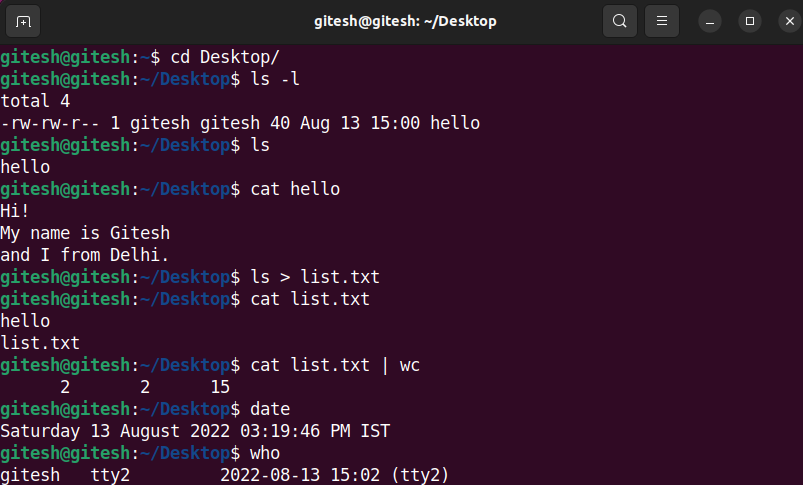
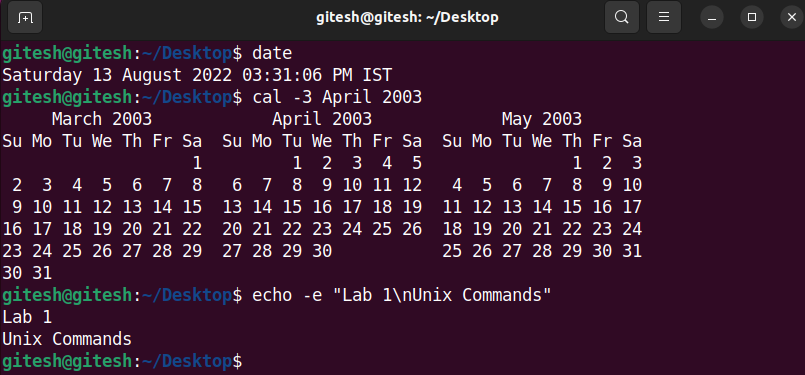
OS Lab File

Lab Assignment 1-2

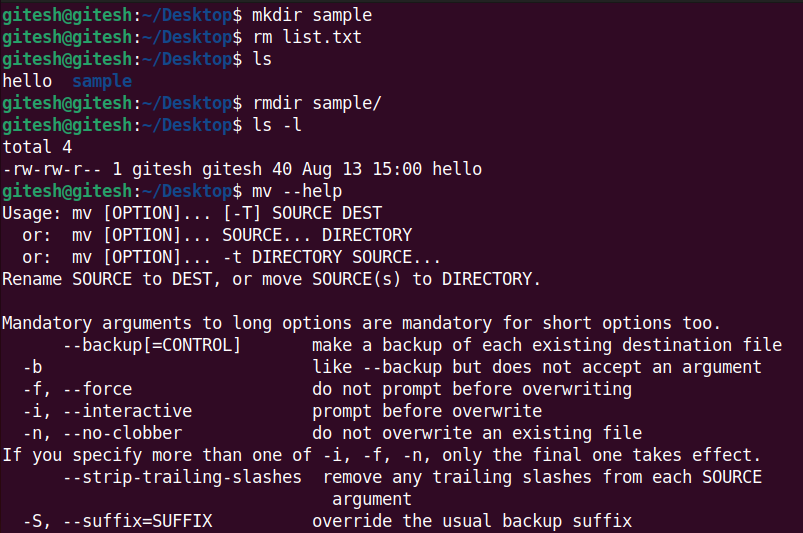
* cd – change directory
* wc – word count
* ls – list all the files/directories
* clear – used to clear the terminal
* exit – used to exit
* cat – displays the contents of the file



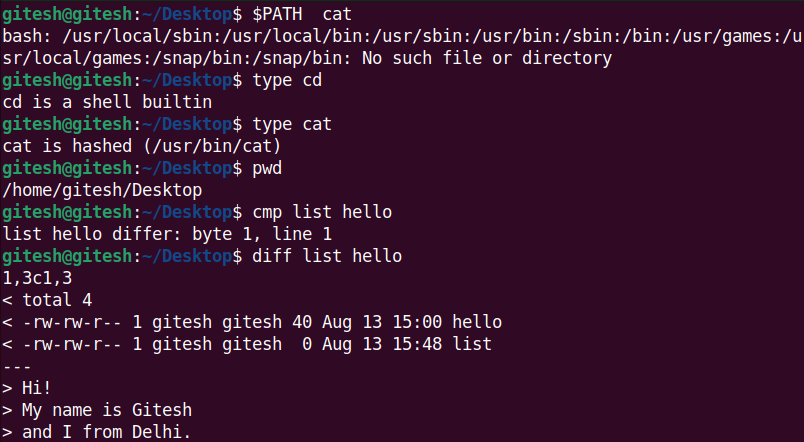
* cal & date – calendar and date & time commands respectively
* echo (-e) – display the input (special characters)
* rm – remove files
* mkdir/ rmdir – make/ remove directory



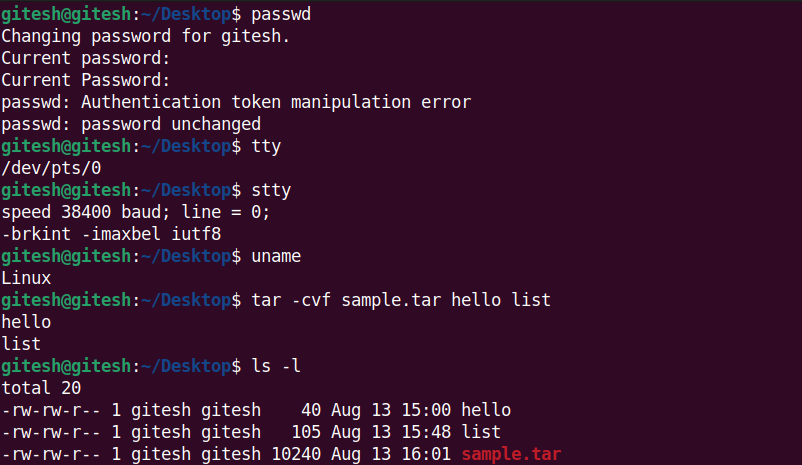
* mv & cp – move & copy files
* help & man – help & manual of commands



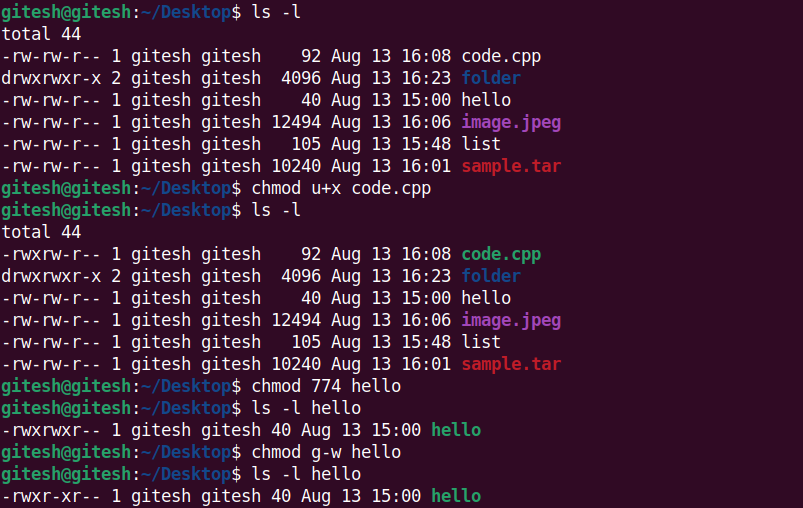
* pwd – present working directory
* cmp/comm/diff – compares, differentiates two files
* type – type of command (Internal or External)



* passwd – change password
* uname – tells username
* tty & stty – displays the location of the terminal and tells its characteristics
* who – list of users
* tar– compress/ extract files in a zip folder (-c – compress, -x – extract, -v – verbose, -f – filename)



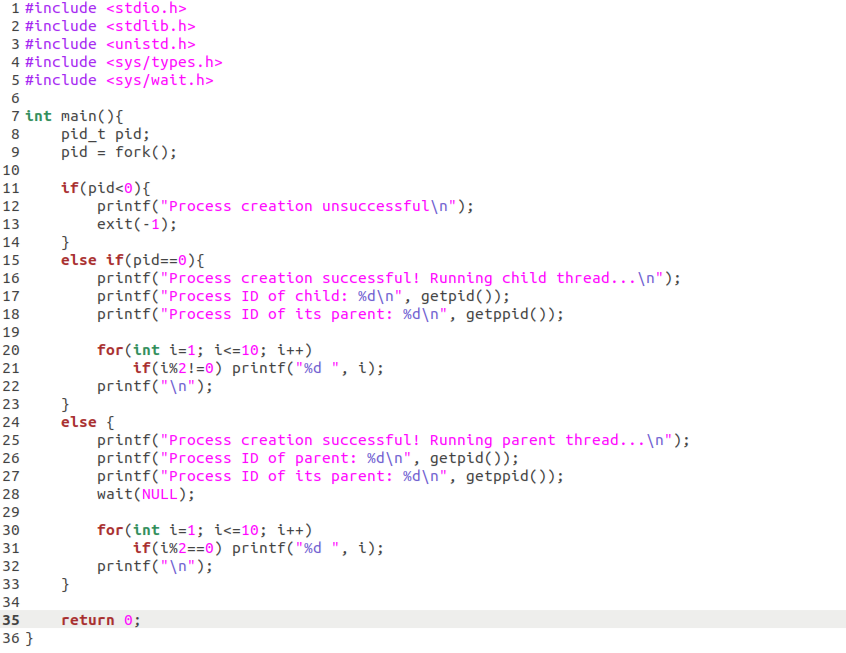
* $PATH – path to the locations where the specified command in present
* $HOME – absolute path to the root directory
* touch – create one/multiple files
* chmod – used to access permissions and **change mode** of a file or directory
  + u/g/o – user, group, others
  + +/- add/remove permission
  + r/w/x – read, write, execute

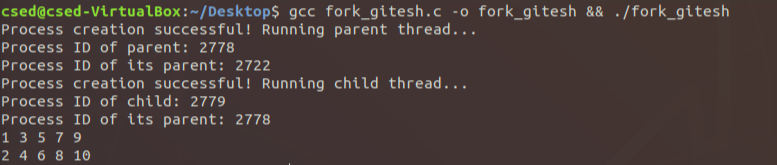


Lab Assignment 3

## System Calls

1. fork() – used to create a new child process
2. getpid() – get the process ID of the current thread
3. getppid() – get the process ID of the parent thread
4. wait(NULL) – wait until the execution of child process isn’t over (get rid of orphan child processes i.e., processes with already executed parent thread)





Lab Assignment 4

## CPU Scheduling (FCFS):

#include <stdio.h>

void swap(int\* a, int\* b){

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void sort(int p[], int at[], int bt[], int size){

    for(int i=0; i<size; i++){

        for(int j=0; j<size-i-1; j++){

            if(at[j]>at[j+1]){

                swap(&at[j], &at[j+1]);

                swap(&bt[j], &bt[j+1]);

                swap(&p[j], &p[j+1]);

            }

        }

    }

}

double sum(int arr[], int size){

    double sum=0;

    for(int k=0; k<size; k++)

        sum+=arr[k];

    return sum;

}

int main(){

    int N;

    printf("Enter no of processes: ");

    scanf("%d", &N);

    //input

    int at[N], bt[N], p[N], wait[N], tat[N], ct[N], idle\_time=0;

    for(int i=0; i<N; i++){

        printf("Enter arrival and burst time for process %d\n", i+1);

        scanf("%d\n%d", &at[i], &bt[i]);

        p[i] = i+1;

    }

    sort(p, at, bt, N);

    //fcfs

    wait[0] = 0;

    ct[0] = at[0] + bt[0];

    for(int i=0; i<N; i++){

        idle\_time=0;

        wait[i] = ct[i] - bt[i] - at[i];

        if(at[i+1] > ct[i]) idle\_time += at[i+1] - ct[i];

        ct[i+1] = ct[i] + bt[i+1] + idle\_time;

        tat[i] = wait[i] + bt[i];

    }

    //display

    printf("\nProcess\tArrival\tBurst\tWait\tTAT\n");

    for(int i=0; i<N; i++)

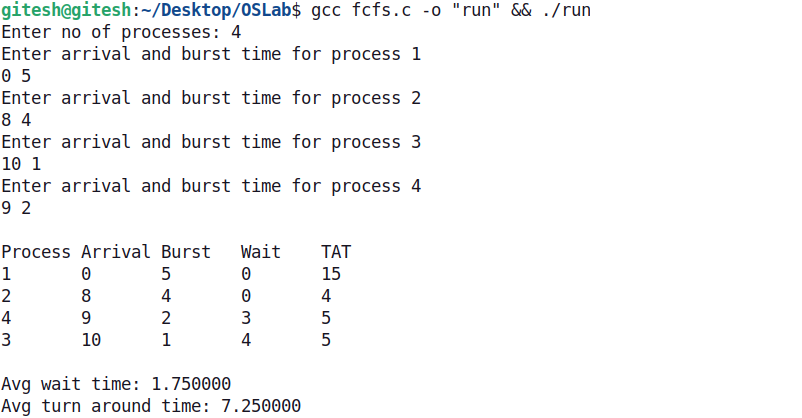
        printf("%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], wait[i], tat[i]);

    printf("\nAvg wait time: %f", sum(wait, N)/N);

    printf("\nAvg turn around time: %f\n", sum(tat, N)/N);

}

## Output:



## SRTF Scheduling algorithm:

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

struct process

{

    int pid;

    int at;

    int st; // start time

    int bt;

    int rt;

    int ct;

    int wt;

    int tat;

};

int main()

{

    int N;

    printf("Enter no of processes: ");

    scanf("%d", &N);

    struct process P[N];

    int curr\_time = 0, idle\_time = 0, completed = 0;

    int is\_completed[100] = {false};

    for (int i = 0; i < N; i++){

        printf("Enter arrival time and burst time for process %d:\n", i+1);

        scanf("%d %d", &P[i].at, &P[i].bt);

        P[i].rt = P[i].bt;

        P[i].pid = i + 1;

    }

    while (completed != N){

        int min\_proc = -1; // returns the index for the proc w min bt

        int min\_bt = INT\_MAX; // largest value int can hold

        for(int i = 0; i < N; i++){

            if (P[i].at <= curr\_time && is\_completed[i] == false){

                if (P[i].rt < min\_bt){

                    min\_bt = P[i].rt;

                    min\_proc = i;

                }

                if (P[i].rt == min\_bt){

                    if (P[i].at < P[min\_proc].at){

                        min\_bt = P[i].rt;

                        min\_proc = i;

                    }

                }

            }

        }

        if (min\_proc == -1){

            // no proc found w min CPU bt in ready queue till curr\_time

            curr\_time++;

            idle\_time++;

        }

        else{

            if (P[min\_proc].rt == P[min\_proc].bt){

                P[min\_proc].st = curr\_time;

            }

            P[min\_proc].rt--;

            curr\_time++;

            if (P[min\_proc].rt == 0){

                P[min\_proc].ct = curr\_time;

                P[min\_proc].tat = P[min\_proc].ct - P[min\_proc].at;

                P[min\_proc].wt = P[min\_proc].tat - P[min\_proc].bt;

                completed++;

                is\_completed[min\_proc] = true;

            }

        }

    }

    printf("\nPId\tArr\tBT\tWait\tTAT\n");

    for (int i = 0; i < N; i++)

        printf("%d\t%d\t%d\t%d\t%d\n", P[i].pid, P[i].at, P[i].bt, P[i].wt, P[i].tat);

    int sum\_wt = 0, sum\_tat = 0;

    for (int i = 0; i < N; i++){

        sum\_wt += P[i].wt;

        sum\_tat += P[i].tat;

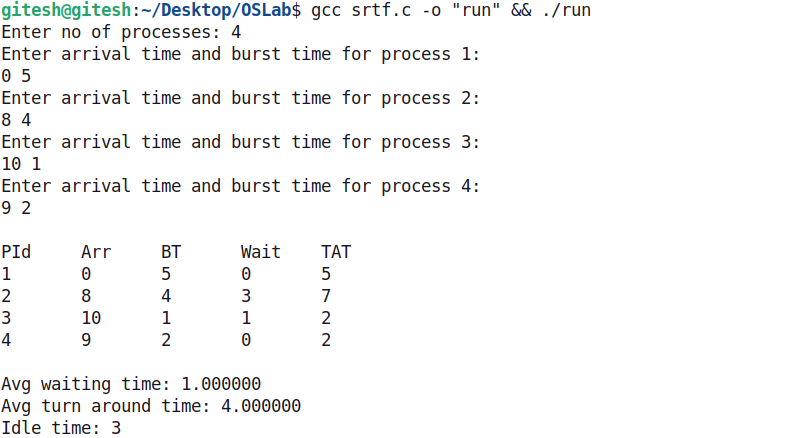
    }

    printf("\nAvg waiting time: %f", (float)sum\_wt / N);

    printf("\nAvg turn around time: %f", (float)sum\_tat / N);

    printf("\nIdle time: %d\n", idle\_time);

## Output:



Lab Assignment 5

## Banker’s Algorithm:

#include <stdio.h>

#include <stdbool.h>

int main(){

    bool flag = true, end = false;

    int P, R, k = 0;

    printf("Enter no of processes, resources: ");

    scanf("%d%d", &P, &R);

    int ss[P], av[R], max[P][R], alloc[P][R], need[P][R];

    bool finished[100] = { false };

    printf("Enter no of resources available: ");

    for (int k = 0; k < R; k++) {

        scanf("%d", &av[k]);

    }

    for (int i = 0; i < P; i++) {

        printf("Enter max no of resources reqd by proc %d: ", i+1);

        for (int j = 0; j < R; j++) {

            scanf("%d", &max[i][j]);

        }

        printf("Enter no of resources allocated to proc %d: ", i+1);

        for (int j = 0; j < R; j++) {

            scanf("%d", &alloc[i][j]);

            need[i][j] = max[i][j] - alloc[i][j];

        }

    }

    while (!end) {

        for (int i = 0; i < P; i++) {

            if (av[0] >= \*need[i] && finished[i] == false) {

                for (int j = 1; j < R; j++) {

                    if (need[i][j] > av[j]) {

                        flag = false;

                    }

                }

                if (flag) {

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

                        av[j] += alloc[i][j];

                    }

                    ss[k] = i+1;

                    k++;

                    finished[i] = true;

                }

                end = true;

                for (int i = 0; i < P; i++) {

                    if (finished[i] == false) {

                        end = false;

                    }

                }

            }

        }

    }

    printf("\nSafety sequence: < ");

    for (int i = 0; i < P; i++) {

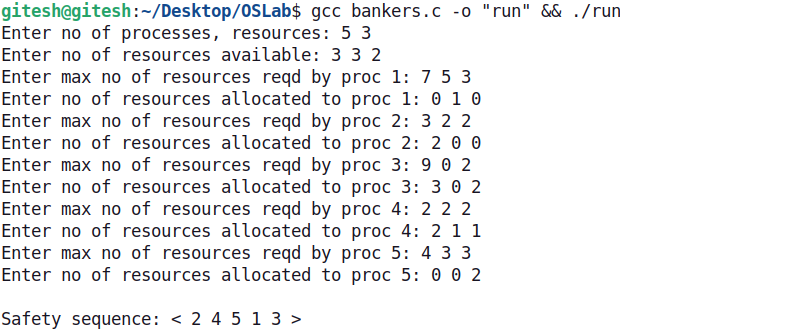
        printf("%d ", ss[i]);

    }

    printf(">\n");

}

## Output:



Lab Assignment 8

## Disk Scheduling (FCFS):

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

int seek\_time(int arr[], int head, int len) {

    int seek\_time = abs(arr[0] - head);

    for (int i = 1; i < len; i++)

        seek\_time += abs(arr[i] - arr[i-1]);

    return seek\_time;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq];

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++)

        scanf("%d", &requests[i]);

    printf("Enter position of head: ");

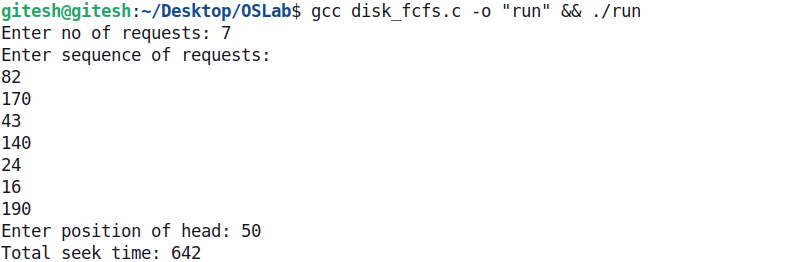
    scanf("%d", &head);

    int ans = seek\_time(requests, head, noOfReq);

    printf("Total seek time: %d\n", ans);

}

## Output:



## SSTF:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

int min\_index(int arr[], int len) {

    int min = arr[0], index = 0;

    for (int i = 1; i < len; i++) {

        if (arr[i] <= min) {

            min = arr[i];

            index = i;

        }

    }

    return index;

}

int seek\_time(int arr[], int head, int len) {

    int seek\_time = 0, req = len;

    while (req--) {

        int closest[len];

        for (int i = 0; i < len; i++) {

            closest[i] = abs(arr[i] - head);

        }

        seek\_time += abs(head - arr[min\_index(closest, len)]);

        head = arr[min\_index(closest, len)];

        arr[min\_index(closest, len)] = INT\_MAX;

    }

    return seek\_time;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq];

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++)

        scanf("%d", &requests[i]);

    printf("Enter position of head: ");

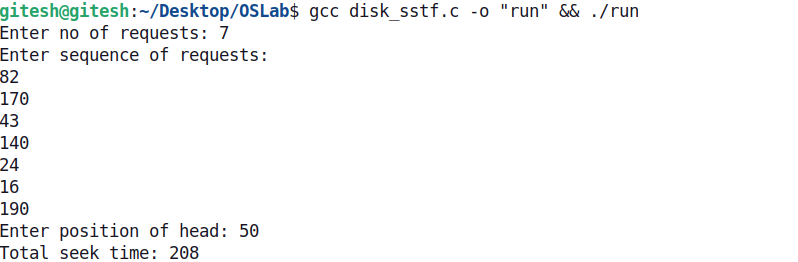
    scanf("%d", &head);

    int ans = seek\_time(requests, head, noOfReq);

    printf("Total seek time: %d\n", ans);

}

## Output:



## SCAN:

#include <stdio.h>

#include <stdlib.h>

int min(int arr[], int len) {

    int min = arr[0];

    for (int i = 1; i < len; i++) {

        if (arr[i] <= min) {

            min = arr[i];

        }

    }

    return min;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq], max\_req = 0;

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++) {

        scanf("%d", &requests[i]);

        max\_req = (requests[i] >= max\_req) ? requests[i] : max\_req;

    }

    printf("Enter position of head: ");

    scanf("%d", &head);

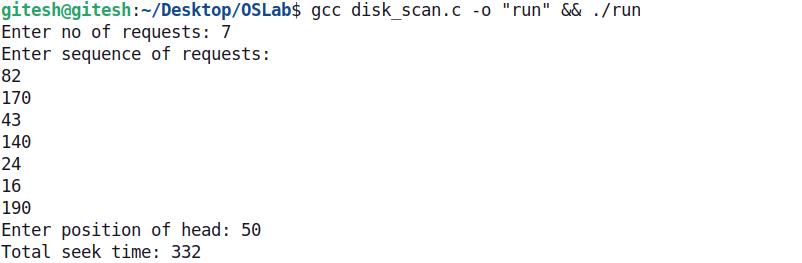
    int range\_min = 0, range\_max = max\_req + (10-1);

    int ans = abs(range\_max - head) + (range\_max - min(requests, noOfReq));

    printf("Total seek time: %d\n", ans);

}

## Output:



## CSCAN:

#include <stdio.h>

#include <stdlib.h>

void sort(int arr[], int len) {

    for (int i = 0; i < len; i++) {

        for (int j = 0; j < len-i-1; j++) {

            if (arr[j] > arr[j+1]) {

                int temp = arr[j+1];

                arr[j+1] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

int min(int arr[], int len, int head) {

    sort(arr, len);

    int val = arr[0];

    for (int i = 0; i < len; i++) {

        if (arr[i] <= head) {

            val = arr[i];

        } else {

            break;

        }

    }

    return val;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq], max\_req = 0;

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++) {

        scanf("%d", &requests[i]);

        max\_req = (requests[i] >= max\_req) ? requests[i] : max\_req;

    }

    printf("Enter position of head: ");

    scanf("%d", &head);

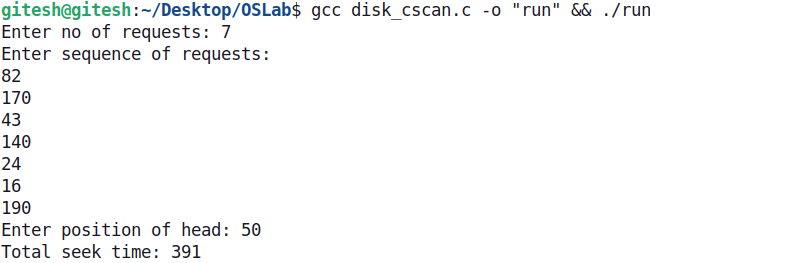
    int range\_min = 0, range\_max = max\_req + (10-1);

    int ans = abs(range\_max - head) + (range\_max - range\_min) + (min(requests, noOfReq, head) - range\_min);

    printf("Total seek time: %d\n", ans);

}

## Output:



## LOOK:

#include <stdio.h>

#include <stdlib.h>

int min(int arr[], int len) {

    int min = arr[0], index = 0;

    for (int i = 1; i < len; i++) {

        if (arr[i] <= min) {

            min = arr[i];

        }

    }

    return min;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq], max\_req = 0;

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++) {

        scanf("%d", &requests[i]);

        max\_req = (requests[i] >= max\_req) ? requests[i] : max\_req;

    }

    printf("Enter position of head: ");

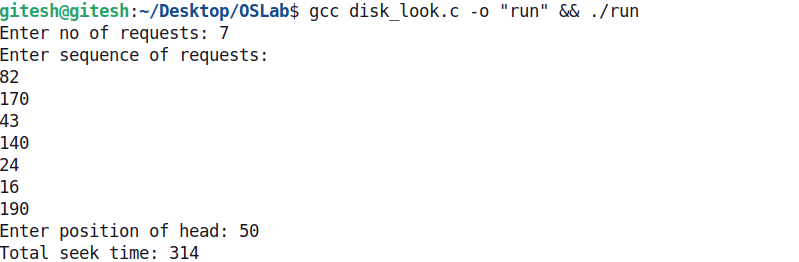
    scanf("%d", &head);

    int ans = abs(max\_req - head) + (max\_req - min(requests, noOfReq));

    printf("Total seek time: %d\n", ans);

}

## Output:



## CLOOK:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

void sort(int arr[], int len) {

    for (int i = 0; i < len; i++) {

        for (int j = 0; j < len-i-1; j++) {

            if (arr[j] > arr[j+1]) {

                int temp = arr[j+1];

                arr[j+1] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

int min(int arr[], int len, int head) {

    sort(arr, len);

    int val = arr[0];

    for (int i = 0; i < len; i++) {

        if (arr[i] <= head) {

            val = arr[i];

        } else {

            break;

        }

    }

    return val;

}

int main() {

    int noOfReq, head;

    printf("Enter no of requests: ");

    scanf("%d", &noOfReq);

    int requests[noOfReq], max\_req = INT\_MIN, min\_req = INT\_MAX;

    printf("Enter sequence of requests:\n");

    for (int i = 0; i < noOfReq; i++) {

        scanf("%d", &requests[i]);

        max\_req = (requests[i] >= max\_req) ? requests[i] : max\_req;

        min\_req = (requests[i] <= min\_req) ? requests[i] : min\_req;

    }

    printf("Enter position of head: ");

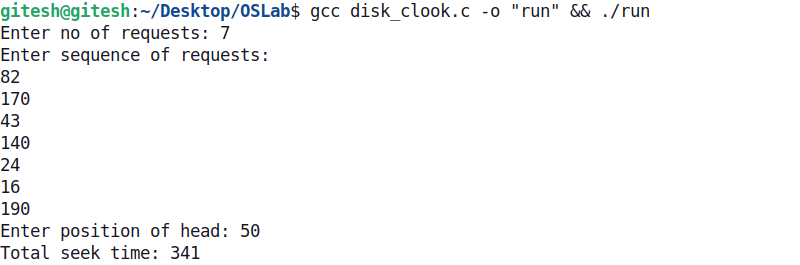
    scanf("%d", &head);

    int ans = abs(max\_req - head) + (max\_req - min\_req) + (min(requests, noOfReq, head) - min\_req);

    printf("Total seek time: %d\n", ans);

}

## Output:



Lab Assignment 9

## Memory Management:

#include <stdio.h>

#include <stdbool.h>

void sort(int arr[], int len) {

    for (int i = 0; i < len; i++) {

        for (int j = 0; j < len-i-1; j++) {

            if (arr[j] >= arr[j+1]) {

                int temp = arr[j+1];

                arr[j+1] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

void inv\_sort(int arr[], int len) {

    for (int i = 0; i < len; i++) {

        for (int j = 0; j < len-i-1; j++) {

            if (arr[j] <= arr[j+1]) {

                int temp = arr[j+1];

                arr[j+1] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

void first\_fit(int proc\_size[], int hole\_size[], int proc, int hole, int alloc[]) {

    int k = 0;

    bool allocated[100] = { false };

    for (int i = 0; i < proc; i++) {

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

            if (proc\_size[i] <= hole\_size[j] && allocated[j] == false) {

                alloc[k] = j+1;

                k++;

                allocated[j] = true;

                break;

            } else {

                alloc[k] = -1;

            }

        }

void worst\_fit(int proc\_size[], int hole\_size[], int proc, int hole, int alloc[]) {

    int k = 0;

    bool allocated[100] = { false };

    inv\_sort(hole\_size, hole);

    for (int i = 0; i < proc; i++) {

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

            if (proc\_size[i] <= hole\_size[j] && allocated[j] == false) {

                alloc[k] = j+1;

                k++;

                allocated[j] = true;

                break;

            } else {

                alloc[k] = -1;

            }

        }

    }

}

void best\_fit(int proc\_size[], int hole\_size[], int proc, int hole, int alloc[]) {

    int k = 0;

    bool allocated[100] = { false };

    sort(hole\_size, hole);

    for (int i = 0; i < proc; i++) {

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

            if (proc\_size[i] <= hole\_size[j] && allocated[j] == false) {

                alloc[k] = j+1;

                k++;

                allocated[j] = true;

                break;

            } else {

                alloc[k] = -1;

            }

        }

    }

}

void display(int P, int proc\_size[], int alloc[]) {

    printf("P.No.\tSize\tBlock\n");

    for (int i = 0; i < P; i++) {

        if (alloc[i] > 0) {

            printf("%d\t%d\t%d\n", i+1, proc\_size[i], alloc[i]);

        } else {

            printf("%d\t%d\t%s\n", i+1, proc\_size[i], "Not allocated");

        }

    }

    printf("\n");

}

int main(){

    int P, H;

    printf("Enter no of processes and no of holes: ");

    scanf("%d%d", &P, &H);

    int P\_size[P], H\_size[H], alloc[P];

    for (int i = 0; i < P; i++) {

        printf("Enter size for process %d: ", i+1);

        scanf("%d", &P\_size[i]);

    }

    for (int i = 0; i < H; i++) {

        printf("Enter size for hole %d: ", i+1);

        scanf("%d", &H\_size[i]);

    }

    int option;

    printf("Enter a choice:\n1. First-fit allocation\n2. Worst-fit allocation\n3. Best-fit allocation\n");

    scanf("%d", &option);

    switch (option) {

        case 1:

            first\_fit(P\_size, H\_size, P, H, alloc);

            display(P, P\_size, alloc);

            break;

        case 2:

            worst\_fit(P\_size, H\_size, P, H, alloc);

            display(P, P\_size, alloc);

            break;

        case 3:

            best\_fit(P\_size, H\_size, P, H, alloc);

            display(P, P\_size, alloc);

            break;

    }

}

## Outputs:

