ASSIGNMENT 2

OS LAB REPORT

NAME: ROHIT SADHU

ROLL NO.: 002010501074

1. CPU SCHEDULER:

FCFS:

```
#include <bits/stdc++.h>
using namespace std;
void FCFS(unordered_map<int,vector<int>> m){
   priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
   // vector<int> ans;

   int time = 0;
   for(auto c : m){
      int jobID = c.first;
      int arrivaltime = c.second[1];
      int priority = c.second[0];
      int burstTime = c.second[2];
      pq.push({arrivaltime,jobID});
   }
   time = pq.top().first;
   while(!pq.empty()){
      auto it = pq.top();
      pq.pop();
      int arrivalTime = it.first;
      int burstTime = m[it.second][2];
      int completionTime = time + burstTime;
      time = completionTime;
      int tat = completionTime - arrivalTime;
      int wt = tat - burstTime;
      cout<<"JOB ID: "<<it.second<<" "<<"Turnaround Time: "<<tat<<" "<<"Waiting Time:
"<<wt<<endl;
   }
}
```

ROUND ROBIN:

```
#include <bits/stdc++.h>
using namespace std;
void RoundRobin(unordered_map<int,vector<int>> m, int timeSlice){
   priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
   // vector<int> ans;

   int time = 0;
   for(auto c : m){
      int jobID = c.first;
      int arrivaltime = c.second[1];
      int priority = c.second[0];
      int burstTime = c.second[2];
      pq.push({arrivaltime,jobID});
   }
   time = pq.top().first;
   queue<pair<int,int>> q;
   while(!pq.empty()){
      auto it  = pq.top();
      pq.pop();
      // int arrivalTime = it.first;
      int jobID = it.second;
      int burstTime = m[it.second][2];
      if(burstTime <= timeSlice){
         int arrivalTime = it.first;
         // int burstTime = m[it.second][2];
         int completionTime = time + burstTime;
         time = completionTime;
         int tat = completionTime - arrivalTime;
         int wt = tat - burstTime;
```

```
        cout<<"JOB ID completed: "<<it.second<<" "<<"Turnaround Time: "<<tat<<" "<<"Waiting Time: "<<wt<<endl;
      }
      else{
        time += timeSlice;
        m[it.second][2] -= timeSlice;
        q.push({burstTime, jobID});
      }
    }
    while(!q.empty()){
      auto it = q.front();
      q.pop();
      int leftburstTime = m[it.second][2];
      if(leftburstTime <= timeSlice){
        int completionTime = time + leftburstTime;
        time = completionTime;
        int tat = completionTime - m[it.second][1];
        int wt = tat - leftburstTime;
        cout<<"JOB ID completed: "<<it.second<<" "<<"Turnaround Time: "<<tat<<" "<<"Waiting Time: "<<wt<<endl;
      }
      else{
        time += timeSlice;
        m[it.second][2] -= timeSlice;
        q.push({it.first,it.second});
      }
    }
}
```

MAIN:

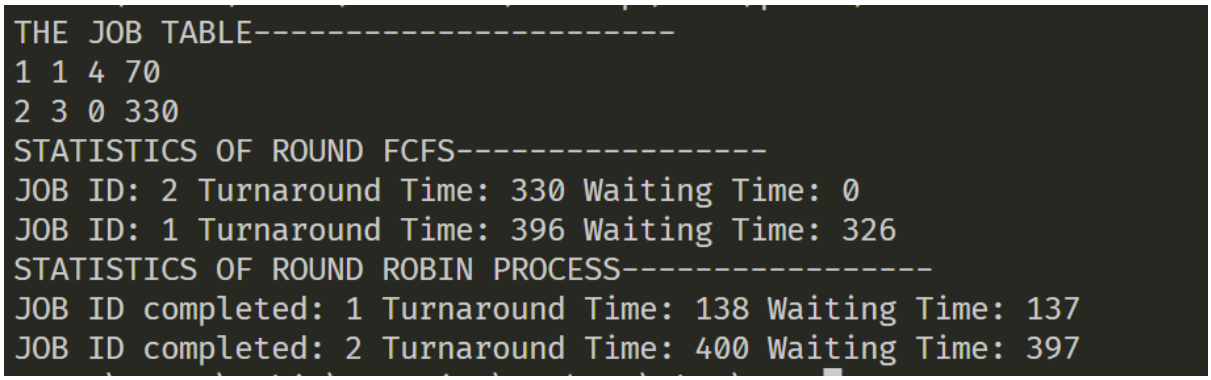
```
#include <bits/stdc++.h>
#include "FCFS.h"
```

```
#include "RoundRobin.h"
#include "PreemtivePriority.h"
#include <fstream>
using namespace std;


int main(){

   string ans = "";
   string b = "";
   ifstream fin;

   // by default open mode = ios::in mode
   fin.open("JobProfile.txt");

   // Execute a loop until EOF (End of File)
   while (fin) {

      // Read a Line from File
      getline(fin, ans);

      // Print line in Console
      // cout << ans << endl;
   }

   // Close the file
   fin.close();
   int n =ans.length();
   int jobProfIndex = 0;
   int bsum = 0;
   int cnt  = 0;
```

```
string temp = "";
unordered_map<int, vector<int>> umap;
for(int i = 0; i<n; i++){
   if(ans[i] == '-' and ans[i+1] == '1'){
      umap[jobProfIndex].push_back(bsum);
      // jobProfIndex++;
      cnt = 0;
      bsum = 0;
      i+=2;

   }
   else{
      if(isdigit(ans[i])){
         temp.push_back(ans[i]);
         // cout<<1;
      }
      else if(ans[i] = ' '){
         cnt++;
         // cout<<temp<<" ";
         int x = stoi(temp);
         temp = "";
         if(cnt == 1)
            jobProfIndex = x;
         if(cnt<=3 and cnt > 1){
            umap[jobProfIndex].push_back(x);
         }
         if(cnt>3){
            bsum += x;
         }
      }
   }
```

```
    }
    cout<<"THE JOB TABLE----------------------\n";
    for(auto c : umap){
        cout<<c.first<<" ";
        for(auto t : c.second){
            cout<<t<<" ";
        }
        cout<<endl;
    }

    umap[2] = {3,0,330};
    umap[1] = {1,4,70};
    cout<<"STATISTICS OF ROUND FCFS-----------------"<<endl;
    FCFS(umap);
    cout<<"STATISTICS OF ROUND ROBIN PROCESS-----------------"<<endl;
    RoundRobin(umap,3);
    cout<<"STATISTICS OF PRIORITY PREEMTIVE PROCESS-----------------"<<endl;
    PreemtivePriority(umap);
}
```

OUTPUT:



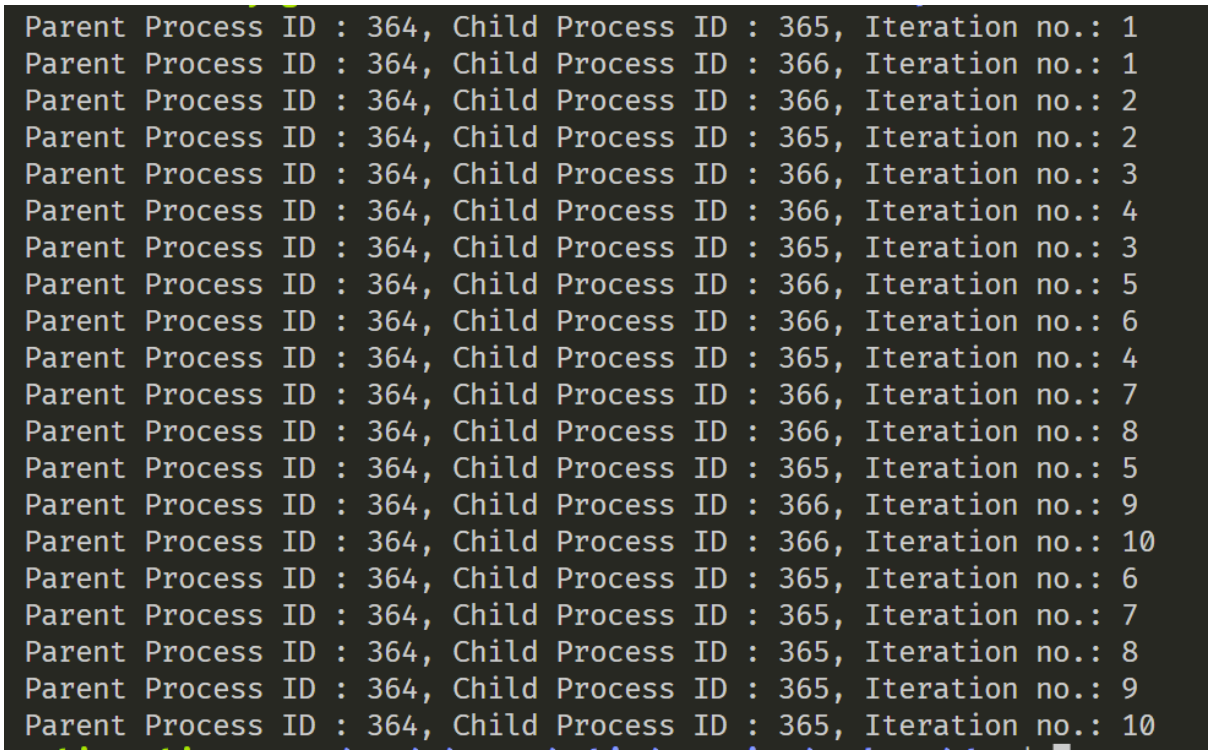
2. CHILD X & Y

a)

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

int main(int argc, char** argv){
   pid_t child_x, child_y;
   pid_t parent = getpid();
   child_x = fork();
   if (child_x < 0){
      printf("Error while forking!");
      exit(EXIT_FAILURE);
   }else if(child_x == 0){
      for (int i = 0; i < 10; i++){
         printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", parent, getpid(),
i+1);
         sleep(2);
      }

   }else{
      child_y = fork();
      if (child_y < 0){
         printf("Error while forking!");
         exit(EXIT_FAILURE);
      }else if(child_y == 0){
         for (int i = 0; i < 10; i++){
            printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", parent,
getpid(), i+1);
            sleep(1);
```

```
        }
      }
    }
    wait(NULL);
    wait(NULL);
    return 0;
}
```

OUTPUT:



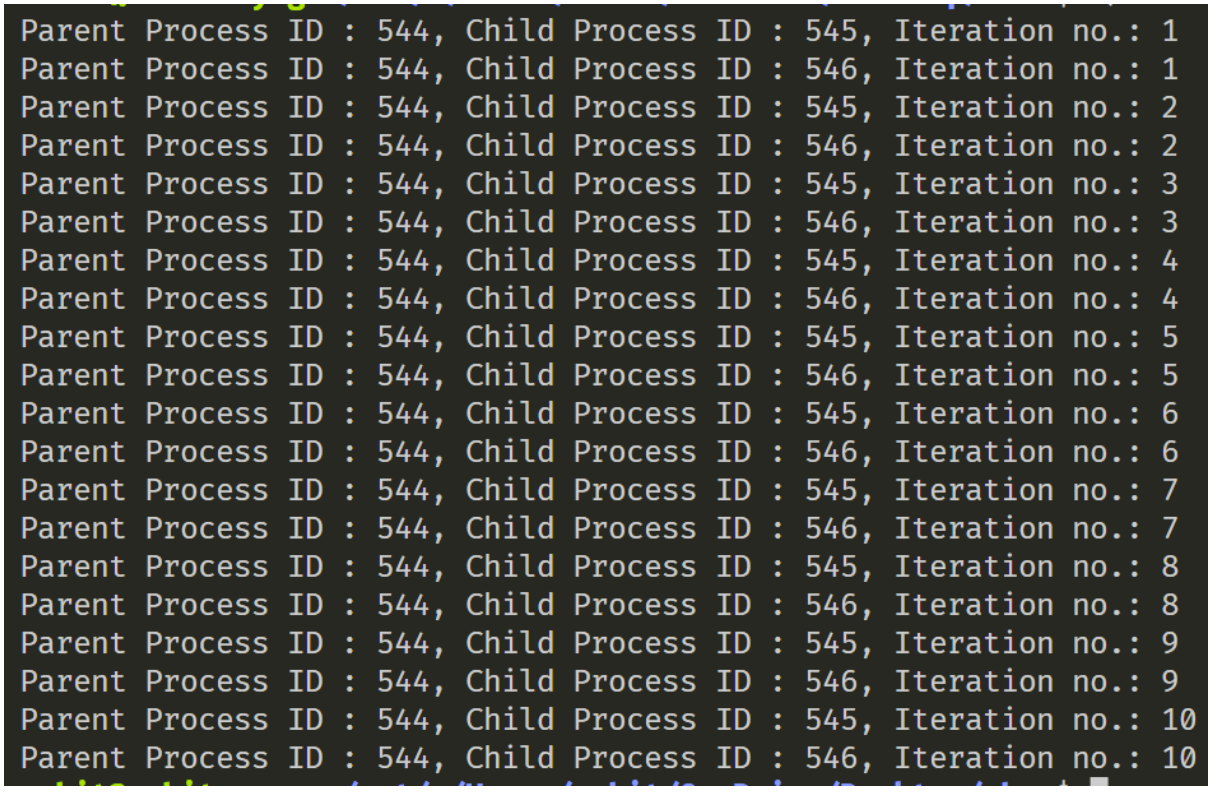
b)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/types.h>
#include <wait.h>
#include <time.h>
```

```
#define S1 "sem1"
#define S2 "sem2"

void display(unsigned int t, sem_t* sw, sem_t* sp){
   for (int i = 0; i < 10; i++){
      sem_wait(sw);
      printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", getppid(), getpid(), i+1);
      sem_post(sp);
      sleep(t);
   }
}

int main(int argc, char** argv){
   sem_t *p_sem1, *p_sem2;
   pid_t pidx = -1, pidy = -1;
   unsigned int shared = 0;
   if((p_sem1 = sem_open(S1, O_CREAT, 0666, 1)) == SEM_FAILED || (p_sem2 = sem_open(S2, O_CREAT, 0666, 0)) == SEM_FAILED){
      printf("Error while opening semaphore!");
      exit(EXIT_FAILURE);
   }
   if ((pidx = fork()) < 0){
      printf("Error while forking!");
      exit(EXIT_FAILURE);
   }else if (pidx == 0){
      display(1, p_sem1, p_sem2);
   }else{
      if ((pidy = fork()) < 0){
         printf("Error while forking!");
         exit(EXIT_FAILURE);
      }else if (pidy == 0){
```

```
            display(1, p_sem2, p_sem1);
        }else{
            wait(NULL);
            wait(NULL);
            if (sem_unlink(S1) == -1 || sem_unlink(S2) == -1){
                perror("Semaphore unlink failed!");
                return 1;
            }
        }
    }
    return 0;
}
```

OUTPUT:



c)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/types.h>
#include <wait.h>
#include <time.h>
#define MUTEX "mutex"

void display(unsigned int t, sem_t* mutex){
   for (int i = 0; i < 10; i++){
      //if(i != 0)
      sem_wait(mutex);
      printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", getppid(), getpid(),
i+1);
      sleep(t);
      sem_post(mutex);
   }
}

int main(int argc, char** argv){
   sem_t *mutex;
   pid_t pidx = -1, pidy = -1;
   unsigned int shared = 0;
   sem_unlink(MUTEX);
   if((mutex = sem_open(MUTEX, O_CREAT, 0666, 2)) == SEM_FAILED){
      printf("Error while opening semaphore!");
      exit(EXIT_FAILURE);
   }
```

```
    if ((pidx = fork()) < 0){
        printf("Error while forking!");
        exit(EXIT_FAILURE);
    }else if (pidx == 0){
        display(1, mutex);
    }else{
        if ((pidy = fork()) < 0){
            printf("Error while forking!");
            exit(EXIT_FAILURE);
        }else if (pidy == 0){
            display(1, mutex);
        }else{
            wait(NULL);
            wait(NULL);
            if (sem_unlink(MUTEX) == -1){
                perror("Semaphore unlink failed!");
                return 1;
            }
        }
    }
    return 0;
}
```

OUTPUT:

```
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 1
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 1
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 2
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 2
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 3
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 3
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 4
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 4
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 5
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 5
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 6
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 6
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 7
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 7
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 8
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 8
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 9
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 9
Parent Process ID : 693, Child Process ID : 695, Iteration no.: 10
Parent Process ID : 693, Child Process ID : 694, Iteration no.: 10
```

d)

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>
#include <semaphore.h>
#include <fcntl.h>

sem_t* mutexX;
sem_t* mutexY;
sem_t* mutexZ;
int main(int argc, char** argv){
   sem_unlink("mutexX");
   sem_unlink("mutexY");
   sem_unlink("mutexZ");

   mutexX = sem_open("mutexX", O_CREAT, 0644, 0);
   mutexY = sem_open("mutexY", O_CREAT, 0644, 0);
   mutexZ = sem_open("mutexZ", O_CREAT, 0644, 0);
   pid_t child_x, child_y, child_z;
   pid_t parent = getpid();
   child_x = fork();
   if (child_x < 0){
      printf("Error while forking!");
      exit(EXIT_FAILURE);
   }else if(child_x == 0){
      for (int i = 0; i < 10; i++){
         int num = rand() % 5;
         if(i != 0){
```

```
            sem_wait(mutexX);
        }
        printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", parent, getpid(),
i+1);
        sleep(num);
        sem_post(mutexY);
    }

  }else{
    child_y = fork();
    if (child_y < 0){
        printf("Error while forking!");
        exit(EXIT_FAILURE);
    }else if(child_y == 0){
        for (int i = 0; i < 10; i++){
            int num = rand() % 5;
            sem_wait(mutexY);
            printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", parent,
getpid(), i+1);
            sleep(num);
            sem_post(mutexZ);
        }
    }else{
        child_z = fork();
        if (child_z < 0){
            printf("Error while forking!");
            exit(EXIT_FAILURE);
        }else if(child_z == 0){
            for (int i = 0; i < 10; i++){
                int num = rand() % 5;
                sem_wait(mutexZ);
```

```
            printf("Parent Process ID : %d, Child Process ID : %d, Iteration no.: %d\n", parent, getpid(), i+1);
            sleep(num);
            sem_post(mutexX);
          }
        }
      }
    }
    wait(NULL);
    wait(NULL);
    wait(NULL);
    sem_destroy(mutexX);
    sem_destroy(mutexY);
    sem_destroy(mutexY);
    return 0;
}
```

OUTPUT:

```
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 1
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 1
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 1
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 2
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 2
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 2
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 3
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 3
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 3
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 4
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 4
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 4
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 5
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 5
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 5
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 6
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 6
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 6
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 7
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 7
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 7
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 8
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 8
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 8
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 9
Parent Process ID : 887, Child Process ID : 889, Iteration no.: 9
Parent Process ID : 887, Child Process ID : 890, Iteration no.: 9
Parent Process ID : 887, Child Process ID : 888, Iteration no.: 10
```

3. IPC

a)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <wait.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#define READ_END 0
#define WRITE_END 1
#define BUFF_LEN 50
int flag = 1;

void __listener_handler(int sig){
   if (sig == SIGINT){
      return;
   }
   if(sig == SIGTERM){
      exit(EXIT_SUCCESS);
   }
}

void __broadcast_signal_handler(int sig){
   flag = 0;
}

int main(int argc, char** argv){
   if (argc != 2){
```

```
    printf("%s: Invalid Arguments\nUsage: %s <station_count>\n", argv[0], argv[0]);
            exit(EXIT_FAILURE);
  }
  int n = atoi(argv[1]);
  pid_t stations[n];
  int fds[n][2];
  int id;
  char buffer[BUFF_LEN];
  for (id = 0; id < n; id++){
    pipe(fds[id]);
    if(stations[id] = fork()){
      goto __listen;
    }else{
      close(fds[id][READ_END]);
    }
  }
  signal(SIGINT, __broadcast_signal_handler);
  srand(time(NULL));
  while (flag) {
    sleep(rand() % 2);
    time_t t;
            time(&t);
            struct tm *time_val = localtime(&t);
            char *arr[] = {"Thunder storm", "Clear sky", "Snowfall", "Overcast", "Haze",
"Drizzles"};
            sprintf(buffer, "%2d:%2d:%2d The weather condition is: %s.", (time_val->tm_hour) %
12, time_val->tm_min,time_val->tm_sec, arr[rand() % 6]);
            int l = strlen(buffer) + 1;
    for (int i = 0; i < n; i++){
      write(fds[i][WRITE_END], buffer, l);
    }
```

```
    }
    for (int i = 0; i < n; i++){
        kill(stations[i], SIGTERM);
    }
    sleep(1);
    printf("Forecast end.");
    fflush(stdout);
    return 0;

    __listen: {
        signal(SIGINT, __listener_handler);
        signal(SIGTERM, __listener_handler);
        for (int i = 0; i < id; i++){
            close(fds[i][WRITE_END]);
        }
        int fd = fds[id][READ_END];
        while (id+1){
            read(fd, buffer, BUFF_LEN);
            printf("Station %d: %s\n", id+1, buffer);
            id--;
        }
    }
}
```

OUTPUT:

```
Station 1: 10:39:57 The weather condition is: Overcast.
Station 2: 10:39:57 The weather condition is: Overcast.
Station 3: 10:39:57 The weather condition is: Overcast.
Station 4: 10:39:57 The weather condition is: Overcast.
Station 7: 10:39:57 The weather condition is: Overcast.
Station 6: 10:39:57 The weather condition is: Overcast.
Station 5: 10:39:57 The weather condition is: Overcast.
Station 8: 10:39:57 The weather condition is: Overcast.
Station 9: 10:39:57 The weather condition is: Overcast.
Station 10: 10:39:57 The weather condition is: Overcast.
```

b)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#define CALLER0_FIFO "/tmp/caller0"
#define CALLER1_FIFO "/tmp/caller1"
#define BUFF_LEN 150

void make_fifo(){
   mkfifo(CALLER0_FIFO, 0666);
   mkfifo(CALLER1_FIFO, 0666);
}

int main(int argc, char** argv){
   if(argc != 2){
      printf("%s: Invalid Arguments\nUsage: %s <caller_no>\n", argv[0], argv[0]);
      exit(EXIT_FAILURE);
   }
   char self_fifo[15], peer_fifo[15];
   char buffer[BUFF_LEN];
   if (argv[1][0] == '0'){
      strncpy(self_fifo, CALLER0_FIFO, 15);
      strncpy(peer_fifo, CALLER1_FIFO, 15);
   }
```

```
    else if (argv[1][0] == '1'){
        strncpy(self_fifo, CALLER1_FIFO, 15);
        strncpy(peer_fifo, CALLER0_FIFO, 15);
    }
    else{
        printf("Please choose among (0/1)!");
        exit(EXIT_FAILURE);
    }
    make_fifo();
    pid_t sender = fork();
    if (sender < 0){
        printf("Unexpected error while forking process!");
        exit(EXIT_FAILURE);
    }else if(sender){
        while(true){
            int fd = open(self_fifo, O_RDONLY);
            read(fd, buffer, BUFF_LEN);
            close(fd);
            printf("%s\n", buffer);
            if(strcmp(buffer, "EOF") == 0) break;
        }
    }else{
        while(true){
            scanf("%[^\n]%*c", buffer);
            int fd = open(peer_fifo, O_WRONLY);
            write(fd, buffer, strlen(buffer)+1);
            close(fd);
        }
    }
    return 0;
}
```

OUTPUT:

4. PRODUCER CONSUMER PROBLEM:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <time.h>
#include <semaphore.h>
#include <sys/wait.h>
#include <sys/signal.h>
#include <sys/mman.h>
#define BUFF_LEN 25
#define PRODUCER_MIN 50
#define PRODUCER_RANGE 51

typedef struct reqmem{
   int total, start, count;
   unsigned char buffer[BUFF_LEN];
   sem_t full, empty, mutex;
}requiredmemory;

int id, count, total;

void consumer_handler(int sig){
   printf("Consumer %d : consumed %d items which adds up to: %d\n", id+1, count, total);
      fflush(stdout);
      raise(SIGKILL);
}

int main(int argc, char** argv){
   total = 0;
```

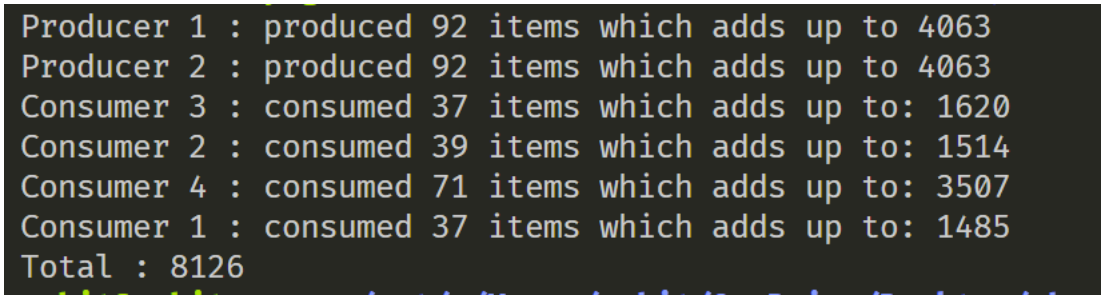
```
    if (argc != 3){
        printf("%s: Invalid Arguments\nUsage: %s <producer_count> <consumer_count>\n", argv[0], argv[0]);
        exit(EXIT_FAILURE);
    }
    int P, C;
      P = atoi(argv[1]);
      C = atoi(argv[2]);
    requiredmemory *shared = (requiredmemory*) mmap(NULL, sizeof(requiredmemory), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (shared == MAP_FAILED){
        printf("Memory map failed");
        exit(EXIT_FAILURE);
    }
    shared->count = 0;
    shared->start = 0;
    shared->total = 0;
    sem_t *full, *empty, *mutex;
    empty = &(shared->empty);
    full = &(shared->full);
    mutex = &(shared->mutex);
    sem_init(empty, 1, BUFF_LEN);
    sem_init(full, 1, 0);
    sem_init(mutex, 1, 1);
    pid_t producers[P], consumers[C];
    int p, c;
    for (c = 0; c < C; c++){
        pid_t pid = fork();
        if (pid < 0)  {
            goto __fork_error_handler_c;
        }else if (pid != 0) {
            consumers[c] = pid;
```

```
    }else {
        id = c;
        goto __consumer;
    }

}
for (p = 0; p < P; p++){
    pid_t pid = fork();
    if (pid < 0)  {
        goto __fork_error_handler_p;
    }else if (pid != 0) {
        producers[p] = pid;
    }else {
        id = p;
        goto __producer;
    }
}
while (p--)
            wait(NULL);
   while (shared->count)
            sleep(1);
   signal(SIGQUIT, SIG_IGN);
   while (c--){
            kill(consumers[c], SIGQUIT);
   }
   sleep(1);
   printf("Total : %d\n", shared->total);
   exit(EXIT_SUCCESS);

__fork_error_handler_c: {
    perror("Unexpected error occurred while forking consumers\n");
```

```
        for (--c; c >= 0; c--)
                kill(consumers[c], SIGKILL);
        exit(EXIT_FAILURE);
}
__fork_error_handler_p: {
    perror("Unexpected error occurred while forking producers\n");
    for (--c; c >= 0; c--)
                kill(consumers[c], SIGKILL);
        for (--p; p >= 0; p--)
                kill(producers[p], SIGKILL);
        exit(EXIT_FAILURE);
}
__producer: {
    srand(time(NULL));
    count = PRODUCER_MIN + rand() % PRODUCER_RANGE;
    int cnt = count;
        while (cnt--){
                unsigned char num = 1 + rand() % 80;
                sem_wait(empty);
                sem_wait(mutex);
                shared->buffer[(shared->start + (shared->count)++) % BUFF_LEN] = num;
                sem_post(mutex);
                sem_post(full);
                total += num;
        }
        printf("Producer %d : produced %d items which adds up to %d\n", id+1, count, total);
        return 0;
}
__consumer: {
    signal(SIGQUIT, consumer_handler);
        count = 0;
```

```
        while (true){
                unsigned char num = 0;
                sem_wait(full);
                sem_wait(mutex);
                num = shared->buffer[shared->start];
                shared->start = (shared->start + 1) % BUFF_LEN;
                shared->count -= 1;
                shared->total += num;
                sem_post(mutex);
                sem_post(empty);
                count++;
                total += num;
        }
        return 0;
    }
}
```

OUTPUT:



5. READER WRITER PROBLEM:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include <unistd.h>
#include <time.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/signal.h>
#define BUFF_LEN 50
#define WRITER_MIN 10
#define WRITER_RANGE 11
#define READER_MIN 10
#define READER_RANGE 11

typedef struct reqmem{
   sem_t write, mutex;
   int readercnt;
   char buffer[BUFF_LEN];
}required_memory;

int total, id, count;

int main(int argc, char** argv){
   total = 0;
   if (argc != 2){
      fprintf(stderr, "Invalid Arguments!\nUsage:%s <reader_count>\n", argv[0]);
      exit(EXIT_FAILURE);
   }
```

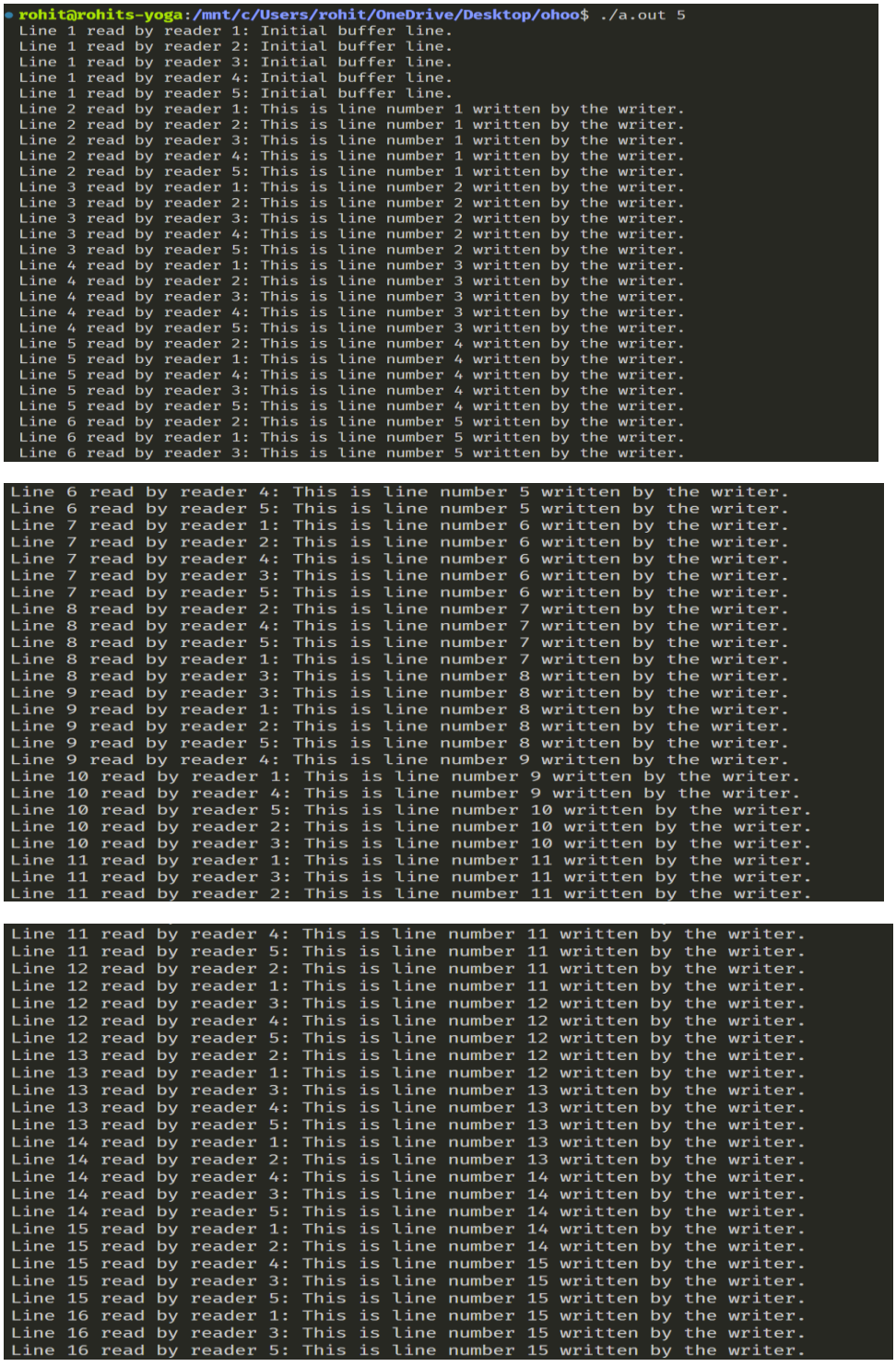
```
   int r = atoi(argv[1]);
   required_memory *shared = (required_memory*) mmap(NULL, sizeof(required_memory),
PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, -1, 0);
   if (shared == MAP_FAILED){
      fprintf(stderr, "Memory map failed\n");
      exit(EXIT_FAILURE);
   }
   strncpy(shared->buffer, "Initial buffer line.", BUFF_LEN);
   shared->readercnt = 0;
   sem_t *write, *mutex;
   write = &(shared->write);
   mutex = &(shared->mutex);
     sem_init(write, 1, 1);
     sem_init(mutex, 1, 1);
     pid_t readers[r], writer;
   int i;
   for (i = 0; i < r; i++){
      pid_t pid = fork();
      if (pid < 0){
         goto __fork_error_handler_r;
      }
      else if (pid){
         readers[i] = pid;
      }
      else{
         id = i;
         goto __reader;
      }
   }
   writer = fork();
   if(writer < 0){
```

```
    goto __fork_error_handler_w;
}else if(!writer){
    goto __writer;
}
while(i--)
    wait(NULL);
wait(NULL);
exit(EXIT_SUCCESS);

__fork_error_handler_r: {
    fprintf(stderr, "Some error while forking readers!\n");
    for(--i; i>=0; i--){
        kill(readers[i], SIGKILL);
    }
    exit(EXIT_FAILURE);
}
__fork_error_handler_w: {
    fprintf(stderr, "Some error while forking writer!\n");
    for(--i; i>=0; i--){
        kill(readers[i], SIGKILL);
    }
    exit(EXIT_FAILURE);
}
__writer: {
    srand(time(NULL));
    count = WRITER_MIN + rand() % WRITER_RANGE;
    int c = count;
    int line = 0;
    while(c--){
                line++;
                char buff[BUFF_LEN];
```

```
            sem_wait(write);
        sprintf(buff, "This is line number %d written by the writer.", line);
            strncpy(shared->buffer, buff, BUFF_LEN);
            sem_post(write);
        sleep(1);
    }
    printf("Writer wrote %d lines.\n", count);
        return 0;
}
__reader: {
    srand(time(NULL));
        int l = READER_MIN + rand() % READER_RANGE;
        while (l--){
            sem_wait(mutex);
        shared->readercnt++;
            if (shared->readercnt == 1){
           sem_wait(write);
        }
            sem_post(mutex);
        count++;
            printf("Line %d read by reader %d: %s\n", count, id+1, shared->buffer);
            sem_wait(mutex);
            shared->readercnt--;
            if (shared->readercnt == 0){
           sem_post(write);
        }
            sem_post(mutex);
        sleep(1);
        }
        printf("Reader %d read %d lines.\n", id+1, count);
        return 0;
```

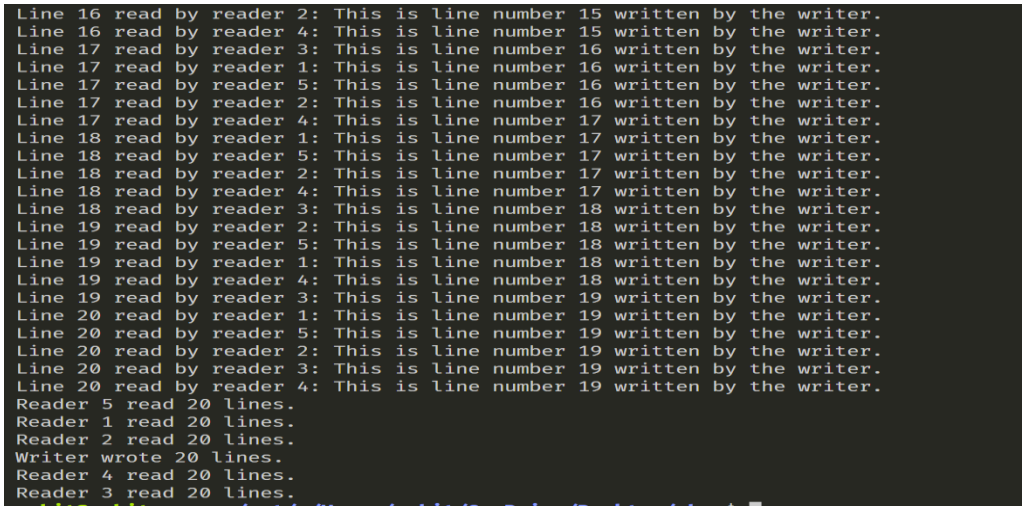
```
    }
}
```

OUTPUT:


rohit@rohits-yoga:/mnt/c/Users/rohit/OneDrive/Desktop/ohoo$ ./a.out 5
Line 1 read by reader 1: Initial buffer line.
Line 1 read by reader 2: Initial buffer line.
Line 1 read by reader 3: Initial buffer line.
Line 1 read by reader 4: Initial buffer line.
Line 1 read by reader 5: Initial buffer line.
Line 2 read by reader 1: This is line number 1 written by the writer.
Line 2 read by reader 2: This is line number 1 written by the writer.
Line 2 read by reader 3: This is line number 1 written by the writer.
Line 2 read by reader 4: This is line number 1 written by the writer.
Line 2 read by reader 5: This is line number 1 written by the writer.
Line 3 read by reader 1: This is line number 2 written by the writer.
Line 3 read by reader 2: This is line number 2 written by the writer.
Line 3 read by reader 3: This is line number 2 written by the writer.
Line 3 read by reader 4: This is line number 2 written by the writer.
Line 3 read by reader 5: This is line number 2 written by the writer.
Line 4 read by reader 1: This is line number 3 written by the writer.
Line 4 read by reader 2: This is line number 3 written by the writer.
Line 4 read by reader 3: This is line number 3 written by the writer.
Line 4 read by reader 4: This is line number 3 written by the writer.
Line 4 read by reader 5: This is line number 3 written by the writer.
Line 5 read by reader 2: This is line number 4 written by the writer.
Line 5 read by reader 1: This is line number 4 written by the writer.
Line 5 read by reader 4: This is line number 4 written by the writer.
Line 5 read by reader 3: This is line number 4 written by the writer.
Line 5 read by reader 5: This is line number 4 written by the writer.
Line 6 read by reader 2: This is line number 5 written by the writer.
Line 6 read by reader 1: This is line number 5 written by the writer.
Line 6 read by reader 3: This is line number 5 written by the writer.

Line 6 read by reader 4: This is line number 5 written by the writer.
Line 6 read by reader 5: This is line number 5 written by the writer.
Line 7 read by reader 1: This is line number 6 written by the writer.
Line 7 read by reader 2: This is line number 6 written by the writer.
Line 7 read by reader 4: This is line number 6 written by the writer.
Line 7 read by reader 3: This is line number 6 written by the writer.
Line 7 read by reader 5: This is line number 6 written by the writer.
Line 8 read by reader 2: This is line number 7 written by the writer.
Line 8 read by reader 4: This is line number 7 written by the writer.
Line 8 read by reader 5: This is line number 7 written by the writer.
Line 8 read by reader 1: This is line number 7 written by the writer.
Line 8 read by reader 3: This is line number 8 written by the writer.
Line 9 read by reader 3: This is line number 8 written by the writer.
Line 9 read by reader 1: This is line number 8 written by the writer.
Line 9 read by reader 2: This is line number 8 written by the writer.
Line 9 read by reader 5: This is line number 8 written by the writer.
Line 9 read by reader 4: This is line number 9 written by the writer.
Line 10 read by reader 1: This is line number 9 written by the writer.
Line 10 read by reader 4: This is line number 9 written by the writer.
Line 10 read by reader 5: This is line number 10 written by the writer.
Line 10 read by reader 2: This is line number 10 written by the writer.
Line 10 read by reader 3: This is line number 10 written by the writer.
Line 11 read by reader 1: This is line number 11 written by the writer.
Line 11 read by reader 3: This is line number 11 written by the writer.
Line 11 read by reader 2: This is line number 11 written by the writer.

Line 11 read by reader 4: This is line number 11 written by the writer.
Line 11 read by reader 5: This is line number 11 written by the writer.
Line 12 read by reader 2: This is line number 11 written by the writer.
Line 12 read by reader 1: This is line number 11 written by the writer.
Line 12 read by reader 3: This is line number 12 written by the writer.
Line 12 read by reader 4: This is line number 12 written by the writer.
Line 12 read by reader 5: This is line number 12 written by the writer.
Line 13 read by reader 2: This is line number 12 written by the writer.
Line 13 read by reader 1: This is line number 12 written by the writer.
Line 13 read by reader 3: This is line number 13 written by the writer.
Line 13 read by reader 4: This is line number 13 written by the writer.
Line 13 read by reader 5: This is line number 13 written by the writer.
Line 14 read by reader 1: This is line number 13 written by the writer.
Line 14 read by reader 2: This is line number 13 written by the writer.
Line 14 read by reader 4: This is line number 14 written by the writer.
Line 14 read by reader 3: This is line number 14 written by the writer.
Line 14 read by reader 5: This is line number 14 written by the writer.
Line 15 read by reader 1: This is line number 14 written by the writer.
Line 15 read by reader 2: This is line number 14 written by the writer.
Line 15 read by reader 4: This is line number 15 written by the writer.
Line 15 read by reader 3: This is line number 15 written by the writer.
Line 15 read by reader 5: This is line number 15 written by the writer.
Line 16 read by reader 1: This is line number 15 written by the writer.
Line 16 read by reader 3: This is line number 15 written by the writer.
Line 16 read by reader 5: This is line number 15 written by the writer.



b)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include <unistd.h>
#include <time.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/signal.h>
#define BUFF_LEN 50
#define WRITER_MIN 10
#define WRITER_RANGE 11
#define READER_MIN 10
#define READER_RANGE 11
typedef struct reqmem{
   sem_t rentry, wentry, write, mut1, mut2;
   int readcnt, writecnt;
   char buffer[BUFF_LEN];
}required_memory;
```

```
int total, count, id;
int main(int argc, char** argv){
   total = 0;
   if (argc != 3){
      fprintf(stderr, "Invalid Arguments!\nUsage:%s <reader_count> <writer_count>\n", argv[0]);
      exit(EXIT_FAILURE);
   }
   int R, W;
   R = atoi(argv[1]);
   W = atoi(argv[2]);
   required_memory *shared = (required_memory*) mmap(NULL, sizeof(required_memory),
PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
          strncpy(shared->buffer, "Initial buffer line", BUFF_LEN);
          shared->readcnt = 0;
          shared->writecnt = 0;
          sem_t *rentry, *wentry, *mut1, *mut2, *wrt;
          rentry = &(shared->rentry);
          wentry = &(shared->wentry);
          wrt = &(shared->write);
          mut1 = &(shared->mut1);
          mut2 = &(shared->mut2);
          sem_init(rentry, 1, 1);
          // sem_init(wentry, 1, 1);
          sem_init(wrt, 1, 1);
          sem_init(mut1, 1, 1);
          sem_init(mut2, 1, 1);
          pid_t readers[R], writers[W];
          int r, w;
          for (r = 0; r < R; r++){
                pid_t pid = fork();
                if (pid < 0)
```

```
                goto __fork_error_handler_r;
        else if (pid){
                readers[r] = pid;
        }
        else{
                id = r;
                goto __reader;
        }
    }
    for (w = 0; w < W; w++)  {
        pid_t pid = fork();
        if (pid < 0)
                goto __fork_error_handler_w;
        else if (pid){
                writers[w] = pid;
        }
        else{
                id = w;
                goto __writer;
        }
    }
    while (r--)
        wait(NULL);
    while (w--)
        wait(NULL);
    exit(EXIT_SUCCESS);

__fork_error_handler_r:{
    fprintf(stderr, "Some error forking readers\n");
    for (--r; r >= 0; r--)
        kill(readers[r], SIGKILL);
```

```
            exit(EXIT_FAILURE);
}
__fork_error_handler_w:{
            fprintf(stderr, "Some error forking writers\n");
            for (--r; r >= 0; r--)
                        kill(readers[r], SIGKILL);
            for (--w; w >= 0; w--)
                        kill(writers[w], SIGKILL);
            exit(EXIT_FAILURE);
}
__writer:{
            srand(time(NULL));
            count = WRITER_MIN + rand() % WRITER_RANGE;
            int cnt = count;
            int l = 0;
            while (cnt--){
                        l++;
                        char buff[BUFF_LEN];
                        sprintf(buff, "Writer: %d Line: %d", id+1, l);
      // prevents other writers if reader already present or if reader
      // not present then blocks reader.
                        sem_wait(mut2);
                        shared->writecnt++;
                        if (shared->writecnt == 1)
                                    sem_wait(rentry);
                        sem_post(mut2);

      // prevents simultaneous writing
                        sem_wait(wrt);
                        strncpy(shared->buffer, buff, BUFF_LEN);
                        sem_post(wrt);
```

```
                sem_wait(mut2);
                shared->writecnt--;
                if (shared->writecnt == 0)
                        sem_post(rentry);
                sem_post(mut2);

        sleep(1);
            }
            printf("Writer %d wrote %d lines finished at %ld\n", id+1, count, clock());
            return 0;
}
__reader:{
            srand(time(NULL));
            int l = READER_MIN + rand() % READER_RANGE;
            while (l--)      {
                        count++;
        // if writer already present then prevents entrance
                        sem_wait(rentry);

        // prevents other readers if writer already present or if writer
        // not present then blocks writer.
                        sem_wait(mut1);
                        if (!shared->readcnt)
                                sem_wait(wrt);     // wrt --> blocking writers from entering
                        sem_post(mut1);

                        sem_post(rentry);

        // read can be done by multiple readers so, signalling rentry
        // before actual operation
```

```
                shared->readcnt++;
                printf("Line: %d read by Reader: %d: '%s'\n", count, id+1, shared->buffer);

                sem_wait(mut1);
                shared->readcnt--;
                if (!shared->readcnt)
                        sem_post(wrt);
                sem_post(mut1);

        sleep(1);
            }
            printf("Reader: %d read %d lines finished at %ld\n", id+1, count, clock());
            return 0;
    }
}
```

OUTPUT:

```
rohit@rohits-yoga:/mnt/c/Users/rohit/OneDrive/Desktop/ohoo$ ./a.out 3 2
 Line: 1 read by Reader: 1: 'Initial buffer line'
 Line: 1 read by Reader: 2: 'Initial buffer line'
 Line: 1 read by Reader: 3: 'Initial buffer line'
 Line: 2 read by Reader: 1: 'Writer: 1 Line: 1'
 Line: 2 read by Reader: 2: 'Writer: 1 Line: 1'
 Line: 2 read by Reader: 3: 'Writer: 1 Line: 1'
 Line: 3 read by Reader: 2: 'Writer: 1 Line: 2'
 Line: 3 read by Reader: 3: 'Writer: 1 Line: 2'
 Line: 3 read by Reader: 1: 'Writer: 1 Line: 3'
 Line: 4 read by Reader: 2: 'Writer: 1 Line: 3'
 Line: 4 read by Reader: 3: 'Writer: 2 Line: 4'
 Line: 4 read by Reader: 1: 'Writer: 1 Line: 4'
 Line: 5 read by Reader: 2: 'Writer: 1 Line: 4'
 Line: 5 read by Reader: 3: 'Writer: 2 Line: 5'
 Line: 5 read by Reader: 1: 'Writer: 1 Line: 5'
 Line: 6 read by Reader: 2: 'Writer: 1 Line: 5'
 Line: 6 read by Reader: 3: 'Writer: 2 Line: 6'
 Line: 6 read by Reader: 1: 'Writer: 1 Line: 6'
 Line: 7 read by Reader: 2: 'Writer: 1 Line: 6'
 Line: 7 read by Reader: 3: 'Writer: 1 Line: 6'
 Line: 7 read by Reader: 1: 'Writer: 1 Line: 7'
 Line: 8 read by Reader: 2: 'Writer: 1 Line: 7'
 Line: 8 read by Reader: 3: 'Writer: 1 Line: 7'
 Line: 8 read by Reader: 1: 'Writer: 2 Line: 8'
 Line: 9 read by Reader: 2: 'Writer: 1 Line: 8'
 Line: 9 read by Reader: 3: 'Writer: 1 Line: 8'
 Line: 9 read by Reader: 1: 'Writer: 1 Line: 9'
 Line: 10 read by Reader: 3: 'Writer: 1 Line: 9'
```

```
Line: 10 read by Reader: 1: 'Writer: 1 Line: 10'
Line: 11 read by Reader: 3: 'Writer: 2 Line: 11'
Line: 11 read by Reader: 2: 'Writer: 1 Line: 11'
Line: 11 read by Reader: 1: 'Writer: 1 Line: 11'
Line: 12 read by Reader: 3: 'Writer: 1 Line: 12'
Line: 12 read by Reader: 1: 'Writer: 1 Line: 12'
Line: 12 read by Reader: 2: 'Writer: 1 Line: 12'
Writer 1 wrote 12 lines finished at 1526
Writer 2 wrote 12 lines finished at 1344
Reader: 3 read 12 lines finished at 2231
Reader: 1 read 12 lines finished at 1681
Reader: 2 read 12 lines finished at 2146
```

6. DINING PHILOSOPHERS PROBLEM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <fcntl.h>
#include <semaphore.h>
#include <time.h>
#include <sys/mman.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int N;
enum Status
{
   THINKING,
   HUNGRY,
   EATING
};
enum Status *state;
sem_t *mutex;
sem_t **condition;
char *mutex_key_from_no(int n)
{
   char *buffer;
   buffer = (char *)malloc(sizeof(char) * 50);
   sprintf(buffer, "mutex%d", n);
```

```
   return buffer;

}

// test left and right void test(int i){ if (state[i] == HUNGRY && state[(i + 1) % N] != EATING &&
state[(i+N-1) % N] !=

EATING)      {

// update state to eating sem_wait(mutex); state[i] = EATING; sem_post(mutex); sleep(2);

// eating printf("Philosopher %d takes fork %d and %d\n", i+1, (i+N-1)%N+1, i+1);
printf("Philosopher %d is Eating\n", i+1); sem_post(condition[i]); // signal }

}

// pickup chopsticks void pickup(int i)

{

   // update state to hungry sem_wait(mutex); state[i] = HUNGRY; printf("Philosopher %d is
Hungry\n", i+1); sem_post(mutex);

   // eat if neighbours are not eating test(i);

   // if unable to eat wait to be signalled sem_wait(condition[i]); sleep(1);

}

// put down chopsticks void putdown(int i) {

// update state to thinking sem_wait(mutex); state[i] = THINKING; sem_post(mutex);
printf("Philosopher %d putting fork %d and %d down\n", i+1, (i+N-1)%N+1, i+1);
printf("Philosopher %d is thinking\n", i+1);

test((i + N - 1) % N);

test((i + 1) % N);

}

// start philosopher action void startPhilosopherAction(int num){ while (1)

{

   sleep(1);

   pickup(num);

   sleep(0);

   putdown(num);

}

}

int main()

{
```

```
   printf("Enter value of N : ");
   scanf("%d", &N);
   // allocate memory state = mmap(NULL, sizeof(enum Status) * N, PROT_READ | PROT_WRITE,
MAP_SHARED
| MAP_ANONYMOUS, -1, 0);
condition = mmap(NULL, sizeof(sem_t *) * N, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
for (int i = 0; i < N; i++)
{
   state[i] = THINKING;
}
// initializemutex semaphore mutex = sem_open("mutex56", O_CREAT, 0777, 1);
// initialize condition semaphore for (int i = 0; i < N; i++){ condition[i] =
sem_open(mutex_key_from_no(i), O_CREAT, 0777, 1);
}
int id = fork();
if (id == 0)
{ // Fork childs for (int i = 0; i < N; i++){ id = fork(); if (id == 0){ startPhilosopherAction(i); exit(1);
}
}
for (int i = 0; i < N; i++)
{
   wait(NULL);
}
exit(1);
}
else
{
   wait(NULL);
}
return 0;
}
```

OUTPUT:

Enter value of N : 5
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 2 putting fork 1 and 2
down Philosopher 2 is thinking
Philosopher 5 putting fork 4 and 5
down Philosopher 5 is thinking
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 1 putting fork 5 and 1
down Philosopher 1 is thinking
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 5 takes fork 4
and 5 Philosopher 5 is Eating
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 3 is Hungry
Philosopher 5 is thinking
Philosopher 2 putting fork 1 and 2
down Philosopher 2 is thinking
Philosopher 1 putting fork 5 and 1
down Philosopher 1 is thinking
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 4 takes fork 3
and 4 Philosopher 1 is
Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 4 is Eating
Philosopher 1 is Eating Philosopher 3 is Hungry
…

7. BANKER'S ALGORITHM

```
import java.io.*; import java.util.Arrays;

class BankersUtil{ int no_of_process; int no_of_resources; int safe_sequence[]; int
available_resources[]; int allocated_resources[][]; int max_resources_request[][]; int
need_resources[][];

BankersUtil(int no_of_process, int no_of_resources){ this.no_of_process = no_of_process;
this.no_of_resources = no_of_resources; this.safe_sequence = new int[no_of_process];
this.available_resources = new int[no_of_resources]; this.allocated_resources = new
int[no_of_process][no_of_resources]; this.max_resources_request = new
int[no_of_process][no_of_resources]; this.need_resources = new
int[no_of_process][no_of_resources];

}

public void setNoOfInstanceForResource(int resource_no, int no_of_instances){
this.available_resources[resource_no] = no_of_instances;

}

public void setMaxResourceForProcess(int process_no, int resource_no, int

no_of_instances){

this.max_resources_request[process_no][resource_no] = no_of_instances;

}

public void calculateNeedMatrix(){ for(int i=0;i<no_of_process;i++){ for(int
j=0;j<no_of_resources;j++){ need_resources[i][j] = max_resources_request[i][j] -

allocated_resources[i][j];

}

} }

public void displayData(){

System.out.println("Process No\tAllocated\tMax\t\tNeed\t\tAvailable"); System.out.println("-----
-----\t---------\t--------\t--------\t---------"); for(int i=0;i<no_of_process;i++){
System.out.print("P"+i+"\t\t"); for(int j=0;j<no_of_resources;j++){

System.out.print(allocated_resources[i][j]+" "); }

System.out.print("\t");

for(int j=0;j<no_of_resources;j++){

System.out.print(max_resources_request[i][j]+" "); }

System.out.print("\t");

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

```
System.out.print(need_resources[i][j]+" "); }

System.out.print("\t"); if(i==0){ for(int j=0;j<no_of_resources;j++){

System.out.print(available_resources[j]+" "); }

}

System.out.println(); }

System.out.println();

}

public void displaySafeSequence(){ System.out.print("Safe Sequence: "); for(int
i=0;i<no_of_process;i++){

System.out.print("P"+safe_sequence[i]+" "); }

System.out.println(); }

public void submitRequest(int process_no, int[] request){ for(int i=0;i<no_of_resources;i++){
if(request[i] > need_resources[process_no][i]){

System.out.println("Request cannot be granted as it exceeds the need

of process P"+process_no); return;

}

if(request[i] > available_resources[i]){

System.out.println("Request cannot be granted as it exceeds the

available resources"); return;

}

}

// create copy of available resources, allocated resources and need resources int[]
available_resources_copy = new int[no_of_resources]; int[][] allocated_resources_copy = new
int[no_of_process][no_of_resources]; int[][] need_resources_copy = new
int[no_of_process][no_of_resources];

// copy data for(int i=0;i<no_of_resources;i++){ available_resources_copy[i] =
available_resources[i];

} for(int i=0;i<no_of_process;i++){ for(int j=0;j<no_of_resources;j++){
allocated_resources_copy[i][j] = allocated_resources[i][j]; need_resources_copy[i][j] =
need_resources[i][j]; }

}

// grant request for(int i=0;i<no_of_resources;i++){ available_resources_copy[i] -= request[i];
allocated_resources_copy[process_no][i] += request[i]; need_resources_copy[process_no][i] -=
request[i];
```

```
}

// create variables boolean[] finished = new boolean[no_of_process]; int[] work =
Arrays.copyOf(available_resources_copy, available_resources_copy.length); int[]
safe_sequence_copy = new int[no_of_process];

// calculate safe sequence

int index = 0; boolean flag = false;

while(true){ for(int i=0;i<no_of_process;i++){ if(finished[i] == false){ boolean canfinish = true;
for(int j=0;j<no_of_resources;j++){ if(need_resources_copy[i][j] > work[j]){ canfinish = false;
break;

} }

if(canfinish){ for(int j=0;j<no_of_resources;j++){ work[j] += allocated_resources_copy[i][j];

} finished[i] = true; safe_sequence_copy[index++] = i; flag = true;

}

} } if(!flag) break; flag = false;

}

// check if safe sequence is valid for(int i=0;i<no_of_process;i++){ if(finished[i] == false){

System.out.println("Request cannot be granted as it will lead to

unsafe state"); return;

}

}

System.out.println("Request can be granted as it will lead to safe state"); System.out.print("Safe
Sequence: "); for(int i=0;i<no_of_process;i++){

System.out.print("P"+safe_sequence_copy[i]+" "); }

System.out.println();

// update data

for(int i=0;i<no_of_resources;i++){ available_resources[i] -= request[i];
allocated_resources[process_no][i] += request[i]; need_resources[process_no][i] -= request[i];

}

// check if need of process becomes zero boolean need_becomes_zero = true; for(int
i=0;i<no_of_resources;i++){ if(need_resources[process_no][i] != 0){ need_becomes_zero = false;
break;

}

}

// if need of process becomes zero, add process to safe sequence if(need_becomes_zero){
```

```
System.out.println("Need of process P"+process_no+" becomes zero, so it

has finished execution"); // free resources for(int i=0;i<no_of_resources;i++){
available_resources[i] += allocated_resources[process_no][i]; allocated_resources[process_no][i]
= 0; }

}

} } public class Question { static int safe_sequence[];

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new FileReader("input.txt"));

int nr = Integer.parseInt(br.readLine()); int np = Integer.parseInt(br.readLine());

BankersUtil banker = new BankersUtil(np, nr);

// Read no of instances for each resource String[] resources = br.readLine().split(" "); for(int
i=0;i<nr;i++){ banker.setNoOfInstanceForResource(i, Integer.parseInt(resources[i])); }

// Read max need for each process for(int i=0;i<np;i++){

String[] _tmp = br.readLine().split(" "); for(int j=0;j<nr;j++){ banker.setMaxResourceForProcess(i, j,
Integer.parseInt(_tmp[j]));

} }

banker.calculateNeedMatrix(); banker.displayData(); while(true){

System.out.println("Enter process number and request for each resource");

BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in)); String[] _tmp =
br1.readLine().split(" "); int process_no = Integer.parseInt(_tmp[0]); int[] request = new int[nr];
for(int i=0;i<nr;i++){ request[i] = Integer.parseInt(_tmp[i+1]);

} banker.submitRequest(process_no, request); banker.displayData();

}

}

}
```

OUTPUT:

Output -

```
Process No     Allocated      Max          Need         Available
----------     ---------     --------      --------      ---------
P0          0 0 0 0      1 1 1 1      1 1 1 1      2 4 5 3
P1          0 0 0 0      2 3 1 2      2 3 1 2
P2          0 0 0 0      2 2 1 3      2 2 1 3

Enter process number and request for each resource
0 1 0 1 1
```

Request can be granted as it will lead to safe state
Safe Sequence: P0 P1 P2

Process No	Allocated	Max	Need	Available
P0	1 0 1 1	1 1 1 1	0 1 0 0	1 4 4 2
P1	0 0 0 0	2 3 1 2	2 3 1 2	
P2	0 0 0 0	2 2 1 3	2 2 1 3	

Enter process number and request for each resource
1 2 0 2 2
Request cannot be granted as it exceeds the available resources

Process No	Allocated	Max	Need	Available
P0	1 0 1 1	1 1 1 1	0 1 0 0	1 4 4 2
P1	0 0 0 0	2 3 1 2	2 3 1 2	
P2	0 0 0 0	2 2 1 3	2 2 1 3	

Enter process number and request for each resource
1 1 0 1 1
Request can be granted as it will lead to safe state
Safe Sequence: P0 P1 P2

Process No	Allocated	Max	Need	Available
P0	1 0 1 1	1 1 1 1	0 1 0 0	0 4 3 1
P1	1 0 1 1	2 3 1 2	1 3 0 1	
P2	0 0 0 0	2 2 1 3	2 2 1 3	

Enter process number and request for each resource
2 0 2 1 1
Request cannot be granted as it will lead to unsafe state

Process No	Allocated	Max	Need	Available
P0	1 0 1 1	1 1 1 1	0 1 0 0	0 4 3 1
P1	1 0 1 1	2 3 1 2	1 3 0 1	
P2	0 0 0 0	2 2 1 3	2 2 1 3	