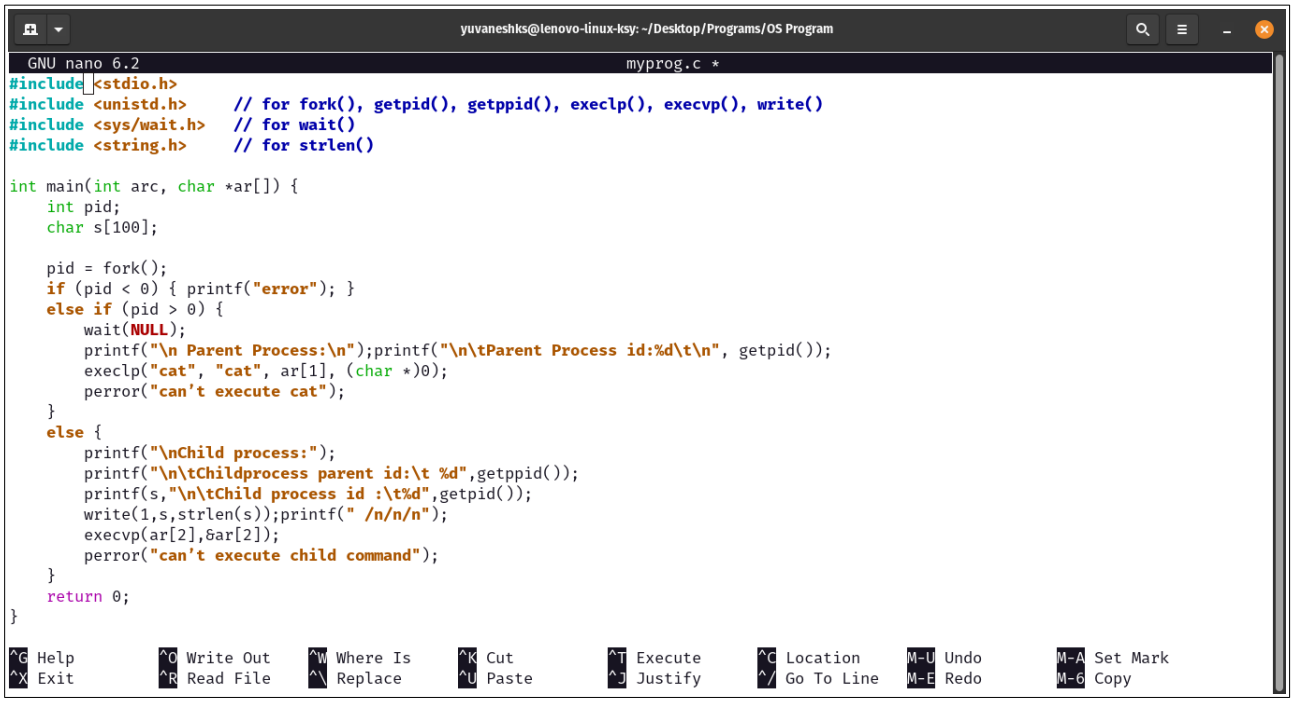
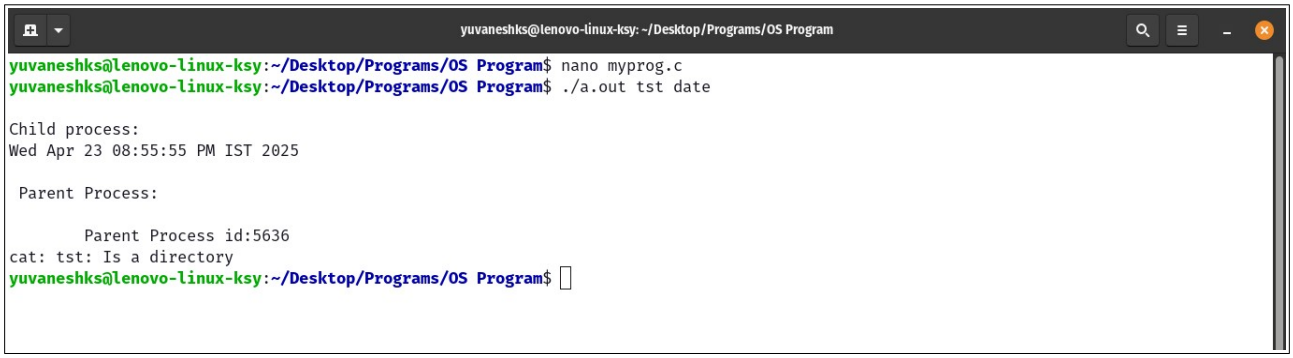
OS Lab Experiment

YuvaneshKS

EX: 3 IMPLEMENTATION OF FORK, EXEC, GETPID, EXIT, WAIT, AND CLOSE SYSTEM CALLS.





EX : 4(a) FCFS SCHEDULING ALGORITHM

```
#include <stdio.h>

int main() {
   int bt[50], wt[50], at[50], wat[50], ft[50], tat[50];
   int i, n;
   float awt = 0, att = 0, sum = 0, sum1 = 0;
   char p[10][5];
```

```
    printf("\nEnter the number of processes: ");
    scanf("%d", &n);

    printf("\nEnter the process name and burst time:\n");
    for (i = 0; i < n; i++) { scanf("%s %d", p[i], &bt[i]); }

    printf("\nEnter the arrival times:\n");
    for (i = 0; i < n; i++) { scanf("%d", &at[i]); }

    wt[0] = 0;
    for (i = 1; i < n; i++) { wt[i] = wt[i - 1] + bt[i - 1]; }

    ft[0] = bt[0];
    for (i = 1; i < n; i++) { ft[i] = ft[i - 1] + bt[i]; }

    printf("\n\n\t\t\tGANTT CHART\n");
    printf("-------------------------------------------------\n");
    for (i = 0; i < n; i++) { printf("|\t%s\t", p[i]); }
    printf("|\n-------------------------------------------------\n");

    for (i = 0; i < n; i++) { printf("%d\t\t", wt[i]); }
    printf("%d\n", wt[n - 1] + bt[n - 1]);

    for (i = 0; i < n; i++) { wat[i] = wt[i] - at[i]; tat[i] = ft[i] - at[i]; }

    printf("\nFIRST COME FIRST SERVE\n");
    printf("\nProcess \tBurst-Time \tArrival-Time \tWaiting-Time \tFinish-Time \t
Turnaround-Time \n");
    for (i = 0; i < n; i++) { printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], at[i],
wat[i], ft[i], tat[i]); }

    for (i = 0; i < n; i++) { sum += wat[i]; sum1 += tat[i]; }

    awt = sum / n;
    att = sum1 / n;

    printf("\nAverage Waiting Time: %.2f", awt);
    printf("\nAverage Turnaround Time: %.2f\n", att);
    return 0;
}
```

```
yuvaneshks@lenovo-linux-ksy: ~/Desktop/Programs/OS Program

yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ ./fcfs

Enter the number of processes: 3

Enter the process name and burst time:
p1 2
p2 3
p3 4

Enter the arrival times:
0 1 2


                        GANTT CHART
-------------------------------------------------
|       p1      |       p2      |       p3      |
-------------------------------------------------
0               2               5               9

FIRST COME FIRST SERVE

Process Burst-Time      Arrival-Time    Waiting-Time    Finish-Tim
e       Turnaround-Time
p1              2               0               0               22
p2              3               1               1               54
p3              4               2               3               97

Average Waiting Time: 1.33
Average Turnaround Time: 4.33
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 4(b) SJF SCHEDULING ALGORITHM

```
#include<stdio.h>
void main() {
   int i, j, n, bt[30], at[30], st[30], ft[30], wat[30], wt[30], temp, temp1, tot, tt[30];
   float awt, att;
   int p[15];
   wat[1] = 0;
```

```
    printf("ENTER THE NO.OF PROCESS: ");
    scanf("%d", &n);
    printf("\nENTER THE PROCESS NUMBER, BURST TIME AND ARRIVAL TIME\n");
    for(i = 1; i <= n; i++) { scanf("%d\t %d\t %d", &p[i], &bt[i], &at[i]); }

    printf("\nPROCESS\tBURST TIME\tARRIVAL TIME\n");
    for(i = 1; i <= n; i++) { printf("\np%d\t%d\t\t%d", p[i], bt[i], at[i]); }

    for(i = 1; i <= n; i++) {
       for(j = i + 1; j <= n; j++) {
          if(bt[i] > bt[j]) {
             temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
             temp1 = p[i]; p[i] = p[j]; p[j] = temp1;
          }
       }
       if(i != 1) {
          st[1] = 0; ft[1] = bt[1]; wt[1] = 0;
          st[i] = ft[i-1]; ft[i] = st[i] + bt[i]; wt[i] = st[i];
       }
    }

    printf("\n\n\t\t\tGANTT CHART\n");
    for(i = 1; i <= n; i++) { printf("|\tp%d\t", p[i]); }
    printf("|\t\n");
    for(i = 1; i <= n; i++) { printf("%d\t\t", wt[i]); }
    printf("%d\n", wt[n] + bt[n]);

    for(i = 2; i <= n; i++) { wat[i] = wt[i] - at[i]; }
    for(i = 1; i <= n; i++) { tt[i] = wat[i] + bt[i] - at[i]; }

    printf("\nPROCESS\tBURST TIME\tARRIVAL TIME\tWAITING TIME\
tTURNAROUND TIME\n");
    for(i = 1; i <= n; i++) { printf("\np%d %5d %15d %15d %15d", p[i], bt[i], at[i], wat[i],
tt[i]); }

    for(i = 1, tot = 0; i <= n; i++) { tot += wt[i]; }
    awt = (float)tot / n;
    printf("\n\n\nAVERAGE WAITING TIME = %f", awt);
```

```
    for(i = 1, tot = 0; i <= n; i++) { tot += tt[i]; }
    att = (float)tot / n;
    printf("\n \n AVERAGE TURNAROUND TIME = %f", att);
}
```

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ ./sjf
ENTER THE NO.OF PROCESS: 3

ENTER THE PROCESS NUMBER, BURST TIME AND ARRIVAL TIME
1 8 1
2 5 1
3 3 1

PROCESS BURST TIME      ARRIVAL TIME

p1      8               1
p2      5               1
p3      3               1

                        GANTT CHART
|       p3      |       p2      |       p1      |
0               3               8               16

PROCESS BURST TIME      ARRIVAL TIME    WAITING TIME    TURNAROUND
 TIME

p3      3               1               0               2
p2      5               1               2               6
p1      8               1               7               14


AVERAGE WAITING TIME = 3.666667

AVERAGE TURNAROUND TIME = 7.333333
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 5 Shared Memory & Inter Process communication

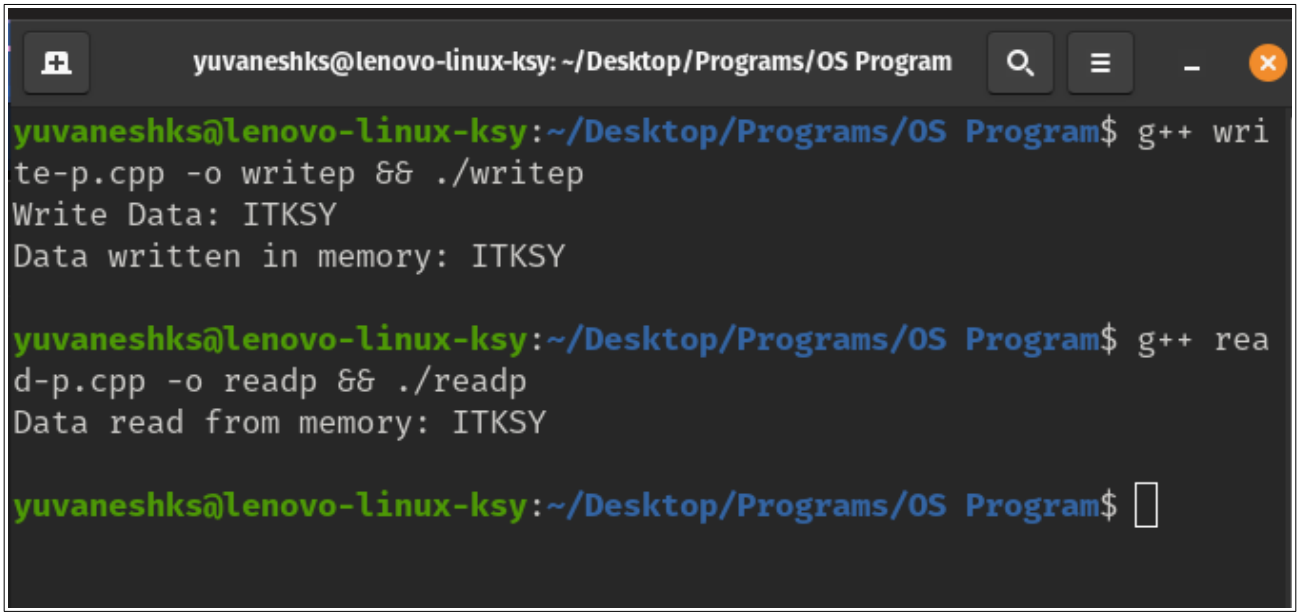
Writer Process (Writing to Shared Memory)

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

using namespace std;

int main() {
// ftok to generate unique key
key_t key = ftok("shmfile", 65);
// shmget returns an identifier in shmid
int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
// shmat to attach to shared memory
char *str = (char*) shmat(shmid, (void*)0, 0);
printf("Write Data: ");
fgets(str, 1024, stdin); // Safer alternative to gets()
printf("Data written in memory: %s\n", str);
// Detach from shared memory
shmdt(str);
return 0;
}
```

Reader Process (Reading from Shared Memory)

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

using namespace std;

int main() {
// ftok to generate unique key
key_t key = ftok("shmfile", 65);
// shmget returns an identifier in shmid
int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
// shmat to attach to shared memory
char *str = (char*) shmat(shmid, (void*)0, 0);
```

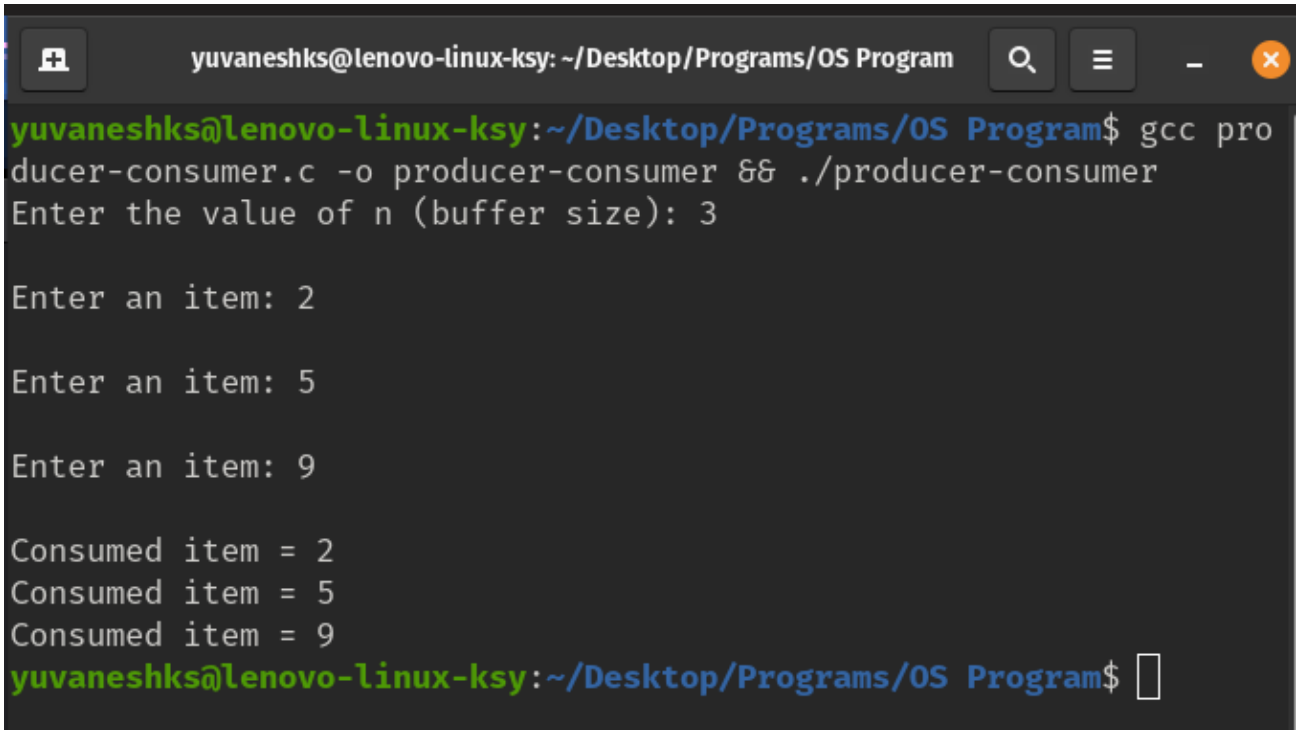
```
// Read and display the data from shared memory
printf("Data read from memory: %s\n", str);
// Detach from shared memory
shmdt(str);
// Destroy the shared memory
shmctl(shmid, IPC_RMID, NULL);
return 0;
}
```



EX : 6 Producer & consumer Problem(Semaphore)

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

#define BUFFERSIZE 10
int mutex = 1, n, empty, full = 0, item, item1;
int buffer[BUFFERSIZE]; int in = 0, out = 0;

void wait(int *s) { while (*s <= 0); (*s)--; }
void signal(int *s) { (*s)++; }

void producer() {
   do { wait(&empty); wait(&mutex); printf("\nEnter an item: "); scanf("%d", &item);
buffer[in] = item; in = (in + 1) % BUFFERSIZE; signal(&mutex); signal(&full); } while
(in != n);
}
```

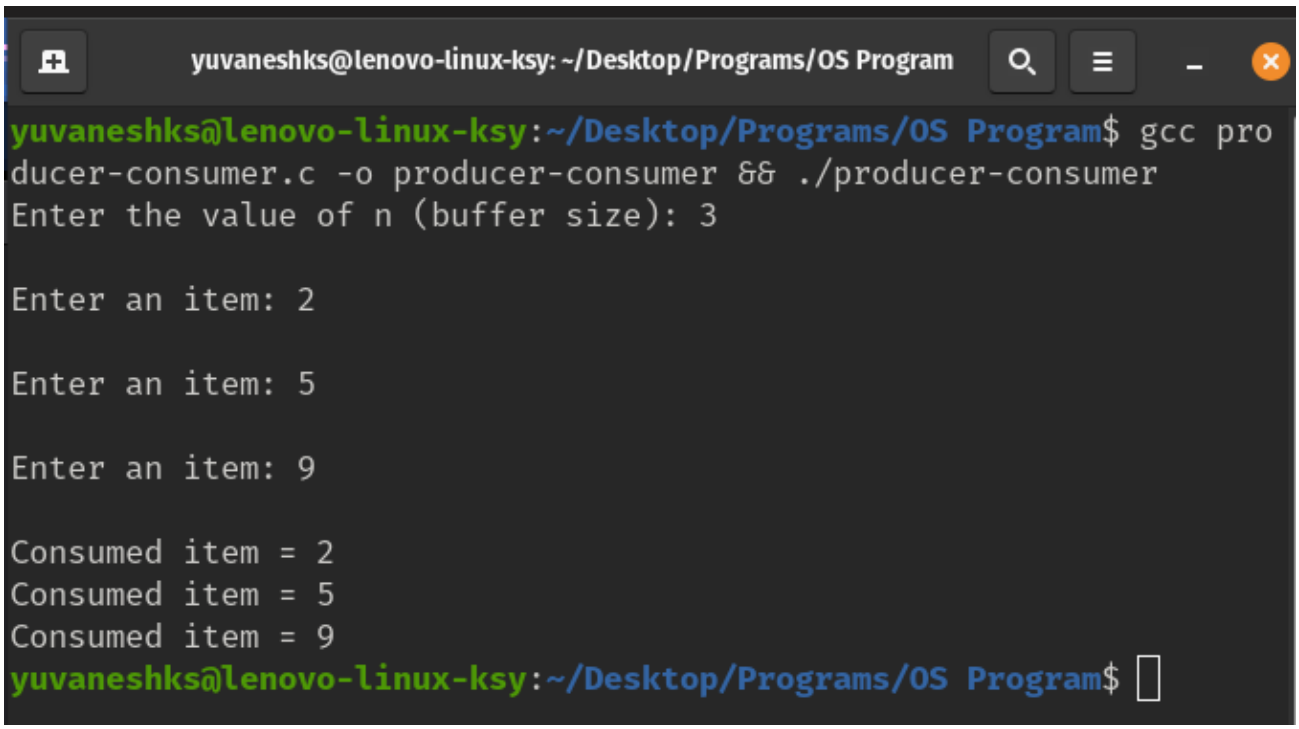
```
void consumer() {
    do { wait(&full); wait(&mutex); item1 = buffer[out]; printf("\nConsumed item = %d",
item1); out = (out + 1) % BUFFERSIZE; signal(&mutex); signal(&empty); } while (out !
= n);
}

int main() {
    printf("Enter the value of n (buffer size): "); scanf("%d", &n);
    empty = n;
    while (in < n) producer();
    while (in != out) consumer();
    printf("\n");
    return 0;
}
```



EX : 7 Deadlock avoidance

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

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main() {
   int pno, rno, i, j, prc, count, total;
   int flag[MAX_PROCESSES], tres[MAX_RESOURCES], max[MAX_PROCESSES]
[MAX_RESOURCES];
   int allocated[MAX_PROCESSES][MAX_RESOURCES], avail[MAX_RESOURCES],
work[MAX_RESOURCES], need[MAX_PROCESSES][MAX_RESOURCES];

   count = 0;

   printf("\nEnter number of processes: "); scanf("%d", &pno);
   printf("\nEnter number of resources: "); scanf("%d", &rno);

   for (i = 0; i < pno; i++) { flag[i] = 0; }

   printf("\nEnter total numbers of each resource: ");
   for (i = 0; i < rno; i++) { scanf("%d", &tres[i]); }

   printf("\nEnter Max resources for each process: ");
   for (i = 0; i < pno; i++) {
      printf("\nFor process %d:\n", i + 1);
      for (j = 0; j < rno; j++) { scanf("%d", &max[i][j]); }
   }

   printf("\nEnter allocated resources for each process: ");
   for (i = 0; i < pno; i++) {
      printf("\nFor process %d:\n", i + 1);
      for (j = 0; j < rno; j++) { scanf("%d", &allocated[i][j]); }
   }

   printf("\nAvailable resources:\n");
   for (j = 0; j < rno; j++) {
      avail[j] = tres[j]; total = 0;
      for (i = 0; i < pno; i++) { total += allocated[i][j]; }
      avail[j] -= total; work[j] = avail[j];
```

```
        printf("%d\t", work[j]);
    }

    do {
        prc = -1;
        for (i = 0; i < pno; i++) { for (j = 0; j < rno; j++) { need[i][j] = max[i][j] - allocated[i]
[j]; } }

        printf("\nAllocated Matrix\tMax Matrix\tNeed Matrix\n");
        for (i = 0; i < pno; i++) {
            for (j = 0; j < rno; j++) { printf("%4d", allocated[i][j]); }
            printf(" | ");
            for (j = 0; j < rno; j++) { printf("%4d", max[i][j]); }
            printf(" | ");
            for (j = 0; j < rno; j++) { printf("%4d", need[i][j]); }
            printf("\n");
        }

        for (i = 0; i < pno; i++) {
            if (flag[i] == 0) { prc = i;
                for (j = 0; j < rno; j++) { if (work[j] < need[i][j]) { prc = -1; break; } }
            }
            if (prc != -1) break;
        }

        if (prc != -1) {
            printf("\nProcess %d completed", prc + 1); count++;
            printf("\nAvailable resources: ");
            for (j = 0; j < rno; j++) {
                work[j] += allocated[prc][j]; allocated[prc][j] = 0;
                max[prc][j] = 0; flag[prc] = 1;
                printf("%d ", work[j]);
            }
        }
    } while (count != pno && prc != -1);

    if (count == pno) printf("\nThe system is in a safe state!!");
    else printf("\nThe system is in an unsafe state!!");

    return 0;
}
```

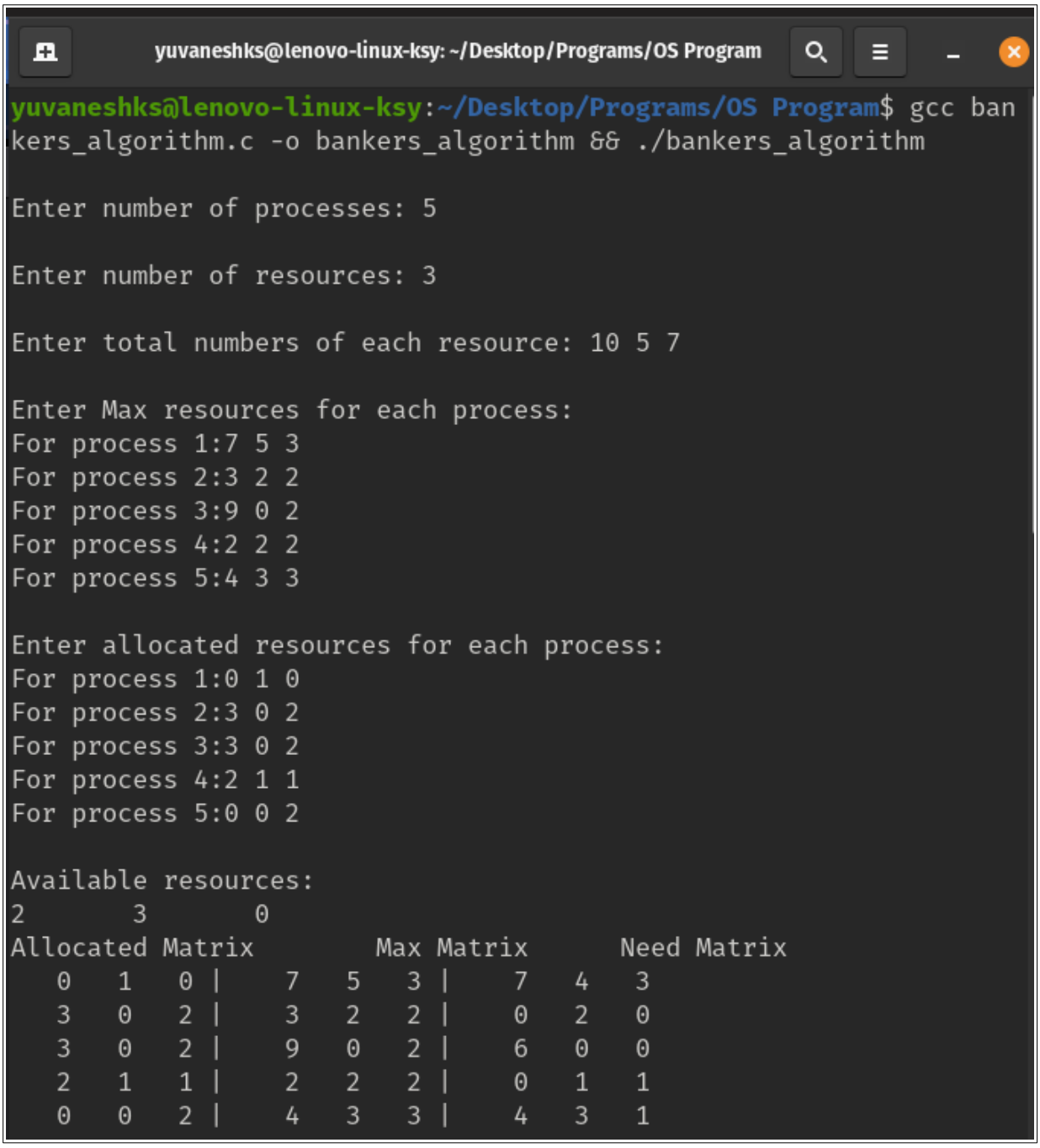


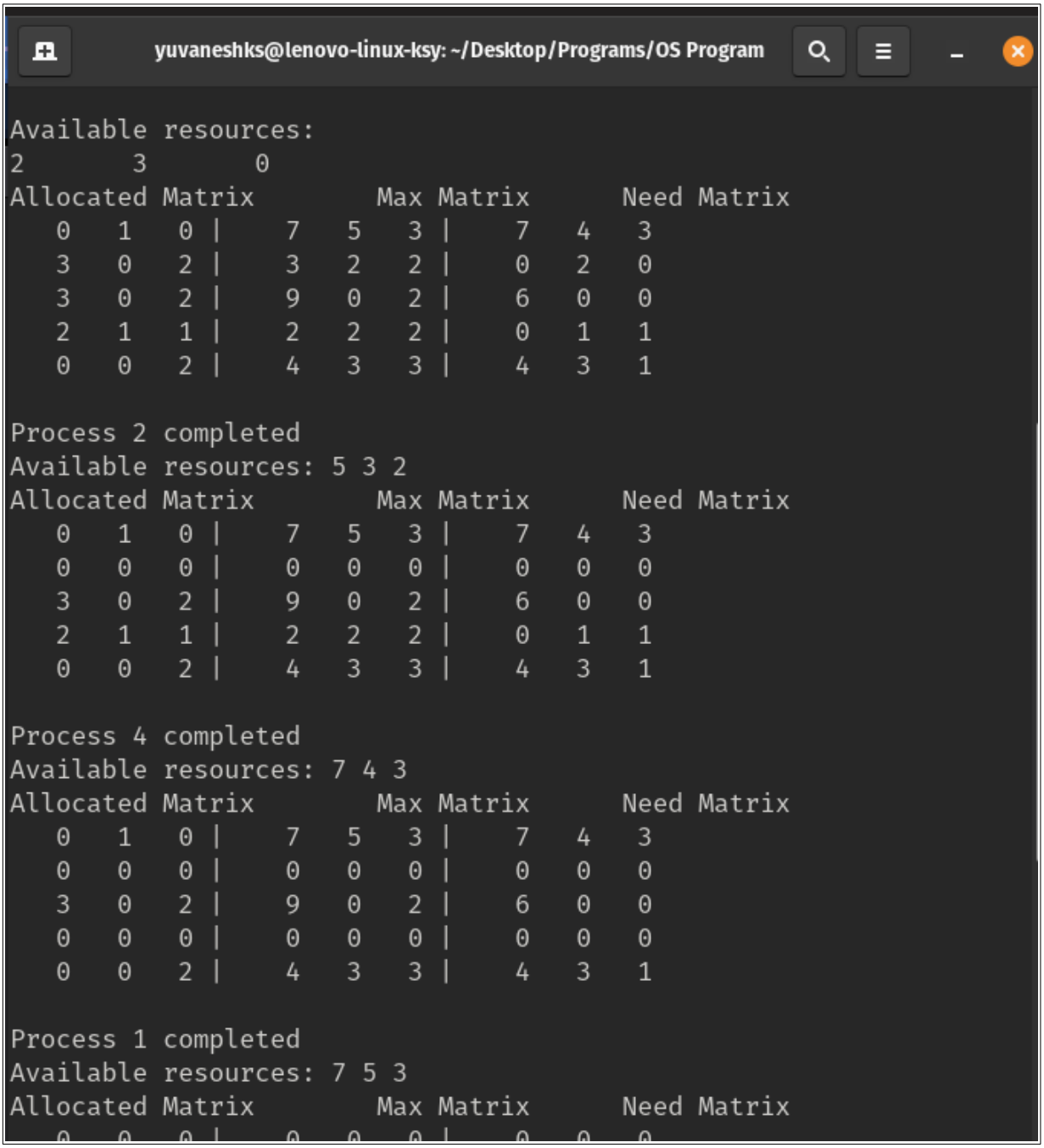
EX : 7 DeadLock Avoidance

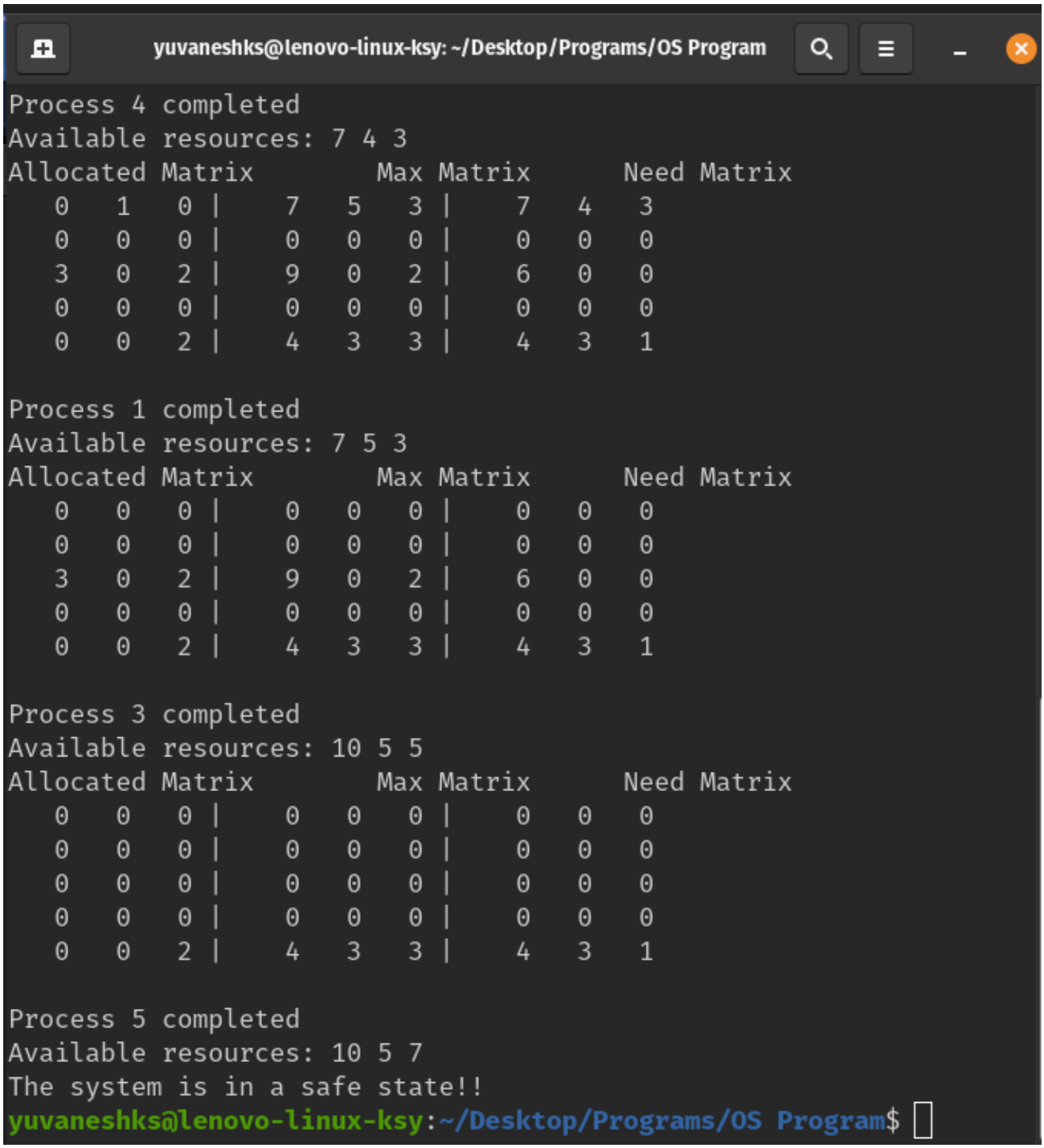
```
#include <stdio.h>
#include <stdlib.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int main() {
   int pno, rno, i, j, prc, count, total;
   int flag[MAX_PROCESSES], tres[MAX_RESOURCES], max[MAX_PROCESSES]
[MAX_RESOURCES];
   int allocated[MAX_PROCESSES][MAX_RESOURCES], avail[MAX_RESOURCES],
work[MAX_RESOURCES], need[MAX_PROCESSES][MAX_RESOURCES];

   count = 0;

   printf("\nEnter number of processes: "); scanf("%d", &pno);
   printf("\nEnter number of resources: "); scanf("%d", &rno);

   for (i = 0; i < pno; i++) { flag[i] = 0; }
```

```
    printf("\nEnter total numbers of each resource: ");
    for (i = 0; i < rno; i++) { scanf("%d", &tres[i]); }

    printf("\nEnter Max resources for each process: \n");
    for (i = 0; i < pno; i++) {
        printf("For process %d:", i + 1);
        for (j = 0; j < rno; j++) { scanf("%d", &max[i][j]); }
    }

    printf("\nEnter allocated resources for each process: \n");
    for (i = 0; i < pno; i++) {
        printf("For process %d:", i + 1);
        for (j = 0; j < rno; j++) { scanf("%d", &allocated[i][j]); }
    }

    printf("\nAvailable resources:\n");
    for (j = 0; j < rno; j++) {
        avail[j] = tres[j]; total = 0;
        for (i = 0; i < pno; i++) { total += allocated[i][j]; }
        avail[j] -= total; work[j] = avail[j];
        printf("%d\t", work[j]);
    }

    do {
        prc = -1;
        for (i = 0; i < pno; i++) { for (j = 0; j < rno; j++) { need[i][j] = max[i][j] - allocated[i]
[j]; } }

        printf("\nAllocated Matrix\tMax Matrix\tNeed Matrix\n");
        for (i = 0; i < pno; i++) {
            for (j = 0; j < rno; j++) { printf("%4d", allocated[i][j]); }
            printf(" | ");
            for (j = 0; j < rno; j++) { printf("%4d", max[i][j]); }
            printf(" | ");
            for (j = 0; j < rno; j++) { printf("%4d", need[i][j]); }
            printf("\n");
        }

        for (i = 0; i < pno; i++) {
            if (flag[i] == 0) { prc = i;
                for (j = 0; j < rno; j++) { if (work[j] < need[i][j]) { prc = -1; break; } }
            }
            if (prc != -1) break;
```

```
        }

        if (prc != -1) {
            printf("\nProcess %d completed", prc + 1); count++;
            printf("\nAvailable resources: ");
            for (j = 0; j < rno; j++) {
                work[j] += allocated[prc][j]; allocated[prc][j] = 0;
                max[prc][j] = 0; flag[prc] = 1;
                printf("%d ", work[j]);
            }
        }
    } while (count != pno && prc != -1);

    if (count == pno) printf("\nThe system is in a safe state!!");
    else printf("\nThe system is in an unsafe state!!");
    printf("\n");

    return 0;
}
```


yuvaneshks@lenovo-linux-ksy: ~/Desktop/Programs/OS Program
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc bankers_algorithm.c -o bankers_algorithm && ./bankers_algorithm
Enter number of processes: 5
Enter number of resources: 3
Enter total numbers of each resource: 10 5 7
Enter Max resources for each process:
For process 1:7 5 3
For process 2:3 2 2
For process 3:9 0 2
For process 4:2 2 2
For process 5:4 3 3
Enter allocated resources for each process:
For process 1:0 1 0
For process 2:3 0 2
For process 3:3 0 2
For process 4:2 1 1
For process 5:0 0 2
Available resources:
2 3 0
Allocated Matrix Max Matrix Need Matrix
0 1 0 | 7 5 3 | 7 4 3
3 0 2 | 3 2 2 | 0 2 0
3 0 2 | 9 0 2 | 6 0 0
2 1 1 | 2 2 2 | 0 1 1
0 0 2 | 4 3 3 | 4 3 1

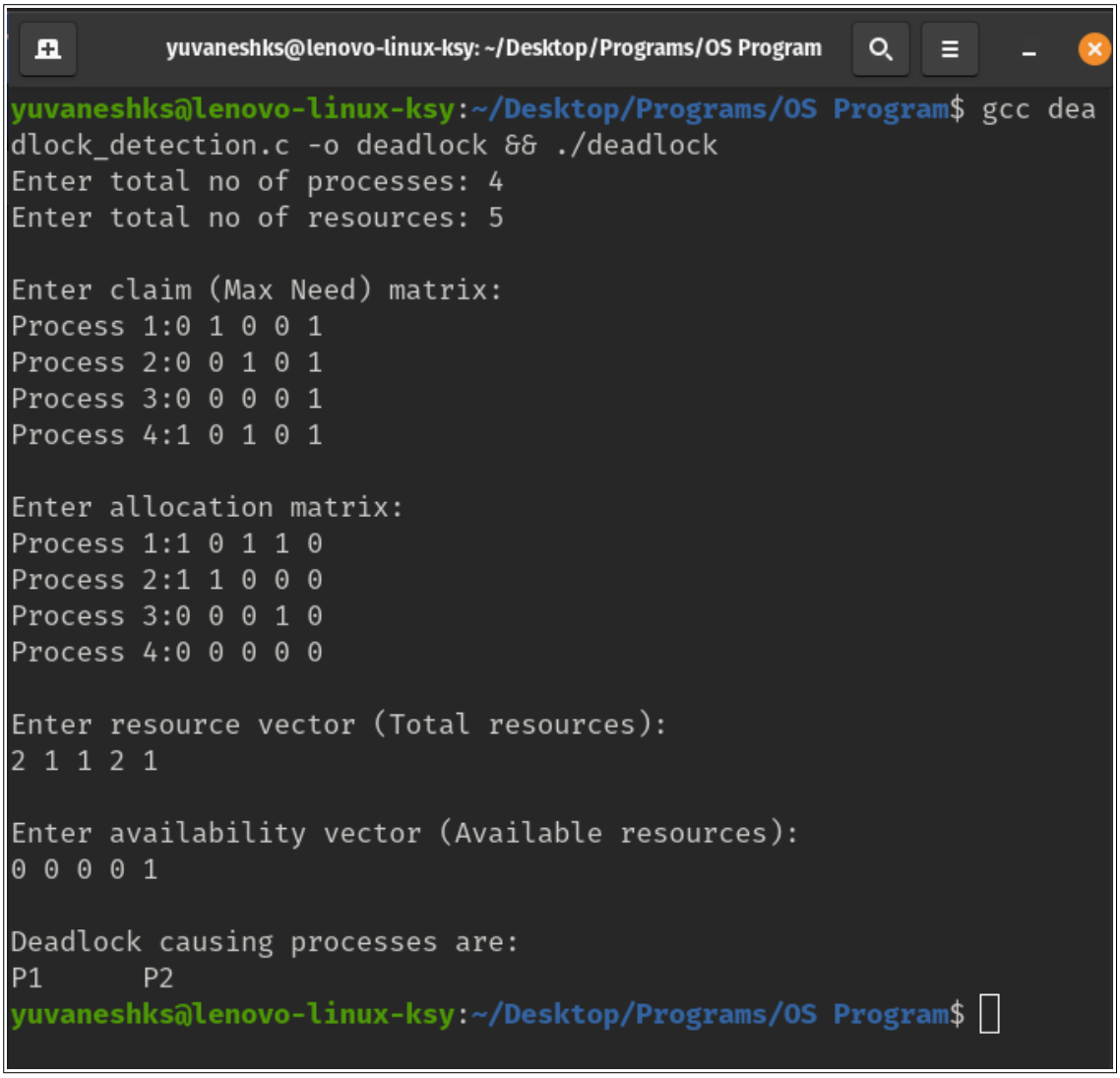

yuvaneshks@lenovo-linux-ksy: ~/Desktop/Programs/OS Program
Available resources:
2 3 0
Allocated Matrix Max Matrix Need Matrix
0 1 0 | 7 5 3 | 7 4 3
3 0 2 | 3 2 2 | 0 2 0
3 0 2 | 9 0 2 | 6 0 0
2 1 1 | 2 2 2 | 0 1 1
0 0 2 | 4 3 3 | 4 3 1
Process 2 completed
Available resources: 5 3 2
Allocated Matrix Max Matrix Need Matrix
0 1 0 | 7 5 3 | 7 4 3
0 0 0 | 0 0 0 | 0 0 0
3 0 2 | 9 0 2 | 6 0 0
2 1 1 | 2 2 2 | 0 1 1
0 0 2 | 4 3 3 | 4 3 1
Process 4 completed
Available resources: 7 4 3
Allocated Matrix Max Matrix Need Matrix
0 1 0 | 7 5 3 | 7 4 3
0 0 0 | 0 0 0 | 0 0 0
3 0 2 | 9 0 2 | 6 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 2 | 4 3 3 | 4 3 1
Process 1 completed
Available resources: 7 5 3
Allocated Matrix Max Matrix Need Matrix


yuvaneshks@lenovo-linux-ksy: ~/Desktop/Programs/OS Program
Process 4 completed
Available resources: 7 4 3
Allocated Matrix Max Matrix Need Matrix
0 1 0 | 7 5 3 | 7 4 3
0 0 0 | 0 0 0 | 0 0 0
3 0 2 | 9 0 2 | 6 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 2 | 4 3 3 | 4 3 1
Process 1 completed
Available resources: 7 5 3
Allocated Matrix Max Matrix Need Matrix
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
3 0 2 | 9 0 2 | 6 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 2 | 4 3 3 | 4 3 1
Process 3 completed
Available resources: 10 5 5
Allocated Matrix Max Matrix Need Matrix
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 2 | 4 3 3 | 4 3 1
Process 5 completed
Available resources: 10 5 7
The system is in a safe state!!
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$

EX : 7 DeadLock Avoidance

```
#include <stdio.h>

void main() {
   int found, flag, l, tp, tr, i, j, k = 1, sum = 0;
   int p[10][10], c[10][10], m[10], r[10], a[10], temp[10];

   printf("Enter total no of processes: "); scanf("%d", &tp);
   printf("Enter total no of resources: "); scanf("%d", &tr);

   printf("\nEnter claim (Max Need) matrix:\n");
   for (i = 1; i <= tp; i++) { printf("Process %d:", i);
   for (j = 1; j <= tr; j++) { scanf("%d", &c[i][j]); } }

   printf("\nEnter allocation matrix:\n");
   for (i = 1; i <= tp; i++) { printf("Process %d:", i);
   for (j = 1; j <= tr; j++) { scanf("%d", &p[i][j]); } }

   printf("\nEnter resource vector (Total resources):\n");
   for (i = 1; i <= tr; i++) { scanf("%d", &r[i]); }

   printf("\nEnter availability vector (Available resources):\n");
   for (i = 1; i <= tr; i++) { scanf("%d", &a[i]); temp[i] = a[i]; }

   for (i = 1; i <= tp; i++) {
         sum = 0;
         for (j = 1; j <= tr; j++) { sum += p[i][j]; }
         if (sum == 0) { m[k++] = i; }
   }

   for (i = 1; i <= tp; i++) {
      flag = 1;
      for (j = 1; j <= tr; j++) {
         if (c[i][j] - p[i][j] > temp[j]) { flag = 0; break; }
      }
      if (flag == 1) {
         m[k++] = i;  for (j = 1; j <= tr; j++) { temp[j] += p[i][j]; }
```

```
        }
    }

    printf("\nDeadlock causing processes are:\n");
    for (j = 1; j <= tp; j++) {
        found = 0;
        for (i = 1; i < k; i++) {
            if (j == m[i]) { found = 1; break; }
        }
        if (found == 0) { printf("P%d\t", j); }
    }
    printf("\n");
}
```



EX : 8 DeadLock Avoidance

```
#include <stdio.h>

void main() {
int found, flag, l, tp, tr, i, j, k = 1, sum = 0;
int p[10][10], c[10][10], m[10], r[10], a[10], temp[10];

printf("Enter total no of processes: "); scanf("%d", &tp);
printf("Enter total no of resources: "); scanf("%d", &tr);

printf("\nEnter claim (Max Need) matrix:\n");
for (i = 1; i <= tp; i++) { printf("Process %d:", i);
for (j = 1; j <= tr; j++) { scanf("%d", &c[i][j]); } }

printf("\nEnter allocation matrix:\n");
for (i = 1; i <= tp; i++) { printf("Process %d:", i);
for (j = 1; j <= tr; j++) { scanf("%d", &p[i][j]); } }

printf("\nEnter resource vector (Total resources):\n");
for (i = 1; i <= tr; i++) { scanf("%d", &r[i]); }

printf("\nEnter availability vector (Available resources):\n");
for (i = 1; i <= tr; i++) { scanf("%d", &a[i]); temp[i] = a[i]; }

for (i = 1; i <= tp; i++) {
sum = 0;
for (j = 1; j <= tr; j++) { sum += p[i][j]; }
if (sum == 0) { m[k++] = i; }
}

for (i = 1; i <= tp; i++) {
flag = 1;
for (j = 1; j <= tr; j++) {
if (c[i][j] - p[i][j] > temp[j]) { flag = 0; break; }
}
if (flag == 1) {
m[k++] = i;
for (j = 1; j <= tr; j++) { temp[j] += p[i][j]; }
}
}

printf("\nDeadlock causing processes are:\n");
```

```
for (j = 1; j <= tp; j++) {
found = 0;
for (i = 1; i < k; i++) {
if (j == m[i]) { found = 1; break; }
}
if (found == 0) { printf("P%d\t", j); }
}

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

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ ./avoid
ance
********** Deadlock Detection Algorithm ************
Enter the number of processes: 3
Enter the number of resource types: 3
Enter the Max Matrix:
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix:
3 3 3
2 0 3
1 2 4
Enter the Available Resources:
1 2 0

Process Allocation      Max             Need
P1      3 3 3   3 6 8   0 3 5
P2      2 0 3   4 3 3   2 3 0
P3      1 2 4   3 4 4   2 2 0

Available Resources: 1 2 0

System is in Deadlock.
Deadlocked processes are: P1 P2 P3
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 9 IMPLEMENTATION OF THREADING

Program:

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

pthread_t tid[2];

// Thread function
void* doSomeThing(void *arg) {
    unsigned long i = 0;
    pthread_t id = pthread_self();

    if (pthread_equal(id, tid[0])) {
        printf("\n[Thread 1] First thread is processing...\n");
    } else {
        printf("\n[Thread 2] Second thread is processing...\n");
    }

    // Simulate some work
    for (i = 0; i < 0xFFFFFFF; i++);

    printf("[Thread %ld] Finished work.\n", id);
    return NULL;
}

int main(void) {
    int i = 0;
    int err;

    printf("=== POSIX Thread Example ===\n");

    // Create two threads
    while (i < 2) {
        err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
        if (err != 0) {
            printf("\n[Error] Can't create thread %d: [%s]\n", i + 1,
strerror(err));
        } else {
            printf("[Main] Thread %d created successfully.\n", i +
1);
        }
        i++;
    }

    // Wait for both threads to finish
    pthread_join(tid[0], NULL);
```

```
    pthread_join(tid[1], NULL);

    printf("[Main] All threads completed. Exiting program.\n");

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc thr
eads.c -o threads -pthread
./threads
=== POSIX Thread Example ===

[Thread 1] First thread is processing...
[Main] Thread 1 created successfully.
[Main] Thread 2 created successfully.

[Thread 2] Second thread is processing...
[Thread 128982177543744] Finished work.
[Thread 128982169151040] Finished work.
[Main] All threads completed. Exiting program.
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 10 IMPLEMENTATION OF PAGING TECHNIQUE

Program:

```
#include <stdio.h>

int main() {
    int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
    int s[10], fno[10][20];

    printf("Enter the memory size: ");
    scanf("%d", &ms);

    printf("Enter the page size: ");
    scanf("%d", &ps);

    nop = ms / ps;
    printf("The number of pages available in memory: %d\n", nop);
```

```
    printf("Enter the number of processes: ");
    scanf("%d", &np);

    rempages = nop;

    for (i = 1; i <= np; i++) {
        printf("Enter number of pages required for process %d: ", i);
        scanf("%d", &s[i]);

        if (s[i] > rempages) {
            printf("Memory is full. Cannot allocate pages for process
%d.\n", i);
            break;
        }

        rempages -= s[i];

        printf("Enter page table for process %d:\n", i);
        for (j = 0; j < s[i]; j++) {
            printf("Page %d frame number: ", j);
            scanf("%d", &fno[i][j]);
        }
    }

    printf("\nEnter Logical Address to find Physical Address\n");
    printf("Enter process number, page number, and offset: ");
    scanf("%d %d %d", &x, &y, &offset);

    if (x > np || y >= s[x] || offset >= ps) {
        printf("Invalid process number, page number, or offset.\n");
    } else {
        pa = fno[x][y] * ps + offset;
        printf("The Physical Address is: %d\n", pa);
    }

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc pag
ing.c -o paging
./paging
Enter the memory size: 1000
Enter the page size: 100
The number of pages available in memory: 10
Enter the number of processes: 3
Enter number of pages required for process 1: 4
Enter page table for process 1:
Page 0 frame number: 8 6 9 5
Page 1 frame number: Page 2 frame number: Page 3 frame number: Ent
er number of pages required for process 2: 5
Enter page table for process 2:
Page 0 frame number: 1
Page 1 frame number: 4
Page 2 frame number: 5
Page 3 frame number: 7
Page 4 frame number: 3
Enter number of pages required for process 3: 5
Memory is full. Cannot allocate pages for process 3.

Enter Logical Address to find Physical Address
Enter process number, page number, and offset: 2
3
60
The Physical Address is: 760
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 11 IMPLEMENTATION OF MEMORY ALLOCATION TECHNIQUES

Program:

```c
#include <stdio.h>

int main() {
    int p[10], np, b[10], nb, ch;
    int alloc[10], flag[10], i, j;
```

```
    int c[10], d[10];  // For best and worst fit block copies

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

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

    printf("Enter the size of each process:\n");
    for (i = 0; i < np; i++) {
        printf("Process %d: ", i);
        scanf("%d", &p[i]);
    }

    printf("Enter the block sizes:\n");
    for (j = 0; j < nb; j++) {
        printf("Block %d: ", j);
        scanf("%d", &b[j]);
        c[j] = b[j];  // Copy for Best Fit
        d[j] = b[j];  // Copy for Worst Fit
    }

    if (np <= nb) {
        do {
            printf("\nMemory Allocation Strategies:\n");
            printf("1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\
n");
            printf("Enter your choice: ");
            scanf("%d", &ch);

            for (i = 0; i < np; i++) flag[i] = 1;  // Initially not
allocated

            switch (ch) {
                case 1:  // First Fit
                    printf("\nFirst Fit Allocation:\n");
                    for (i = 0; i < np; i++) {
                        for (j = 0; j < nb; j++) {
                            if (p[i] <= b[j]) {
                                alloc[i] = j;
                                b[j] -= p[i];
                                flag[i] = 0;
                                printf("Process %d of size %d
```

```
allocated in Block %d\n", i, p[i], j);
                                break;
                            }
                        }
                        if (flag[i])
                            printf("Process %d of size %d NOT
allocated\n", i, p[i]);
                    }
                    break;

                case 2:  // Best Fit
                    printf("\nBest Fit Allocation:\n");
                    // Sort blocks ascending
                    for (i = 0; i < nb - 1; i++) {
                        for (j = i + 1; j < nb; j++) {
                            if (c[i] > c[j]) {
                                int temp = c[i];
                                c[i] = c[j];
                                c[j] = temp;
                            }
                        }
                    }

                    for (i = 0; i < np; i++) {
                        for (j = 0; j < nb; j++) {
                            if (p[i] <= c[j]) {
                                alloc[i] = j;
                                c[j] -= p[i];
                                flag[i] = 0;
                                printf("Process %d of size %d
allocated in Block %d\n", i, p[i], j);
                                break;
                            }
                        }
                        if (flag[i])
                            printf("Process %d of size %d NOT
allocated\n", i, p[i]);
                    }
                    break;

                case 3:  // Worst Fit
                    printf("\nWorst Fit Allocation:\n");
                    // Sort blocks descending
```

```
                    for (i = 0; i < nb - 1; i++) {
                        for (j = i + 1; j < nb; j++) {
                            if (d[i] < d[j]) {
                                int temp = d[i];
                                d[i] = d[j];
                                d[j] = temp;
                            }
                        }
                    }

                    for (i = 0; i < np; i++) {
                        for (j = 0; j < nb; j++) {
                            if (p[i] <= d[j]) {
                                alloc[i] = j;
                                d[j] -= p[i];
                                flag[i] = 0;
                                printf("Process %d of size %d
allocated in Block %d\n", i, p[i], j);
                                break;
                            }
                        }
                        if (flag[i])
                            printf("Process %d of size %d NOT
allocated\n", i, p[i]);
                    }
                    break;

                case 4:
                    printf("Exiting...\n");
                    break;

                default:
                    printf("Invalid choice!\n");
                    break;
            }
        } while (ch != 4);
    } else {
        printf("Error: Number of processes should not exceed number
of blocks.\n");
    }

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc all
ocation.c -o allocation
./allocation
Enter the number of processes: 3
Enter the number of blocks: 3
Enter the size of each process:
Process 0: 100
Process 1: 150
Process 2: 200
Enter the block sizes:
Block 0: 300
Block 1: 350
Block 2: 200

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

First Fit Allocation:
Process 0 of size 100 allocated in Block 0
Process 1 of size 150 allocated in Block 0
Process 2 of size 200 allocated in Block 1

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
```

```
Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

Best Fit Allocation:
Process 0 of size 100 allocated in Block 0
Process 1 of size 150 allocated in Block 1
Process 2 of size 200 allocated in Block 2

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

Worst Fit Allocation:
Process 0 of size 100 allocated in Block 0
Process 1 of size 150 allocated in Block 0
Process 2 of size 200 allocated in Block 1

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 6
Invalid choice!
```

```
Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

Worst Fit Allocation:
Process 0 of size 100 allocated in Block 0
Process 1 of size 150 allocated in Block 0
Process 2 of size 200 allocated in Block 1

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 6
Invalid choice!

Memory Allocation Strategies:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 4
Exiting...
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 12(a) IMPLEMENT FIFO PAGE REPLACEMENT ALGORITHM

Program:

```
#include <stdio.h>
int main() {
    int i, j, k, f, pf = 0, count = 0, rs[25], m[10], n;

    printf("Enter the length of reference string: ");
```

```
    scanf("%d", &n);

    printf("Enter the reference string: ");
    for (i = 0; i < n; i++)
        scanf("%d", &rs[i]);

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

    for (i = 0; i < f; i++)
        m[i] = -1;

    printf("\nPage Replacement Process:\n");

    for (i = 0; i < n; i++) {
        for (k = 0; k < f; k++) {
            if (m[k] == rs[i])  // Page Hit
                break;
        }

        if (k == f) {  // Page Fault
            m[count++] = rs[i];
            pf++;
        }

        for (j = 0; j < f; j++) {
            if (m[j] != -1)
                printf("%d\t", m[j]);
            else
                printf("-\t");
        }

        if (k == f) printf("Page Fault %d", pf);

        printf("\n");

        if (count == f) count = 0;
    }

    printf("\nTotal number of page faults using FIFO: %d\n", pf);

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc fif
o.c -o fifo
./fifo
Enter the length of reference string: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0
1
Enter number of frames: 3

Page Replacement Process:
7       -       -       Page Fault 1
7       0       -       Page Fault 2
7       0       1       Page Fault 3
2       0       1       Page Fault 4
2       0       1
2       3       1       Page Fault 5
2       3       0       Page Fault 6
4       3       0       Page Fault 7
4       2       0       Page Fault 8
4       2       3       Page Fault 9
0       2       3       Page Fault 10
0       2       3
0       2       3
0       1       3       Page Fault 11
0       1       2       Page Fault 12
0       1       2
0       1       2
7       1       2       Page Fault 13
7       0       2       Page Fault 14
7       0       1       Page Fault 15

Total number of page faults using FIFO: 15
```

EX : 12(b) IMPLEMENT LRU PAGE REPLACEMENT

Program:

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

```c
int main() {
    int i, j, k, min, rs[25], m[10], count[10], flag[25], n, f, pf =
0, next = 1;

    printf("Enter the length of reference string: ");
    scanf("%d", &n);

    printf("Enter the reference string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &rs[i]);
        flag[i] = 0;
    }

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

    for (i = 0; i < f; i++) {
        count[i] = 0;
        m[i] = -1;
    }

    printf("\nThe Page Replacement Process is:\n");

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

        for (j = 0; j < f; j++) {
            if (m[j] == rs[i]) {
                found = 1;
                count[j] = next++;
                break;
            }
        }

        if (!found) {
            pf++;
            int pos = -1;
            for (j = 0; j < f; j++) {
                if (m[j] == -1) {
                    pos = j;
                    break;
                }
            }
```

```
            if (pos == -1) {
                // Find least recently used
                min = 0;
                for (j = 1; j < f; j++) {
                    if (count[j] < count[min])
                        min = j;
                }
                pos = min;
            }

            m[pos] = rs[i];
            count[pos] = next++;
        }

        for (j = 0; j < f; j++) {
            if (m[j] != -1)
                printf("%d\t", m[j]);
            else
                printf("-\t");
        }

        if (!found)
            printf("PF No. -- %d", pf);
        printf("\n");
    }

    printf("\nTotal number of page faults using LRU: %d\n", pf);

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc lru
.c -o lru
./lru
Enter the length of reference string: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0
1
Enter the number of frames: 3

The Page Replacement Process is:
7       -       -       PF No. -- 1
7       0       -       PF No. -- 2
7       0       1       PF No. -- 3
2       0       1       PF No. -- 4
2       0       1
2       0       3       PF No. -- 5
2       0       3
4       0       3       PF No. -- 6
4       0       2       PF No. -- 7
4       3       2       PF No. -- 8
0       3       2       PF No. -- 9
0       3       2
0       3       2
1       3       2       PF No. -- 10
1       3       2
1       0       2       PF No. -- 11
1       0       2
1       0       7       PF No. -- 12
1       0       7
1       0       7

Total number of page faults using LRU: 12
```

EX : 12(c) IMPLEMENT LFU PAGE REPLACEMENT ALGORITHM

Program:

```c
#include <stdio.h>

int main() {
    int rs[50], i, j, k, m, f;
```

```c
    int cntr[20], a[20], min, pf = 0;

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

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

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

    for (i = 0; i < f; i++) {
        cntr[i] = 0;
        a[i] = -1;
    }

    printf("\nThe Page Replacement Process is:\n");

    for (i = 0; i < m; i++) {
        int found = 0;
        for (j = 0; j < f; j++) {
            if (rs[i] == a[j]) {
                cntr[j]++; // increase frequency
                found = 1;
                break;
            }
        }

        if (!found) {
            int pos = -1;

            // Check for empty frame first
            for (j = 0; j < f; j++) {
                if (a[j] == -1) {
                    pos = j;
                    break;
                }
            }

            // If no empty frame, find LFU
            if (pos == -1) {
```

```
                min = 0;
                for (j = 1; j < f; j++) {
                    if (cntr[j] < cntr[min])
                        min = j;
                }
                pos = min;
            }

            a[pos] = rs[i];
            cntr[pos] = 1;
            pf++;
        }

        for (j = 0; j < f; j++) {
            if (a[j] != -1)
                printf("%d\t", a[j]);
            else
                printf("-\t");
        }
        if (!found)
            printf("PF No. %d", pf);
        printf("\n");
    }

    printf("\nTotal number of page faults: %d\n", pf);

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc lfu
.c -o lfu && ./lfu
Enter number of page references: 10
Enter the reference string: 1 2 3 4 5 2 5 1 4 3
Enter the available number of frames: 3

The Page Replacement Process is:
1       -       -       PF No. 1
1       2       -       PF No. 2
1       2       3       PF No. 3
4       2       3       PF No. 4
5       2       3       PF No. 5
5       2       3
5       2       3
5       2       1       PF No. 6
5       2       4       PF No. 7
5       2       3       PF No. 8

Total number of page faults: 8
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 13 IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES

A) SINGLE LEVEL DIRECTORY

Program:

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

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

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

```
    dir.fcnt = 0;

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

    while(1) {
        printf("\n\n1. Create File\t2. Delete File\t3. Search File\
n4. Display Files\t5. Exit\nEnter your choice -- ");
        scanf("%d", &ch);

        switch(ch) {
            case 1:
                printf("\nEnter the name of the file -- ");
                scanf("%s", dir.fname[dir.fcnt]);
                dir.fcnt++;
                break;
            case 2:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for(i = 0; i < dir.fcnt; i++) {
                    if(strcmp(f, dir.fname[i]) == 0) {
                        printf("File %s is deleted\n", f);
                        strcpy(dir.fname[i], dir.fname[dir.fcnt-1]);
                        dir.fcnt--;
                        break;
                    }
                }
                if(i == dir.fcnt)
                    printf("File %s not found\n", f);
                break;
            case 3:
                printf("\nEnter the name of the file -- ");
                scanf("%s", f);
                for(i = 0; i < dir.fcnt; i++) {
                    if(strcmp(f, dir.fname[i]) == 0) {
                        printf("File %s is found\n", f);
                        break;
                    }
                }
                if(i == dir.fcnt)
                    printf("File %s not found\n", f);
                break;
            case 4:
```

```
                if(dir.fcnt == 0)
                    printf("\nDirectory Empty\n");
                else {
                    printf("\nThe Files are -- ");
                    for(i = 0; i < dir.fcnt; i++)
                        printf("\t%s", dir.fname[i]);
                    printf("\n");
                }
                break;
            case 5:
                printf("\nExiting...\n");
                exit(0);
            default:
                printf("\nInvalid choice. Please try again.\n");
        }
    }
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc dir
ectory_management.c -o directory_management && ./directory_managem
ent

Enter name of directory -- CSE


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 1

Enter the name of the file -- A


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- B

Enter the name of the file --

1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 1

Enter the name of the file -- C


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 4
```

```
1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 4

The Files are --        A       B       C


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 3

Enter the name of the file -- ABC
File ABC not found


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 2

Enter the name of the file -- B
File B is deleted


1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice -- 5

Exiting...
```

EX : 13 IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES

B) TWO LEVEL DIRECTORY

Program:

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

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

void main() {
    int i, ch, dcnt = 0, k;
```

```
    char f[30], d[30];

    while (1) {
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete
File");
        printf("\n4. Search File\t\t5. Display\t6. Exit\nEnter your
choice -- ");
        scanf("%d", &ch);

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

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

            case 3:
                printf("\nEnter name of the directory -- ");
                scanf("%s", d);
                for (i = 0; i < dcnt; i++) {
                    if (strcmp(d, dir[i].dname) == 0) {
                        printf("Enter name of the file -- ");
                        scanf("%s", f);
                        for (k = 0; k < dir[i].fcnt; k++) {
                            if (strcmp(f, dir[i].fname[k]) == 0) {
```

```
                                printf("File %s is deleted", f);
                                strcpy(dir[i].fname[k],
dir[i].fname[dir[i].fcnt - 1]);
                                dir[i].fcnt--;
                                goto jmp;
                            }
                        }
                        printf("File %s not found", f);
                        goto jmp;
                    }
                }
                printf("Directory %s not found", d);
            jmp:
                break;

            case 4:
                printf("\nEnter name of the directory -- ");
                scanf("%s", d);
                for (i = 0; i < dcnt; i++) {
                    if (strcmp(d, dir[i].dname) == 0) {
                        printf("Enter the name of the file -- ");
                        scanf("%s", f);
                        for (k = 0; k < dir[i].fcnt; k++) {
                            if (strcmp(f, dir[i].fname[k]) == 0) {
                                printf("File %s is found", f);
                                goto jmp1;
                            }
                        }
                        printf("File %s not found", f);
                        goto jmp1;
                    }
                }
                printf("Directory %s not found", d);
            jmp1:
                break;

            case 5:
                if (dcnt == 0)
                    printf("\nNo Directories");
                else {
                    printf("\nDirectory\tFiles");
                    for (i = 0; i < dcnt; i++) {
                        printf("\n%s\t\t", dir[i].dname);
```

```
                        for (k = 0; k < dir[i].fcnt; k++)
                            printf("%s\t", dir[i].fname[k]);
                    }
                }
                break;

            case 6:
                exit(0);

            default:
                printf("Invalid Choice!");
        }
    }
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc dir
ectory.c -o directory && ./directory


1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit
Enter your choice -- 1

Enter name of directory -- DIR
Directory created

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit
Enter your choice -- 1

Enter name of directory -- DIR2
Directory created

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit
Enter your choice -- 2

Enter name of the directory -- DIR
Enter name of the file -- A1
File created

1. Create Directory     2. Create File  3. Delete File
4. Search File          5. Display      6. Exit
Enter your choice -- 2
```

EX : 14(a) IMPLEMENT SEQUENTIAL FILE ALLOCATION TECHNIQUE

Program:

```
#include <stdio.h>

int main() {
    int f[50], i, st, j, len, c;

    for (i = 0; i < 50; i++)
        f[i] = 0;

    do {
        printf("\nEnter the starting block & length of the file: ");
        scanf("%d%d", &st, &len);

        int allocated = 1;

        // Check if blocks are available
        for (j = st; j < (st + len); j++) {
            if (f[j] != 0) {
                printf("Block %d already allocated.\n", j);
                allocated = 0;
                break;
            }
        }

        // If available, allocate them
        if (allocated) {
            for (j = st; j < (st + len); j++) {
                f[j] = 1;
                printf("%d -> Allocated\n", j);
            }
            printf("The file is successfully allocated to disk.\n");
        }

        printf("\nDo you want to enter more files? (1-Yes / 0-No):
");
        scanf("%d", &c);

    } while (c == 1);
```

```
    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc seq
uential.c -o sequential && ./sequential

Enter the starting block & length of the file: 4
104->1
Block 50 already allocated.

Do you want to enter more files? (1-Yes / 0-No): yuvaneshks@lenovo
-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 14(b) LINKED FILE ALLOCATION

Program:

```
#include <stdio.h>

int main() {
    int f[50], p, i, j, k, a, st, len, c;

    // Initialize all blocks to 0 (free)
    for (i = 0; i < 50; i++)
        f[i] = 0;

    printf("Enter how many blocks are already allocated: ");
    scanf("%d", &p);

    printf("Enter the block numbers that are already allocated: ");
    for (i = 0; i < p; i++) {
        scanf("%d", &a);
        f[a] = 1;
    }

    do {
        printf("\nEnter the starting index block and length: ");
        scanf("%d%d", &st, &len);

        k = len;
```

```
        for (j = st; j < (st + k); j++) {
            if (f[j] == 0) {
                f[j] = 1;
                printf("%d -> Allocated\n", j);
            } else {
                printf("%d -> Already allocated\n", j);
                // extend to get required number of free blocks
                k++;
            }
        }

        printf("Do you want to enter one more file? (yes-1 / no-0):
");
        scanf("%d", &c);

    } while (c == 1);

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc ind
exed.c -o indexed && ./indexed
Enter how many blocks are already allocated: 3
Enter the block numbers that are already allocated: 4 7 9

Enter the starting index block and length: 3 7
3 -> Allocated
4 -> Already allocated
5 -> Allocated
6 -> Allocated
7 -> Already allocated
8 -> Allocated
9 -> Already allocated
10 -> Allocated
11 -> Allocated
12 -> Allocated
Do you want to enter one more file? (yes-1 / no-0): 0
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 14(c) INDEXED FILE ALLOCATION

Program:

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

int main() {
    int f[50], i, j, k, indexBlock, blockCount, blockList[50],
moreFiles = 1;

    // Initialize all blocks as free
    for(i = 0; i < 50; i++)
        f[i] = 0;

    while(moreFiles == 1) {
        printf("\nEnter index block: ");
        scanf("%d", &indexBlock);

        if(f[indexBlock] == 0) {
            f[indexBlock] = 1;

            printf("Enter number of blocks for the file: ");
            scanf("%d", &blockCount);

            printf("Enter the block numbers:\n");
            for(i = 0; i < blockCount; i++)
                scanf("%d", &blockList[i]);

            // Check if blocks are already allocated
            for(i = 0; i < blockCount; i++) {
                if(f[blockList[i]] == 1) {
                    printf("Block %d is already allocated! Try
again.\n", blockList[i]);
                    f[indexBlock] = 0;
                    goto end;
                }
            }

            // Allocate blocks
            for(j = 0; j < blockCount; j++)
```

```
                f[blockList[j]] = 1;

            printf("Allocated Successfully.\n");
            printf("File Indexed:\n");
            for(k = 0; k < blockCount; k++)
                printf("%d -> %d : Allocated\n", indexBlock,
blockList[k]);

        } else {
            printf("Index block %d is already allocated. Try again.\
n", indexBlock);
        }

        end:
        printf("\nEnter 1 to enter more files, 0 to exit: ");
        scanf("%d", &moreFiles);
    }

    return 0;
}
```

Output:

```
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$ gcc ina
l.c -o inal && ./inal

Enter index block: 9
Enter number of blocks for the file: 3 1 2 3
Enter the block numbers:
Allocated Successfully.
File Indexed:
9 -> 1 : Allocated
9 -> 2 : Allocated
9 -> 3 : Allocated

Enter 1 to enter more files, 0 to exit: 0
yuvaneshks@lenovo-linux-ksy:~/Desktop/Programs/OS Program$
```

EX : 15 DISK SCHEDULING ALGORITHMS

FCFS Disk Scheduling

Program:

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

```c
main() {
    int t[20], n, i, j, tohm[20], tot=0;
    float avhm;
    clrscr();
    printf("enter the no.of tracks");
    scanf("%d", &n);
    printf("enter the tracks to be traversed");
    for(i=2; i<n+2; i++)
        scanf("%d", &t[i]);
    for(i=1; i<n+1; i++) {
        tohm[i] = t[i+1] - t[i];
        if(tohm[i] < 0)
            tohm[i] = -tohm[i];
    }
    for(i=1; i<n+1; i++)
        tot += tohm[i];
    avhm = (float)tot / n;
    printf("Tracks traversed\tDifference between tracks\n");
    for(i=1; i<n+1; i++)
        printf("%d\t\t\t%d\n", t[i], tohm[i]);
    printf("\nAverage header movements: %f", avhm);
    getch();
}
```

Output:

```
Enter no.of tracks: 9
Enter track position: 55 58 60 70 18 90 150 160 184

Tracks traversed        Difference between tracks
55                      45
58                      3
60                      2
70                      10
18                      52
90                      72
150                     60
160                     10
184                     24

Average header movements: 30.888889
```

SCAN Disk Scheduling

Program:

```
#include<stdio.h>
main() {
    int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
    clrscr();
    printf("enter the no of tracks to be traversed");
    scanf("%d", &n);
    printf("enter the position of head");
    scanf("%d", &h);
    t[0]=0;
    t[1]=h;
    printf("enter the tracks");
    for(i=2; i<n+2; i++)
        scanf("%d", &t[i]);
    for(i=0; i<n+2; i++) {
        for(j=0; j<(n+2)-i-1; j++) {
            if(t[j] > t[j+1]) {
                temp = t[j];
                t[j] = t[j+1];
                t[j+1] = temp;
            }
        }
    }
    for(i=0; i<n+2; i++)
        if(t[i] == h)
            j = i;
    k = i;
    p = 0;
    while(t[j] != 0) {
        atr[p] = t[j];
        j--;
        p++;
    }
    atr[p] = t[j];
    for(p=k+1; p<n+2; p++, k++)
        atr[p] = t[k+1];
    for(j=0; j<n+1; j++) {
        if(atr[j] > atr[j+1])
            d[j] = atr[j] - atr[j+1];
        else
```

```
            d[j] = atr[j+1] - atr[j];
        sum += d[j];
    }
    printf("\nAverage header movements: %f", (float)sum/n);
    getch();
}
```

Output:

```
Enter no.of tracks: 9
Enter track position: 55 58 60 70 18 90 150 160 184

Tracks traversed        Difference between tracks
150                     50
160                     10
184                     24
90                      94
70                      20
60                      10
58                      2
55                      3
18                      37

Average header movements: 27.77
```

C-SCAN Disk Scheduling

Program:

```
#include<stdio.h>
main() {
    int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
    clrscr();
    printf("enter the no of tracks to be traversed");
    scanf("%d", &n);
    printf("enter the position of head");
    scanf("%d", &h);
    t[0]=0;
    t[1]=h;
    printf("enter total tracks");
    scanf("%d", &tot);
    t[2] = tot - 1;
    printf("enter the tracks");
```

```
    for(i=3; i<=n+2; i++)
        scanf("%d", &t[i]);
    for(i=0; i<=n+2; i++) {
        for(j=0; j<=(n+2)-i-1; j++) {
            if(t[j] > t[j+1]) {
                temp = t[j];
                t[j] = t[j+1];
                t[j+1] = temp;
            }
        }
    }
    for(i=0; i<=n+2; i++)
        if(t[i] == h) {
            j = i;
            break;
        }
    p = 0;
    while(t[j] != tot-1) {
        atr[p] = t[j];
        j++;
        p++;
    }
    atr[p] = t[j];
    p++;
    i = 0;
    while(p != (n+3) && t[i] != h) {
        atr[p] = t[i];
        i++;
        p++;
    }
    for(j=0; j<n+2; j++) {
        if(atr[j] > atr[j+1])
            d[j] = atr[j] - atr[j+1];
        else
            d[j] = atr[j+1] - atr[j];
        sum += d[j];
    }
    printf("total header movements: %d", sum);
    printf("\navg is %f", (float)sum/n);
    getch();
}
```

Output:

```
Enter the track position: 55 58 60 70 18 90 150 160 184
Enter starting position: 100

Tracks traversed        Difference Between tracks
150                     50
160                     10
184                     24
18                      240
55                      37
58                      3
60                      2
70                      10
90                      29

Average seek time: 35.777778
```