5:
161         printf("\n\nExiting...\n");
162         return 0;
163
164     default:
165         printf("\n\n\t Invalid choice.");
166     }
167
168     wait_for_enter();
169 }
170
171 } while (ch != 5);
172
173 return 0;
174 }
175
176 int safeseq(void)
177 {
178     int i, j, t;
179     ss = 0;
180
181     for (j = 0; j < nor; j++)
182         work[j] = avail[j];
183
184     for (j = 0; j < n; j++)
185         p[j].finished = 0;
186
187
188     for (t = 0; t < n; t++) {
189         for (j = 0; j < n; j++) {
190             if (p[j].finished == 0) {
191                 check1 = 0;
192                 for (i = 0; i < nor; i++)
193                     if (p[j].need[i] <= work[i])
194                         check1++;
195
196                 if (check1 == nor) {
197                     for (i = 0; i < nor; i++)
198                         work[i] += p[j].all[i];
199                     p[j].finished = 1;
200                     sseq[ss++] = j;
201                 }
202             }
203         }
204     }
205
206     check2 = 0;
207     for (i = 0; i < n; i++)
208         if (p[i].finished == 1)
209             check2++;
210
211     if (check2 >= n) {
212         printf("\n\n\t The system is in SAFE state\n\t Safe sequence : ");
213         for (i = 0; i < n; i++)

```

```
214     printf("P%d%s", sseq[i], (i < n - 1) ? " -> " : "");  
215     return 1;  
216 } else {  
217     printf("\n\n\t The system is NOT in safe state");  
218     return 0;  
219 }  
220 }
```

Listing 7:

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

File: Ass\_6.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4
5 #define MAX_PAGES 200
6 #define MAX_FRAME 20
7
8 static int in_frames(char frames[], int fsz, char p) {
9     for (int i = 0; i < fsz; i++) if (frames[i] == p) return i;
10    return -1;
11}
12
13 static void print_row(char req, char frames[], int used, int hit) {
14     printf("\n\t%c\t\t", req);
15     for (int i = 0; i < used; i++) {
16         printf("%c\t", frames[i] ? frames[i] : '-');
17     }
18     printf("\t\t%s", hit ? "HIT" : "FAULT");
19 }
20
21 void fifo(char pages[], int n, int frameSize) {
22     char frames[MAX_FRAME] = {0};
23     int used = 0, next = 0, faults = 0;
24
25     printf("\nData Requested\tFrame contents\t\tStatus\n");
26     printf("===== ===== =====\n");
27     for (int i = 0; i < n; i++) {
28         char p = pages[i];
29         int hit = (in_frames(frames, used, p) != -1);
30
31         if (!hit) {
32             faults++;
33             if (used < frameSize) {
34                 frames[used++] = p;
35             } else {
36                 frames[next] = p;
37                 next = (next + 1) % frameSize;
38             }
39         }
40         print_row(p, frames, used, hit);
41     }
42     printf("\n===== \n");
43     printf("\nTotal no. of Page Faults: %d\n\n", faults);
44 }
45
46 void optimal(char pages[], int n, int frameSize) {
47     char frames[MAX_FRAME] = {0};
48     int used = 0, faults = 0;
49
50     printf("\nData Requested\tFrame contents\t\tStatus\n");
51     printf("===== ===== =====\n");
52     for (int i = 0; i < n; i++) {
53         char p = pages[i];
54         int pos = in_frames(frames, used, p);
55         int hit = (pos != -1);
56
57         if (!hit) {
58             faults++;
59             if (used < frameSize) {
60                 frames[used++] = p;
61             } else {
62
63                 int far_idx = 0, far_dist = -1;
64                 for (int f = 0; f < used; f++) {
65                     int dist = 1000000;
66                     for (int j = i + 1; j < n; j++) {
```

```

65         if (pages[j] == frames[f]) {
66             dist = j - i;
67             break;
68         }
69     if (dist > far_dist) {
70         far_dist = dist;
71         far_idx = f;
72     }
73 }
74 frames[far_idx] = p;
75 }
76 print_row(p, frames, used, hit);
77 }
78 printf("\n\n=====\\n");
79 printf("\nTotal no. of Page Faults: %d\\n\\n", faults);
80 }
81
82 void lru(char pages[], int n, int frameSize) {
83     char frames[MAX_FRAME] = {0};
84     int last_used[MAX_FRAME] = {0};
85     int used = 0, faults = 0;
86
87     printf("\\nData Requested\\tFrame contents\\t Status\\n"
88           "=====\\n");
89     for (int time = 0; time < n; time++) {
90         char p = pages[time];
91         int idx = in_frames(frames, used, p);
92         int hit = (idx != -1);
93
94         if (hit) {
95             last_used[idx] = time;
96         } else {
97             faults++;
98             if (used < frameSize) {
99                 frames[used] = p;
100                last_used[used] = time;
101                used++;
102            } else {
103
104                 int lru_idx = 0, lru_time = last_used[0];
105                 for (int f = 1; f < used; f++) {
106                     if (last_used[f] < lru_time) {
107                         lru_time = last_used[f];
108                         lru_idx = f;
109                     }
110                 }
111                 frames[lru_idx] = p;
112                 last_used[lru_idx] = time;
113             }
114         }
115     print_row(p, frames, used, hit);
116 }
117 printf("\n\n=====\\n");
118 printf("\nTotal no. of Page Faults: %d\\n\\n", faults);
119 }
120
121 int main(void) {
122     char line[512];
123     char pages[MAX_PAGES];
124     int n = 0, frameSize, ch;
125
126     printf("Enter the reference string (chars, spaces allowed): ");
127     if (!fgets(line, sizeof(line), stdin)) return 0;
128
129
130     for (int i = 0; line[i] && n < MAX_PAGES; i++) {
131         if (!isspace((unsigned char)line[i])) {
132             pages[n++] = line[i];
133         }
134     }
135 }
136

```

```

137 printf("Enter the size of the frame: ");
138 if (scanf("%d", &frameSize) != 1 || frameSize <= 0 || frameSize > MAX_FRAME) {
139     printf("Invalid frame size.\n");
140     return 0;
141 }
142
143 do {
144     printf("\nMENU\n====\n1. FIFO\n2. Least Recently Used (LRU)\n3. Optimal\n4. Exit\n\nYour
145         Choice: ");
146     if (scanf("%d", &ch) != 1) break;
147     switch (ch) {
148         case 1: fifo(pages, n, frameSize); break;
149         case 2: lru(pages, n, frameSize); break;
150         case 3: optimal(pages, n, frameSize); break;
151         case 4: return 0;
152         default: printf("\nInvalid choice! Please try again!\n");
153     }
154 } while (ch != 4);
155
156 return 0;
}

```

Listing 8:

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

### File: Ass7\_sender.c - Sender Process (Process 1)

```
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/stat.h>
4 #include<fcntl.h>
5 #define Max_Buff 1024
6
7 int main()
8 {
9     int fd1, fd2, c = 0;
10    char *myfifo1 = "myfifo1";
11    char *myfifo2 = "myfifo2";
12    char buff[Max_Buff], ch;
13
14    mkfifo(myfifo1, 0777);
15    mkfifo(myfifo2, 0777);
16
17    printf("\nEnter the string (end with #):\n");
18
19    while((ch = getchar()) != '#')
20        buff[c++] = ch;
21    buff[c] = '\0';
22
23    fd1 = open(myfifo1, O_WRONLY);
24    write(fd1, buff, c + 1);
25    close(fd1);
26
27    fd2 = open(myfifo2, O_RDONLY);
28    read(fd2, buff, Max_Buff);
29    printf("\nContents of file:\n%s\n", buff);
30    close(fd2);
31
32    return 0;
33 }
```

Listing 9:

## File: Ass7\_sender.c - Receiver Process (Process 2)

```
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/stat.h>
4 #include<fcntl.h>
5 #define MAX_BUF 1024
6
7 int main()
8 {
9     int words = 0, lines = 0, chars = 0, i = 0;
10    char buf[MAX_BUF];
11    int fd, fd1;
12    FILE *fp;
13
14    char *myfifo1 = "myfifo1";
15    char *myfifo2 = "myfifo2";
16
17    mkfifo(myfifo1, 0777);
18    mkfifo(myfifo2, 0777);
19
20    fd = open(myfifo1, O_RDONLY);
21    read(fd, buf, MAX_BUF);
22    close(fd);
23
24    printf("\nMessage received:\n%s\n", buf);
25
26    while (buf[i] != '\0')
27    {
28        if (buf[i] == ' ' || buf[i] == '\n')
29            words++;
30        else
31            chars++;
32
33        if (buf[i] == '\n')
34            lines++;
35
36        i++;
37    }
38
39    words++;
40    lines++;
41
42    fp = fopen("a.txt", "w");
43    fprintf(fp, "\nNo. of lines are : %d\n", lines);
44    fprintf(fp, "\nNo. of words are : %d\n", words);
45    fprintf(fp, "\nNo. of chars are : %d\n", chars);
46    fclose(fp);
47
48    fd1 = open(myfifo2, O_WRONLY);
49    dprintf(fd1, "\nNo. of lines are : %d\nNo. of words are : %d\nNo. of chars are : %d\n",
50            lines, words, chars);
51    close(fd1);
52
53    return 0;
54 }
```

Listing 10:

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

File: Ass\_7B\_shared\_ayush.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/shm.h>
8 #include <sys/wait.h>
9
10 #define SHM_SIZE 4096
11
12 int main(void) {
13
14     int shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
15     if (shmid < 0) {
16         perror("shmget");
17         return 1;
18     }
19     printf("Shared memory created. ID = %d\n", shmid);
20
21
22     void *addr = shmat(shmid, NULL, 0);
23     if (addr == (void *)-1) {
24         perror("shmat");
25
26         shmctl(shmid, IPC_RMID, NULL);
27         return 1;
28     }
29     char *shm_buf = (char *)addr;
30
31
32     char input[SHM_SIZE];
33     printf("[Server] Enter the message to write: ");
34     fflush(stdout);
35
36     if (!fgets(input, sizeof(input), stdin)) {
37         fprintf(stderr, "Failed to read input.\n");
38         shmdt(addr);
39         shmctl(shmid, IPC_RMID, NULL);
40         return 1;
41     }
42
43     input[strcspn(input, "\n")] = '\0';
44
45
46     strncpy(shm_buf, input, SHM_SIZE - 1);
47     shm_buf[SHM_SIZE - 1] = '\0';
48     printf("[Server] Message written to shared memory.\n");
49
50
51     pid_t pid = fork();
52     if (pid < 0) {
53         perror("fork");
54         shmdt(addr);
55         shmctl(shmid, IPC_RMID, NULL);
56         return 1;
57     }
58
59     if (pid == 0) {
60
61         printf("\n[Client] Reading from shared memory...\n");
62         printf("[Client] Message: %s\n", shm_buf);
63
64         if (shmdt(addr) == -1) {
65             perror("shmdt (client)");
66             return 1;
67     }
```

```
67     }
68     return 0;
69 } else {
70
71     wait(NULL);
72
73     if (shmrdt(addr) == -1) {
74         perror("shmrdt (server)");
75     }
76
77     if (shmctl(shmid, IPC_RMID, NULL) == -1) {
78         perror("shmctl IPC_RMID");
79         return 1;
80     }
81
82     printf("[Server] Shared memory detached and removed. Done.\n");
83     return 0;
84 }
85
86 }
```

Listing 11:

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

File: Ass8.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void run_SSTF(void);
5 void run_SCAN(void);
6 void run_CLOOK(void);
7
8 static void sort(int a[], int n) {
9     for (int i = 0; i < n - 1; ++i) {
10         for (int j = 0; j < n - i - 1; ++j) {
11             if (a[j] > a[j + 1]) {
12                 int t = a[j];
13                 a[j] = a[j + 1];
14                 a[j + 1] = t;
15             }
16         }
17     }
18 }
19
20 int main(void) {
21     while (1) {
22         int ch;
23         printf("\n===== DISK SCHEDULING =====\n");
24         printf("1) SSTF\n");
25         printf("2) SCAN\n");
26         printf("3) C-LOOK\n");
27         printf("4) EXIT\n");
28         printf("Enter choice: ");
29         if (scanf("%d", &ch) != 1) return 0;
30
31         switch (ch) {
32             case 1: run_SSTF(); break;
33             case 2: run_SCAN(); break;
34             case 3: run_CLOOK(); break;
35             case 4: return 0;
36             default: printf("Invalid choice.\n"); break;
37         }
38     }
39 }
40
41 void run_SSTF(void) {
42     int n, initial;
43     int RQ[100];
44     int used[100] = {0};
45     long long total = 0;
46
47     printf("\nSSTF\nEnter number of requests: ");
48     scanf("%d", &n);
49     if (n < 1 || n > 100) { printf("Bad n.\n"); return; }
50
51     printf("Enter %d requests: ", n);
52     for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);
53
54     printf("Enter initial head position: ");
55     scanf("%d", &initial);
56
57     int head = initial;
58     printf("Service order: %d", head);
59
60     for (int served = 0; served < n; ++served) {
61         int best = -1;
62         int bestDist = 1e9;
63         for (int i = 0; i < n; ++i) {
64             if (!used[i]) {
65                 int d = RQ[i] - head;
66                 if (d < 0) d = -d;
67                 if (d < bestDist) {
68                     bestDist = d;
```

```

69             best = i;
70         }
71     }
72 }
73 total += bestDist;
74 head = RQ[best];
75 used[best] = 1;
76 printf(" -> %d", head);
77 }
78 printf("\nTotal head movement: %lld\n", total);
79 }

80 void run_SCAN(void) {
81     int n, initial, size, move;
82     int RQ[100];
83
84     printf("\nSCAN\nEnter number of requests: ");
85     scanf("%d", &n);
86     if (n < 1 || n > 100) { printf("Bad n.\n"); return; }
87
88     printf("Enter %d requests: ", n);
89     for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);
90
91     printf("Enter initial head position: ");
92     scanf("%d", &initial);
93
94     printf("Enter total disk size (number of cylinders): ");
95     scanf("%d", &size);
96
97     printf("Head direction (1 = towards higher, 0 = towards lower): ");
98     scanf("%d", &move);
99
100    sort(RQ, n);
101
102    int head = initial;
103    long long total = 0;
104    printf("Service order: %d", head);
105
106
107    int idx = 0;
108    while (idx < n && RQ[idx] < head) idx++;
109
110    if (move == 1) {
111
112        for (int i = idx; i < n; ++i) {
113            total += (RQ[i] >= head) ? (RQ[i] - head) : (head - RQ[i]);
114            head = RQ[i];
115            printf(" -> %d", head);
116        }
117
118        if (idx > 0) {
119            total += (size - 1 - head);
120            head = size - 1;
121            printf(" -> %d", head);
122            total += (head - RQ[idx - 1]);
123            head = RQ[idx - 1];
124            printf(" -> %d", head);
125            for (int i = idx - 2; i >= 0; --i) {
126                total += (head - RQ[i]);
127                head = RQ[i];
128                printf(" -> %d", head);
129            }
130        }
131    } else {
132
133        for (int i = idx - 1; i >= 0; --i) {
134            total += (head >= RQ[i]) ? (head - RQ[i]) : (RQ[i] - head);
135            head = RQ[i];
136            printf(" -> %d", head);
137        }
138
139        if (idx < n) {
140            total += head;
141        }

```

```

142     head = 0;
143     printf(" -> %d", head);
144     total += (RQ[idx] - head);
145     head = RQ[idx];
146     printf(" -> %d", head);
147     for (int i = idx + 1; i < n; ++i) {
148         total += (RQ[i] - head);
149         head = RQ[i];
150         printf(" -> %d", head);
151     }
152 }
153 }
154
155 printf("\nTotal head movement: %lld\n", total);
156 }

157 void run_CLOOK(void) {
158     int n, initial, size, move;
159     int RQ[100];
160
161     printf("\nC-LLOOK\nEnter number of requests: ");
162     scanf("%d", &n);
163     if (n < 1 || n > 100) { printf("Bad n.\n"); return; }
164
165     printf("Enter %d requests: ", n);
166     for (int i = 0; i < n; ++i) scanf("%d", &RQ[i]);
167
168     printf("Enter initial head position: ");
169     scanf("%d", &initial);
170
171     printf("Enter total disk size (ignored by C-LLOOK logic but kept for parity): ");
172     scanf("%d", &size);
173
174     printf("Head direction (1 = towards higher, 0 = towards lower): ");
175     scanf("%d", &move);
176
177     sort(RQ, n);
178
179     int head = initial;
180     long long total = 0;
181     printf("Service order: %d", head);
182
183     int idx = 0;
184     while (idx < n && RQ[idx] < head) idx++;
185
186     if (move == 1) {
187
188         for (int i = idx; i < n; ++i) {
189             total += (RQ[i] - head >= 0) ? (RQ[i] - head) : (head - RQ[i]);
190             head = RQ[i];
191             printf(" -> %d", head);
192         }
193         if (idx > 0) {
194
195             total += (RQ[n - 1] >= RQ[0]) ? (RQ[n - 1] - RQ[0]) : (RQ[0] - RQ[n - 1]);
196             head = RQ[0];
197             printf(" -> %d", head);
198             for (int i = 1; i < idx; ++i) {
199                 total += (RQ[i] - head);
200                 head = RQ[i];
201                 printf(" -> %d", head);
202             }
203         }
204     } else {
205
206         for (int i = idx - 1; i >= 0; --i) {
207             total += (head - RQ[i] >= 0) ? (head - RQ[i]) : (RQ[i] - head);
208             head = RQ[i];
209             printf(" -> %d", head);
210         }
211         if (idx < n) {
212
213             total += (RQ[n - 1] >= RQ[0]) ? (RQ[n - 1] - RQ[0]) : (RQ[0] - RQ[n - 1]);

```

```
215     head = RQ[n - 1];
216     printf(" -> %d", head);
217     for (int i = n - 2; i >= idx; --i) {
218         total += (head - RQ[i]);
219         head = RQ[i];
220         printf(" -> %d", head);
221     }
222 }
223
224 printf("\nTotal head movement: %lld\n", total);
225 }
```

Listing 12: