

DR. B R AMBEDKAR NATIONAL
INSTITUTE OF TECHNOLOGY
JALANDHAR-144011, PUNJAB(INDIA)
COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS (CSX-305)

Submitted By:

Aman Garg

13103050

5th Semester

Submitted To:

Mr. Ashish Kumar

Asst. Professor

Dept. of CSE

July-December, 2015
Lab Practicals Record



EXPERIMENT-1.

To display steps of process creation using fork

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

int main()
{
    pid_t pid; int status;
    printf("Starting process (%d) \n",(int)getpid());

    pid=fork();

    if(pid<0)
        perror("\nCouldn't create a child process.\n");

    else if (pid==0)
    {
        printf("\nChild process (%d) of parent
            (%d)\n",getpid(),getppid());

        exit(NULL);
    }

    else if (pid>0)
    {
        printf("\nParent process (%d)\n",getpid());
        waitpid(pid,&status,0);

        if(WIFEXITED(status))
            printf("Child Exits normally : %d\n", WEXITSTATUS(status));
        else
            perror("Abnormally exits \n");
    }

    return 0;
}
```

EXPERIMENT-2.

To demonstrate exec calls from a process

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

char *path="/home/aman/Desktop/prog/OS/8-3/count";

/* Program to count the number of digits from a minimum to a
   maximum and distributing
   the work in k child processes. eg- ./mncount min max k. Please
   note that the number of digits in the min and the maximum
   should be same due to the way sprintf works.
*/

int main(int argc,char **argv)
{
    if(argc<4)
    {
        perror("\nArguments must be greater than 3\n");
        return -1;
    }

    int llim=atoi(argv[1]);
    int ulim=atoi(argv[2]);
    int mid =atoi(argv[3]);
    int status;
    pid_t pid[mid];
    int i,offset,left,a,a1,a2;

    int diff=ulim-llim;
    if(mid>diff)
    {
        perror("\n Given counts can not be distributed amongst
            entered processes\n");
        return -1;
    }
}
```

```

    }

    offset=diff%mid;
    left=diff/mid;

    for(i=0; i<mid; i++)
    {
        pid[i]=fork();

        if(pid[i]==0)
        {
            printf("\n Child(%d): %d ",i,getpid());

            a=atoi(argv[1]);
            a1=a+ left*i;
            if(i==mid-1)
                a2=a1+left+offset;
            else a2=a+ left*(i+1);

            sprintf(argv[1],"%d",a1);
            sprintf(argv[2],"%d",a2);

            char *new_arg[4]={"count",argv[1],argv[2],NULL};

            int r=execvp(path,new_arg);

            if(r==-1)
                perror("\nCouldn't create child sub-process\n");
        }

        else if(pid[i]>0)
        {
            printf("\n Parent(%d): %d",i,getpid());
            waitpid(pid[i],&status,0);
            printf("\n Child(%d) ",i);

            if(WIFEXITED(status))
                printf("Exits normally : %d\n", WEXITSTATUS(status));

            else if(WIFSTOPPED(status))

```

```

        printf("Stopped By : %d\n", WSTOPSIG(status));

        else if(WIFSIGNALED(status))
            printf("Signalled by : %d\n", WTERMSIG(status));
        else
            perror("Abnormally exits \n");

        continue;
    }

    else perror("\nCouldn't create a process\n");
}
printf("\n");
return 0;
}

```

The code for the called process count:

```

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

int main(int argc,char **argv)
{
    int i;
    int llim=atoi(argv[1]);
    int ulim=atoi(argv[2]);

    printf("\n Displaying count from %d to %d",llim,ulim);

    for(i=llim; i<ulim; i++)
        printf("\n Count : %d ",i);

    printf("\n");
    return 120;
}

```

EXPERIMENT-3.

To demonstrate handling of signals in a process

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

static int sigint_count=0;
static int sigcount=0;

int main()
{
    int pid;

    void child_over(int SIG)
    {
        if(SIG==SIGCHLD)
        {
            sigcount++;
            printf("\n Child has departed(SIGCHLD)");
        }
    }

    void term_if_you_can(int SIG)
    {
        if(SIG==SIGTERM)
        {
            sigcount++;
            printf("\n Attempt to terminate the program");
        }
    }

    void stopped(int SIG)
    {
        if(SIG==SIGTSTP)
        {
            sigcount++;
        }
    }
}
```

```

        printf("\n Can't stop. ");
    }
}

void interrupt(int SIG)
{
    if(SIG==SIGINT)
    {
        sigcount++;
        sigint_count+=1;
        printf("\n Can't interrupt.");
        if (sigint_count==2)
    }
}

void abortion(int SIG)
{
    if(SIG==SIGABRT)
    {
        sigcount++;
        printf("\n Program has aborted");
    }

    signal(SIGCHLD,child_over);
    signal(SIGTERM,term_if_you_can);
    signal(SIGTSTP,stopped);
    signal(SIGABRT,abortion);
    signal(SIGINT,interrupt);

    pid=fork();

    if (pid==0)
    {
        printf("\n Child process (%d): \n",getpid());
        sleep(3);
        raise(SIGABRT);
    }

    else if (pid>0)

```

```
{
    int x=pause();
    printf("\n Slept for a total of %d sec \n",x);

    printf("\n\n Encountered a total of %d signals
           \n\n",sigcount);
}

else
    printf("\n Couldn't create a new process.");
}
return 0;
}
```

EXPERIMENT-4.

To implement 'First Come First Serve' (FCFS) scheduling
(Non-preemptive)

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

static int no_processes;
static int idle_time=0;

typedef struct process{
    int id,arriv_time;
    int burst_time;
    int wait_time;
    int ta_time;
    float resp_ratio;
    float weighted_ta;
}process_list;

typedef struct proc_execution{
    int time;
    int serving;
    int *ready;
}proc_ex;

typedef struct process_queue{
    int pid;
    struct process_queue* next;
}queue;

queue *front,*rear;
void enqueue(int data){
    queue *temp;
    temp=(queue *)malloc(sizeof(queue));
    temp->next=NULL;
    temp->pid=data;
```

```

        if(rear==NULL)
            front=rear=temp;
        else {
            queue *p=front;
            if(rear->pid==data)return;
            while(p!=rear){
                if (p->pid==data)return;
                p=p->next;
            }
            rear->next=temp;
            rear=temp;
        }
    }

int dequeue(){
    queue * p=front;
    int data=0;

    if(p==NULL)return -1;
    if(p->next!=NULL) {
        data=p->pid;
        p=p->next;
        free(front);
        front=p;
    }
    else {
        data=p->pid;
        free(front);
        front=rear=NULL ;
    }
    return data;
}

void create_process_queue(){
    front=rear=NULL;
}

void display_processes(process_list* proc){
    printf("\nID\tArrival\tBurst\tWait\tTurn\tResp\tWeighted TA\n");

```

```

for(int i=0; i<no_processes; i++){
    printf("%d\t",proc[i].id );
    printf("%d\t",proc[i].arriv_time);
    printf("%d\t",proc[i].burst_time);
    printf("%d\t",proc[i].wait_time);
    printf("%d\t",proc[i].ta_time);
    printf("%0.2f\t",proc[i].resp_ratio);
    printf("%0.2f\n",proc[i].weighted_ta);
}
FILE *fp=fopen("graph_proc.txt","w+");
fprintf(fp,"\n\nWait Time\n\n");
for(int i=0; i<no_processes; i++)
    fprintf(fp,"%d %d\t\n",proc[i].id,proc[i].wait_time);

fprintf(fp,"\n\nTurn Aroundd Time\n\n");
for(int i=0; i<no_processes; i++)
    fprintf(fp,"%d %d\t\n",proc[i].id,proc[i].ta_time);

fprintf(fp,"\n\nWeighted Turn Around Time\n\n");
for(int i=0; i<no_processes; i++)
    fprintf(fp,"%d %f\t\n",proc[i].id,proc[i].weighted_ta);

fprintf(fp,"\n\nCompletion Time\n\n");
for(int i=0; i<no_processes; i++)
    fprintf(fp,"%d
        %d\t\n",proc[i].id,proc[i].ta_time-proc[i].arriv_time);
fclose(fp);
}

void display_execution(proc_ex ps){
    printf("%d\t",ps.time );
    if (ps.serving!=0)printf("P%d \t",ps.serving);
    else printf("-\t");

    for(int j=0; j<no_processes; j++)
        if (ps.ready[j]==1)printf("P%d ",j+1);
        else printf("- ");
    printf("\n");
}

```

```

void isReady(process_list* proc,proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        if (proc[i].arriv_time<=ps.time){
            ps.ready[i]=1;
            enqueue(i);
        }
    }
}

int get_min(process_list *proc){
    int minimum=INT_MAX;
    for (int i=0; i<no_processes; i++)
        if (proc[i].arriv_time<=minimum)
            minimum=proc[i].arriv_time;
    return minimum;
}

int not_all_serviced(proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]!=-1)
            return 1;
    }
    return 0;
}

void display_results(process_list*proc){
    float mean_wta=0,mean_ta=0,mean_resp=0,mean_w=0;
    for (int i=0; i<no_processes; i++){
        mean_wta+=proc[i].weighted_ta;
        mean_ta+=proc[i].ta_time;
        mean_resp+=proc[i].resp_ratio;
        mean_w+=proc[i].wait_time;
    }
    printf("\nMean Turn Around Time: %.2f\n",mean_ta/no_processes);
    printf("Mean Weighted Turn Around Time:
        %.2f\n",mean_wta/no_processes);
    printf("Mean Response Ratio: %.2f\n",mean_resp/no_processes);
    printf("Mean Waiting Time: %.2f\n",mean_w/no_processes );
    printf("Mean CPU Idle Time: %d\n\n",idle_time );
}

```

```

}

int main(){

    FILE *fp=fopen("proc_input.txt","r");
    fscanf(fp,"%d",&no_processes);

    process_list*
        proc=(process_list*)malloc(sizeof(process_list)*no_processes);
    assert(proc!=NULL);

    for(int i=0; i<no_processes;
        i++)fscanf(fp,"%d",&proc[i].arriv_time);
    for(int i=0; i<no_processes;
        i++)fscanf(fp,"%d",&proc[i].burst_time);
    for(int i=0; i<no_processes; i++)proc[i].id=i+1;

    printf("\nArrival: \t"); for(int i=0; i<no_processes;
        i++)printf(" %d ",proc[i].arriv_time);
    printf("\nBurst : \t"); for(int i=0; i<no_processes;
        i++)printf(" %d ",proc[i].burst_time);
    printf("\n\nTime\tServe\tReady\n");

    proc_ex ps;
    create_process_queue();

    ps.time=get_min(proc);
    ps.ready=(int *)malloc(sizeof(int)*no_processes);
    isReady(proc,ps);
    ps.serving=dequeue()+1;

    display_execution(ps);

    while(not_all_serviced(ps)){
        int i=ps.serving-1;
        if (i==--1){
            ps.time+=1;
            idle_time+=1;
        }
    }
}

```

```

else{
    ps.time+=proc[i].burst_time;
    proc[i].ta_time=ps.time-proc[i].arriv_time;
    proc[i].wait_time=proc[i].ta_time-proc[i].burst_time;
    proc[i].resp_ratio=(float)proc[i].ta_time/proc[i].burst_time;
    proc[i].weighted_ta=(float)proc[i].ta_time/proc[i].burst_time;
    ps.ready[i]=-1;
}
isReady(proc,ps);
ps.serving=dequeue()+1;
display_execution(ps);
}

display_processes(proc);
display_results(proc);

return 0;
}

```

OUTPUT:

```

Arrival:      0  2  0  4  5
Burst  :      3  4  7  8  1

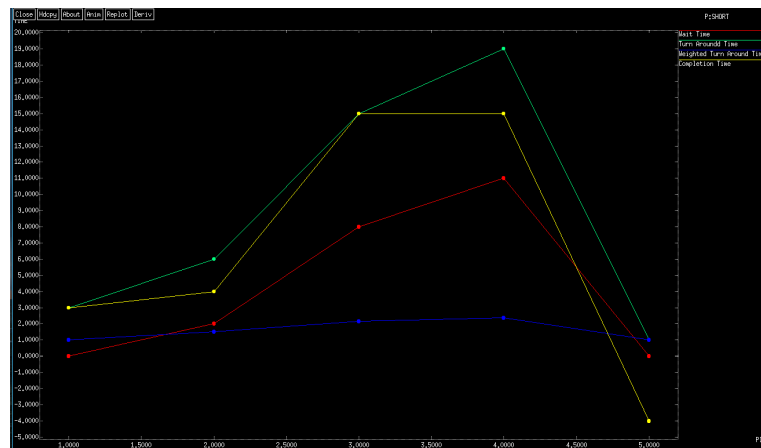
Time   Serve   Ready
0      P1      P1 - P3 - -
3      P3      - P2 P3 - -
10     P2      - P2 - P4 P5
14     P4      - - - P4 P5
22     P5      - - - - P5
23     -      - - - - -

ID      Arrival Burst   Wait   Turn   Resp   Weighted TA
1        0         3      0       3     1.00    1.00
2        2         4      8      12     3.00    3.00
3        0         7      3      10     1.43    1.43
4        4         8     10     18     2.25    2.25
5        5         1     17     18    18.00   18.00

Mean Turn Around Time: 12.20
Mean Weighted Turn Around Time: 5.14
Mean Response Ratio: 5.14
Mean Waiting Time: 7.60
Mean CPU Idle Time: 0

```

GRAPH:



EXPERIMENT-5.

To implement 'Shortest Job First' (SJS) scheduling
(Non-preemptive)

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

static int no_processes;
static int idle_time=0;

void isReady(process_list* proc,proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        if (proc[i].arriv_time<=ps.time)
            ps.ready[i]=1;
    }
}

int get_next_in_ready(process_list *proc,proc_ex ps){
    int minimum=INT_MAX;
    int min_pid=-1;

    if(!not_all_serviced(ps))return 0;
    for (int i=0; i<no_processes; i++){
        if (proc[i].burst_time<minimum && ps.ready[i]==1 ){
            minimum=proc[i].burst_time;
            min_pid=i;
        }
    }
    return min_pid+1;
}

int main(){

    FILE *fp=fopen("proc_input.txt","r");
    fscanf(fp,"%d",&no_processes);
```



```

process_list*
    proc=(process_list*)malloc(sizeof(process_list)*no_processes);
assert(proc!=NULL);

for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].arriv_time);
for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].burst_time);
for(int i=0; i<no_processes; i++)proc[i].id=i+1;

printf("\nArrival: \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].arriv_time);
printf("\nBurst : \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].burst_time);
printf("\n\nTime\tServe\tReady\n");

proc_ex ps;

ps.time=get_min(proc);
ps.ready=(int *)malloc(sizeof(int)*no_processes);
isReady(proc,ps);
ps.serving=get_next_in_ready(proc,ps);

display_execution(ps);

while(not_all_serviced(ps)){
    int i=ps.serving-1;
    if (i== -1){
        ps.time+=1;
        idle_time+=1;
    }

    else{
        ps.time+=proc[i].burst_time;
        proc[i].ta_time=ps.time-proc[i].arriv_time;
        proc[i].wait_time=proc[i].ta_time-proc[i].burst_time;
        proc[i].resp_ratio=(float)proc[i].ta_time/proc[i].burst_time;
        proc[i].weighted_ta=(float)proc[i].ta_time/proc[i].burst_time;
        ps.ready[i]=-1;
    }
}

```

```
        isReady(proc,ps);
        ps.serving=get_next_in_ready(proc,ps);
        display_execution(ps);
    }

    display_processes(proc);
    display_results(proc);

    return 0;
}
```

OUTPUT:

```

Arrival:      0  2  0  4  5
Burst  :      3  4  7  8  1

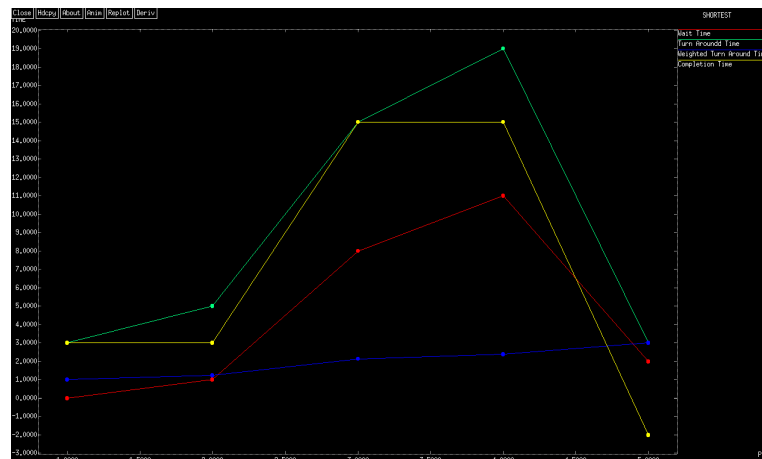
Time   Serve   Ready
0      P1      P1 - P3 - -
3      P2      - P2 P3 - -
7      P5      - - P3 P4 P5
8      P3      - - P3 P4 -
15     P4      - - - P4 -
23     -      - - - - -

ID      Arrival Burst   Wait   Turn   Resp   Weighted TA
1        0        3      0       3     1.00    1.00
2        2        4      1       5     1.25    1.25
3        0        7      8      15     2.14    2.14
4        4        8     11     19     2.38    2.38
5        5        1      2       3     3.00    3.00

Mean Turn Around Time: 9.00
Mean Weighted Turn Around Time: 1.95
Mean Response Ratio: 1.95
Mean Waiting Time: 4.40
Mean CPU Idle Time: 0

```

GRAPH:



EXPERIMENT-6.

To implement 'Highest Response Ration next' (HRR) scheduling
(Non-preemptive)

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

static int no_processes;
static int idle_time=0;

void isReady(process_list* proc,proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        if (proc[i].arriv_time<=ps.time)
            ps.ready[i]=1;
    }
}

int get_next_in_ready(proc_ex ps,int
    curr_waiting,process_list*proc){
    float response_ratio=0,resp;
    int pid=-1;

    if(!not_all_serviced(ps))return 0;
    for(int i=0;i<no_processes; i++){
        if (ps.ready[i]==1){
            resp=(float)(proc[i].burst_time+curr_waiting-proc[i].arriv_time);
            resp/=proc[i].burst_time;
            if (resp>response_ratio){
                response_ratio=resp;
                pid=i;
            }
        }
    }
    return pid+1;
}
```

```

int main(){

    FILE *fp=fopen("proc_input.txt","r");
    fscanf(fp,"%d",&no_processes);

    process_list*
        proc=(process_list*)malloc(sizeof(process_list)*no_processes);
    assert(proc!=NULL);

    for(int i=0; i<no_processes;
        i++)fscanf(fp,"%d",&proc[i].arriv_time);
    for(int i=0; i<no_processes;
        i++)fscanf(fp,"%d",&proc[i].burst_time);
    for(int i=0; i<no_processes; i++)proc[i].id=i+1;

    printf("\nArrival: \t"); for(int i=0; i<no_processes;
        i++)printf(" %d ",proc[i].arriv_time);
    printf("\nBurst : \t"); for(int i=0; i<no_processes;
        i++)printf(" %d ",proc[i].burst_time);
    printf("\n\nTime\tServe\tReady\n");

    proc_ex ps;

    ps.time=get_min(proc);
    ps.ready=(int *)malloc(sizeof(int)*no_processes);
    isReady(proc,ps);
    ps.serving=get_next_in_ready(ps,ps.time,proc);

    display_execution(ps);

    while(not_all_serviced(ps)){
        int i=ps.serving-1;
        if (i== -1){
            ps.time+=1;
            idle_time+=1;
        }

        else{
            ps.time+=proc[i].burst_time;
            proc[i].ta_time=ps.time-proc[i].arriv_time;

```

```

        proc[i].wait_time=proc[i].ta_time-proc[i].burst_time;
        proc[i].resp_ratio=(float)proc[i].ta_time/proc[i].burst_time;
        proc[i].weighted_ta=(float)proc[i].ta_time/proc[i].burst_time;
        ps.ready[i]=-1;
    }

    isReady(proc,ps);
    ps.serving=get_next_in_ready(ps,ps.time,proc);
    display_execution(ps);
}

display_processes(proc);
display_results(proc);

return 0;
}

```

OUTPUT:

```

Arrival:      0  2  0  4  5
Burst  :      3  4  7  8  1

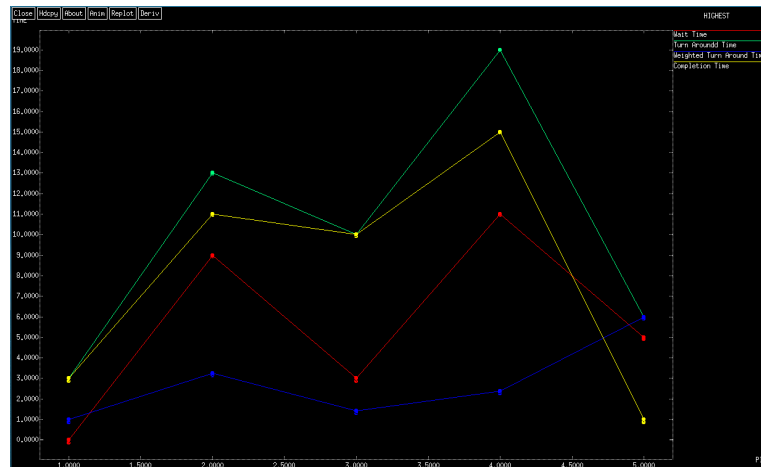
Time   Serve   Ready
0      P1      P1 - P3 - -
3      P3      - P2 P3 - -
10     P5      - P2 - P4 P5
11     P2      - P2 - P4 -
15     P4      - - - P4 -
23     -      - - - - -

ID      Arrival Burst   Wait   Turn   Resp   Weighted TA
1        0         3       0       3      1.00     1.00
2        2         4       9      13     3.25     3.25
3        0         7       3      10     1.43     1.43
4        4         8      11     19     2.38     2.38
5        5         1       5       6     6.00     6.00

Mean Turn Around Time: 10.20
Mean Weighted Turn Around Time: 2.81
Mean Response Ratio: 2.81
Mean Waiting Time: 5.60
Mean CPU Idle Time: 0

```

GRAPH:



EXPERIMENT-7.

To implement 'Shortest Remaining Next' (SRN) scheduling
(Preemptive)

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

static int no_processes;
static int idle_time=0;

void isReady(process_list* proc,proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        if (proc[i].arriv_time<=ps.time)
            ps.ready[i]=1;
    }
}

void store_previous(process_list*proc,process_list *p2){
    for(int i=0; i<no_processes; i++)
        p2[i].burst_time=proc[i].burst_time;
}

void restore_previous(process_list*proc,process_list*p2){
    for(int i=0; i<no_processes; i++)
        proc[i].burst_time=p2[i].burst_time;
}

int get_next_in_ready(process_list *proc,proc_ex ps){
    int minimum=INT_MAX;
    int min_pid=-1;

    if(!not_all_serviced(ps))return 0;
    for (int i=0; i<no_processes; i++){
        if (ps.ready[i]==1 && proc[i].burst_time<minimum){
```



```

        minimum=proc[i].burst_time;
        min_pid=i;
    }
}
return min_pid+1;
}

int main(){

FILE *fp=fopen("proc_input.txt","r");
fscanf(fp,"%d",&no_processes);

process_list*
    proc=(process_list*)malloc(sizeof(process_list)*no_processes);
process_list*
    p_copy=(process_list*)malloc(sizeof(process_list)*no_processes);
assert(proc!=NULL && p_copy!=NULL);

for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].arriv_time);
for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].burst_time);
for(int i=0; i<no_processes; i++)proc[i].id=i+1;

printf("\nArrival: \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].arriv_time);
printf("\nBurst : \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].burst_time);
printf("\n\nTime\tServe\tReady\n");

proc_ex ps;
store_previous(proc,p_copy);

ps.time=get_min(proc);
ps.ready=(int *)malloc(sizeof(int)*no_processes);
isReady(proc,ps);
ps.serving=get_next_in_ready(proc,ps);

display_execution(ps,proc);

```

```

while(not_all_serviced(ps)){
    int i=ps.serving-1;
    ps.time+=1;

    if(i!=-1){
        proc[i].burst_time-=1;

        if (proc[i].burst_time==0){
            ps.ready[i]=-1;
            proc[i].ta_time=ps.time-proc[i].arriv_time;
            proc[i].wait_time=proc[i].ta_time-p_copy[i].burst_time;
            proc[i].resp_ratio=(float)proc[i].ta_time/p_copy[i].burst_time;
            proc[i].weighted_ta=(float)proc[i].ta_time/p_copy[i].burst_time;
        }
    }
    else idle_time+=1;
    isReady(proc,ps);
    ps.serving=get_next_in_ready(proc,ps);
    display_execution(ps,proc);
}

restore_previous(proc,p_copy);
display_processes(proc);
display_results(proc);

return 0;
}

```

OUTPUT:

```

Burst :      3  4  7  8  1

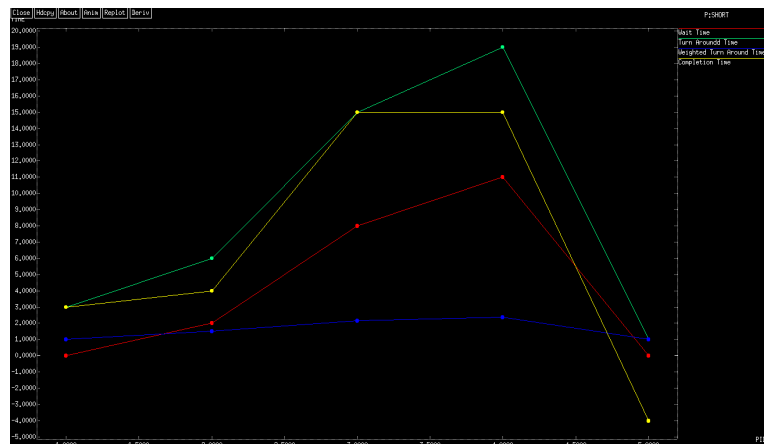
Time  Serve  Ready
0     P1     P1 - P3 - -      3 4 7 8 1
1     P1     P1 - P3 - -      2 4 7 8 1
2     P1     P1 P2 P3 - -      1 4 7 8 1
3     P2     - P2 P3 - -      0 4 7 8 1
4     P2     - P2 P3 P4 -      0 3 7 8 1
5     P5     - P2 P3 P4 P5      0 2 7 8 1
6     P2     - P2 P3 P4 -      0 2 7 8 0
7     P2     - P2 P3 P4 -      0 1 7 8 0
8     P3     - - P3 P4 -      0 0 7 8 0
9     P3     - - P3 P4 -      0 0 6 8 0
10    P3     - - P3 P4 -      0 0 5 8 0
11    P3     - - P3 P4 -      0 0 4 8 0
12    P3     - - P3 P4 -      0 0 3 8 0
13    P3     - - P3 P4 -      0 0 2 8 0
14    P3     - - P3 P4 -      0 0 1 8 0
15    P4     - - - P4 -      0 0 0 8 0
16    P4     - - - P4 -      0 0 0 7 0
17    P4     - - - P4 -      0 0 0 6 0
18    P4     - - - P4 -      0 0 0 5 0
19    P4     - - - P4 -      0 0 0 4 0
20    P4     - - - P4 -      0 0 0 3 0
21    P4     - - - P4 -      0 0 0 2 0
22    P4     - - - P4 -      0 0 0 1 0
23    -      - - - - -      0 0 0 0 0

ID      Arrival Burst  Wait  Turn  Resp  Weighted TA
1        0        3      0      3      1.00    1.00
2        2        4      2      6      1.50    1.50
3        0        7      8     15      2.14    2.14
4        4        8     11     19      2.38    2.38
5        5        1      0      1      1.00    1.00

Mean Turn Around Time: 8.80
Mean Weighted Turn Around Time: 1.60
Mean Response Ratio: 1.60
Mean Waiting Time: 4.20

```

GRAPH:



EXPERIMENT-8.

To implement 'Round Robin' (RR) scheduling
(Preemptive)

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

static int no_processes;
static int quantum;
static int idle_time=0;

void isReady(process_list* proc,proc_ex ps){
    int fair_queue[no_processes];
    int count=0;

    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        else if (ps.ready[i]==1)fair_queue[count++]=i;
        else if (proc[i].arriv_time<=ps.time){
            ps.ready[i]=1;
            enqueue(i);
        }
    }
    for(int i=0; i<count; i++)enqueue(fair_queue[i]);
}

int main(){

    FILE *fp=fopen("proc_input.txt","r");
    fscanf(fp,"%d",&no_processes);

    process_list*
        proc=(process_list*)malloc(sizeof(process_list)*no_processes);
    process_list*
        p_copy=(process_list*)malloc(sizeof(process_list)*no_processes);
    assert(proc!=NULL && p_copy!=NULL);
```

```

    for(int i=0; i<no_processes;
        i++)fscanf(fp,"%d",&proc[i].arriv_time);
for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].burst_time);
for(int i=0; i<no_processes; i++)proc[i].id=i+1;
fscanf(fp,"%d",&quantum);

printf("\nArrival: \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].arriv_time);
printf("\nBurst : \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].burst_time);
printf("\nTime Quantum: \t %d \n",quantum);
printf("\nTime\tServe\tReady\n");

proc_ex ps;
store_previous(proc,p_copy);
create_process_queue();

ps.time=get_min(proc);
ps.ready=(int *)malloc(sizeof(int)*no_processes);
isReady(proc,ps);
ps.serving=dequeue()+1;

display_execution(ps,proc);

while(not_all_serviced(ps)){
    int i=ps.serving-1;
    if(i== -1){
        ps.time+=1;
        idle_time+=1;
    }

    else{
        if (proc[i].burst_time<quantum){
            ps.time+=proc[i].burst_time;
            proc[i].burst_time=0;
        }
        else{
            ps.time+=quantum;
            proc[i].burst_time-=quantum;
        }
    }
}

```

```

    }

    if (proc[i].burst_time==0){
        ps.ready[i]=-1;
        proc[i].ta_time=ps.time-proc[i].arriv_time;
        proc[i].wait_time=proc[i].ta_time-p_copy[i].burst_time;
        proc[i].resp_ratio=(float)proc[i].ta_time/p_copy[i].burst_time;
        proc[i].weighted_ta=(float)proc[i].ta_time/p_copy[i].burst_time;
    }
}
isReady(proc,ps);
ps.serving=dequeue()+1;
display_execution(ps,proc);
}

restore_previous(proc,p_copy);
display_processes(proc);
display_results(proc);

return 0;
}

```

OUTPUT:

```

Arrival:      0  2  0  4  5
Burst  :      3  4  7  8  1
Time Quantum: 2

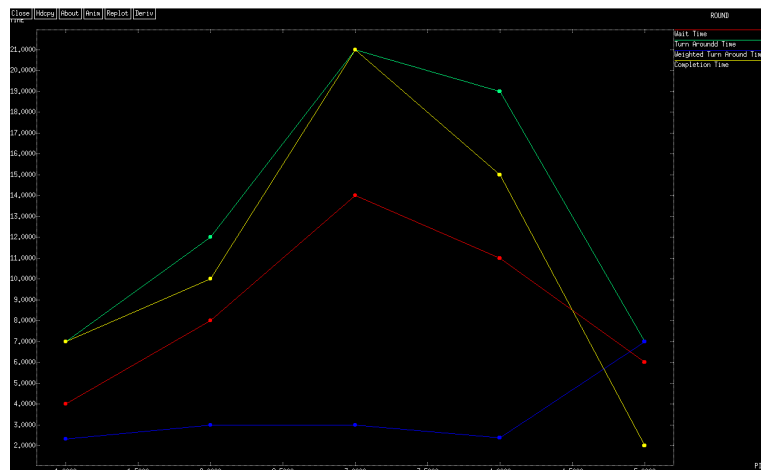
Time  Serve  Ready
0     P1     P1 - P3 - -   3 4 7 8 1
2     P3     P1 P2 P3 - -   1 4 7 8 1
4     P2     P1 P2 P3 P4 -   1 4 5 8 1
6     P1     P1 P2 P3 P4 P5  1 2 5 8 1
7     P4     - P2 P3 P4 P5  0 2 5 8 1
9     P3     - P2 P3 P4 P5  0 2 5 6 1
11    P5     - P2 P3 P4 P5  0 2 3 6 1
12    P2     - P2 P3 P4 -   0 2 3 6 0
14    P4     - - P3 P4 -   0 0 3 6 0
16    P3     - - P3 P4 -   0 0 3 4 0
18    P4     - - P3 P4 -   0 0 1 4 0
20    P3     - - P3 P4 -   0 0 1 2 0
21    P4     - - - P4 -   0 0 0 2 0
23    -     - - - - -   0 0 0 0 0

ID      Arrival Burst  Wait  Turn  Resp  Weighted TA
1        0        3      4      7    2.33    2.33
2        2        4      8     12    3.00    3.00
3        0        7     14     21    3.00    3.00
4        4        8     11     19    2.38    2.38
5        5        1      6      7     7.00    7.00

Mean Turn Around Time: 13.20
Mean Weighted Turn Around Time: 3.54
Mean Response Ratio: 3.54
Mean Waiting Time: 8.60
Mean CPU Idle Time: 0

```

GRAPH:



EXPERIMENT-9.

To implement 'Least Completed Next' (LCN) scheduling (Preemptive)

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

static int no_processes;
static int idle_time=0;

void isReady(process_list* proc,proc_ex ps){
    for(int i=0; i<no_processes; i++){
        if (ps.ready[i]==-1)continue;
        if (proc[i].arriv_time<=ps.time)
            ps.ready[i]=1;
    }
}

int get_next_in_ready(process_list *proc,proc_ex ps){
    int minimum=INT_MAX;
    int min_pid=-1;

    if(!not_all_serviced(ps))return 0;
    for (int i=0; i<no_processes; i++){
        if (ps.ready[i]==1 && proc[i].consumed_time<minimum){
            minimum=proc[i].consumed_time;
            min_pid=i;
        }
    }
    return min_pid+1;
}

int main(){

    FILE *fp=fopen("proc_input.txt","r");
    fscanf(fp,"%d",&no_processes);
```



```

process_list*
    proc=(process_list*)malloc(sizeof(process_list)*no_processes);
process_list*
    p_copy=(process_list*)malloc(sizeof(process_list)*no_processes);
assert(proc!=NULL && p_copy!=NULL);

for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].arriv_time);
for(int i=0; i<no_processes;
    i++)fscanf(fp,"%d",&proc[i].burst_time);
for(int i=0; i<no_processes; i++)proc[i].id=i+1;

printf("\nArrival: \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].arriv_time);
printf("\nBurst : \t"); for(int i=0; i<no_processes;
    i++)printf(" %d ",proc[i].burst_time);
printf("\n\nTime\tServe\tReady\n");

proc_ex ps;
store_previous(proc,p_copy);

ps.time=get_min(proc);
ps.ready=(int *)malloc(sizeof(int)*no_processes);
isReady(proc,ps);
ps.serving=get_next_in_ready(proc,ps);

display_execution(ps,proc);

while(not_all_serviced(ps)){
    int i=ps.serving-1;
    if(i==-1){
        ps.time+=1;
        idle_time+=1;
    }

    else{
        ps.time+=1;
        proc[i].burst_time-=1;
        proc[i].consumed_time+=1;
    }
}

```

```

        if (proc[i].burst_time==0){
            ps.ready[i]=-1;
            proc[i].ta_time=ps.time-proc[i].arriv_time;
            proc[i].wait_time=proc[i].ta_time-p_copy[i].burst_time;
            proc[i].resp_ratio=(float)proc[i].ta_time/p_copy[i].burst_time;
            proc[i].weighted_ta=(float)proc[i].ta_time/p_copy[i].burst_time;
        }
    }
    isReady(proc,ps);
    ps.serving=get_next_in_ready(proc,ps);
    display_execution(ps,proc);
}

restore_previous(proc,p_copy);
display_processes(proc);
display_results(proc);

return 0;
}

```

OUTPUT:

```

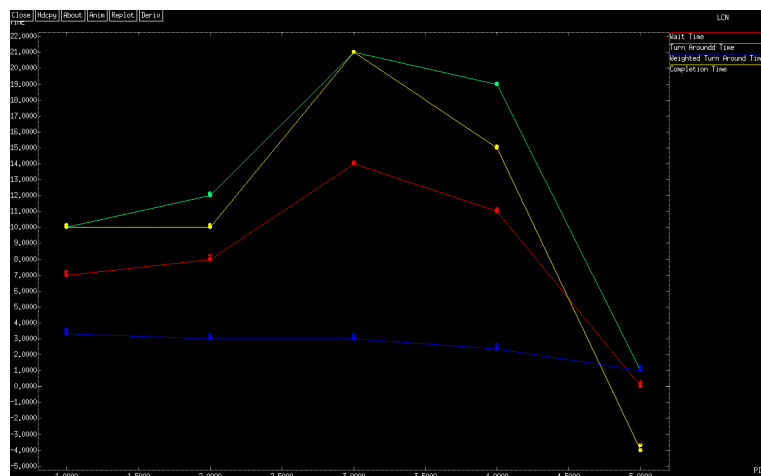
Time   Serve   Ready
0      P1      P1 - P3 - -   0 0 0 0 0
1      P3      P1 - P3 - -   1 0 0 0 0
2      P2      P1 P2 P3 - -   1 0 1 0 0
3      P1      P1 P2 P3 - -   1 1 1 0 0
4      P4      P1 P2 P3 P4 -   2 1 1 0 0
5      P5      P1 P2 P3 P4 P5 2 1 1 1 0
6      P2      P1 P2 P3 P4 -   2 1 1 1 1
7      P3      P1 P2 P3 P4 -   2 2 1 1 1
8      P4      P1 P2 P3 P4 -   2 2 2 1 1
9      P1      P1 P2 P3 P4 -   2 2 2 2 1
10     P2      - P2 P3 P4 -   3 2 2 2 1
11     P3      - P2 P3 P4 -   3 3 2 2 1
12     P4      - P2 P3 P4 -   3 3 3 2 1
13     P2      - P2 P3 P4 -   3 3 3 3 1
14     P3      - - P3 P4 -   3 4 3 3 1
15     P4      - - P3 P4 -   3 4 4 3 1
16     P3      - - P3 P4 -   3 4 4 4 1
17     P4      - - P3 P4 -   3 4 5 4 1
18     P3      - - P3 P4 -   3 4 5 5 1
19     P4      - - P3 P4 -   3 4 6 5 1
20     P3      - - P3 P4 -   3 4 6 6 1
21     P4      - - - P4 -   3 4 7 6 1
22     P4      - - - P4 -   3 4 7 7 1
23     -      - - - - -   3 4 7 8 1

ID      Arrival Burst   Wait   Turn   Resp   Weighted TA
1        0         3       7      10     3.33    3.33
2        2         4       8      12     3.00    3.00
3        0         7      14      21     3.00    3.00
4        4         8      11      19     2.38    2.38
5        5         1       0       1      1.00    1.00

Mean Turn Around Time: 12.60
Mean Weighted Turn Around Time: 2.54
Mean Response Ratio: 2.54
Mean Waiting Time: 8.00

```

GRAPH:



EXPERIMENT-10.

To implement Page Replacement Techniques

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

void lru_replacement(vector<int> & input_pages, const int &
    page_size){
    cout<<"\n LEAST RECENTLY USED \n";

    static int page_fault=0;
    static int page_hits=0;

    deque<int>window;
    vector<int>::iterator it=input_pages.begin();
    deque<int>::iterator qt=window.begin();

    for(;it!=input_pages.end(); it++){
        cout<<"\n For: "<<*it;
        if (find(window.begin(),window.end(),*it)!=window.end())
            page_hits+=1;

        else{
            page_fault+=1;
            if(int(window.size())==page_size) {
                //Replace the least recently used

                map<int,int> queue_used;
                for(qt=window.begin() ;qt!=window.end(); qt++)
                    queue_used.insert(make_pair(*qt,0));

                //Find the current index of the loop => j
                //From j down to 0 check which first page_size -1
                numbers have appeared
            }
        }
    }
}
```

```

    int current_index=it-input_pages.begin();
    int priority=0;
    int element_to_be_replaced=0;

    vector<bool>already_marked(input_pages.size(),false);
    for(int j=current_index-1; j>=0; j--){
        if (queue_used.find(input_pages[j])!=queue_used.end()
            && !already_marked[input_pages[j]]){
            queue_used[input_pages[j]]=priority++;
            already_marked[input_pages[j]]=true;
        }
    }

    int temp=INT_MIN;

    map<int,int>:: iterator mi=queue_used.begin();
    for(;mi!=queue_used.end();mi++){
        if (mi->second>temp){
            element_to_be_replaced=mi->first;
            temp=mi->second;
        }
    }

    cout<<"\t Replacing : "<<element_to_be_replaced<<" with:
        "<<*it;
    int replace_index =
        find(window.begin(),window.end(),element_to_be_replaced)-window.begin();
    window[replace_index]=*it;

}
else window.push_back(*it);
cout<<"\t Queue: ";
for(qt=window.begin() ;qt!=window.end(); qt++)
    cout<<" "<<*qt;
}

}

cout<<"\n\nTotal page faults: "<<page_fault;
cout<<"\nTotal page hits: "<<page_hits<<endl;

```

```

}

void optimal_replacement(vector<int> & input_pages, const int &
    page_size){
    cout<<"\n OPTIMAL REPLACEMENT \n";

    static int page_fault=0;
    static int page_hits=0;

    deque<int>window;
    vector<int>::iterator it=input_pages.begin();
    deque<int>::iterator qt=window.begin();

    for(;it!=input_pages.end(); it++){
        cout<<"\n For: "<<*it;
        if (find(window.begin(),window.end(),*it)!=window.end())
            page_hits+=1;

        else{
            page_fault+=1;
            if(int(window.size())==page_size) {
                //Replace the to-be least recently used

                map<int,int> queue_used;
                for(qt=window.begin() ;qt!=window.end(); qt++)
                    queue_used.insert(make_pair(*qt,int(input_pages.size())));

                //Find the current index of the loop => j
                //From j upto n check which first page_size -1 numbers
                have appeared

                int current_index=it-input_pages.begin();
                int priority=0;
                int element_to_be_replaced=0;

                vector<bool>already_marked(input_pages.size(),false);

                for(int j=current_index; j<int(input_pages.size()); j++){
                    if (queue_used.find(input_pages[j])!=queue_used.end()

```

```

        && !already_marked[input_pages[j]]){
            queue_used[input_pages[j]]=priority++;
            already_marked[input_pages[j]]=true;
        }
    }

    int temp=INT_MIN;

    map<int,int>:: iterator mi=queue_used.begin();
    for(;mi!=queue_used.end();mi++){
        if (mi->second>temp){
            element_to_be_replaced=mi->first;
            temp=mi->second;
        }
    }

    cout<<"\t Replacing : "<<element_to_be_replaced<<" with:
        "<<*it;
    int replace_index =
        find(window.begin(),window.end(),element_to_be_replaced)-window.begin();
    window[replace_index]=*it;

}
else window.push_back(*it);
cout<<"\t Queue: ";
for(qt=window.begin() ;qt!=window.end(); qt++)
    cout<<" "<<*qt;
}

}

cout<<"\n\nTotal page faults: "<<page_fault;
cout<<"\nTotal page hits: "<<page_hits<<endl;

}

void fifo_replacement(vector<int> & input_pages,const int &
    page_size){
    cout<<"\n FIRST IN FIRST OUT \n";

    static int page_fault=0;

```

```

static int page_hits=0;

deque<int>window;
vector<int>::iterator it=input_pages.begin();
deque<int>::iterator qt=window.begin();

for(;it!=input_pages.end(); it++){
    cout<<"\n For: "<<*it;
    if (find(window.begin(),window.end(),*it)!=window.end())
        page_hits+=1;

    else{
        page_fault+=1;
        if(int(window.size())==page_size) window.pop_front();
        window.push_back(*it);
        cout<<"\t Queue: ";
        for(qt=window.begin() ;qt!=window.end(); qt++)
            cout<<" "<<*qt;
    }

}

cout<<"\n\nTotal page faults: "<<page_fault;
cout<<"\nTotal page hits: "<<page_hits<<endl;

}

int main(){
    char input;
    vector<int>input_pages;

    ifstream infile("input.txt");

    while((input=infile.get())!=EOF){
        if (int(input)==32)continue;
        input_pages.push_back(input-'0');
    }

    infile.close();

```



```

//Enter the variable page_size here
const int page_size=4;

fifo_replacement(input_pages,page_size);
lru_replacement(input_pages,page_size);
optimal_replacement(input_pages,page_size);

return 0;
}

```

For Input File:

```
1 2 3 4 5 3 4 1 6 7 8 7 8 9 7 8 9 5 4 5 4 2
```

Output:

```

Terminal
FIRST IN FIRST OUT
For: 1 Queue: 1
For: 2 Queue: 1 2
For: 3 Queue: 1 2 3
For: 4 Queue: 1 2 3 4
For: 5 Queue: 2 3 4 5
For: 3
For: 4
For: 1 Queue: 3 4 5 1
For: 6 Queue: 4 5 1 6
For: 7 Queue: 5 1 6 7
For: 8 Queue: 1 6 7 8
For: 7
For: 8
For: 9 Queue: 6 7 8 9
For: 7
For: 8
For: 9
For: 5 Queue: 7 8 9 5
For: 4 Queue: 8 9 5 4
For: 5
For: 4
For: 2 Queue: 9 5 4 2
Total page faults: 13
Total page hits: 9

```

Terminal		
LEAST RECENTLY USED		
For: 1	Queue: 1	
For: 2	Queue: 1 2	
For: 3	Queue: 1 2 3	
For: 4	Queue: 1 2 3 4	
For: 5	Replacing : 1 with: 5	Queue: 5 2 3 4
For: 3		
For: 4		
For: 1	Replacing : 2 with: 1	Queue: 5 1 3 4
For: 6	Replacing : 5 with: 6	Queue: 6 1 3 4
For: 7	Replacing : 3 with: 7	Queue: 6 1 7 4
For: 8	Replacing : 4 with: 8	Queue: 6 1 7 8
For: 7		
For: 8		
For: 9	Replacing : 1 with: 9	Queue: 6 9 7 8
For: 7		
For: 8		
For: 9		
For: 5	Replacing : 6 with: 5	Queue: 5 9 7 8
For: 4	Replacing : 7 with: 4	Queue: 5 9 4 8
For: 5		
For: 4		
For: 2	Replacing : 8 with: 2	Queue: 5 9 4 2
Total page faults: 13		
Total page hits: 9		
Terminal		
OPTIMAL REPLACEMENT		
For: 1	Queue: 1	
For: 2	Queue: 1 2	
For: 3	Queue: 1 2 3	
For: 4	Queue: 1 2 3 4	
For: 5	Replacing : 2 with: 5	Queue: 1 5 3 4
For: 3		
For: 4		
For: 1		
For: 6	Replacing : 1 with: 6	Queue: 6 5 3 4
For: 7	Replacing : 3 with: 7	Queue: 6 5 7 4
For: 8	Replacing : 6 with: 8	Queue: 8 5 7 4
For: 7		
For: 8		
For: 9	Replacing : 4 with: 9	Queue: 8 5 7 9
For: 7		
For: 8		
For: 9		
For: 5		
For: 4	Replacing : 7 with: 4	Queue: 8 5 4 9
For: 5		
For: 4		
For: 2	Replacing : 4 with: 2	Queue: 8 5 2 9
Total page faults: 11		
Total page hits: 11		

EXPERIMENT-11.

To implement various memory allocation techniques

```
#include <bits/stdc++.h>
using namespace std;

static int curr_match;

void display_block(vector<int> & blocks){
    vector<int>:: iterator it=blocks.begin();
    for(;it!=blocks.end();it++)
        cout<<" "<<*it;
}

bool predicate_first_fit(int i){
    /*A filter function that returns true if the block is able to
       store the required memory
       denoted by i and curr_match respectively*/
    return i>=curr_match;
}

void first_fit(vector<int>& blocks,vector<int> & procs){

    cout<<" FIRST FIT MEMORY ALLOCATION: ";
    cout<<"\t Free Blocks: "; display_block(blocks);
    cout<<endl;

    vector<int>:: iterator it=procs.begin();
    for(;it!=procs.end();it++){

        curr_match=*it;

        vector<int>:: iterator si =
            find_if(blocks.begin(),blocks.end(),predicate_first_fit);

        if (si==blocks.end())
            cout<<"\n"<<*it<<"K must wait for memory.";

        else{
```

```

        cout<<"\n"<<*it<<"K is put in "<<*si<<"K partition";
        int index_of_block = si-blocks.begin();
        blocks[index_of_block]-=*it;
    }
    cout<<"\t Free Blocks: "; display_block(blocks);
}
cout<<endl<<endl;
}

void best_fit(vector<int>& blocks,vector<int> & procs){

    cout<<" BEST FIT MEMORY ALLOCATION: ";
    cout<<"\t Free Blocks: "; display_block(blocks);
    cout<<endl;

    vector<int>:: iterator it=procs.begin();
    for(;it!=procs.end();it++){

        sort(blocks.begin(),blocks.end());
        vector<int>:: iterator si =
            upper_bound(blocks.begin(),blocks.end(),*it);

        if (si==blocks.end())
            cout<<"\n"<<*it<<"K must wait for memory.";

        else{
            cout<<"\n"<<*it<<"K is put in "<<*si<<"K partition";
            int index_of_block = si-blocks.begin();
            blocks[index_of_block]-=*it;
        }
        cout<<"\t Free Blocks: "; display_block(blocks);
    }
    cout<<endl<<endl;
}

void worst_fit(vector<int>& blocks,vector<int> & procs){

    cout<<" WORST FIT MEMORY ALLOCATION: ";
    cout<<"\t Free Blocks: "; display_block(blocks);
    cout<<endl;
}

```

```

vector<int>:: iterator it=procs.begin();
for(;it!=procs.end();it++){

    curr_match=*it;
    auto largest_block_index=0;
    auto largest_block_size=0;

    vector<int>:: iterator si =
        find_if(blocks.begin(),blocks.end(),predicate_first_fit);

    if (si==blocks.end())
        cout<<"\n"<<*it<<"K must wait for memory.";

    else{
        while (si!=blocks.end()){
            if (*si>largest_block_size){
                largest_block_size = *si;
                largest_block_index = si - blocks.begin();
            }
            si = find_if(next(si),blocks.end(),predicate_first_fit);
        }

        cout<<"\n"<<*it<<"K is put in "<<largest_block_size<<"K
            partition";
        blocks[largest_block_index]-=*it;
    }
    cout<<"\t Free Blocks: "; display_block(blocks);
}
cout<<endl<<endl;
}

void next_fit(vector<int>& blocks,vector<int> & procs){

    cout<<" NEXT FIT MEMORY ALLOCATION: ";
    cout<<"\t Free Blocks: "; display_block(blocks);
    cout<<endl;
    int offset=0;

    vector<int>:: iterator it=procs.begin();

```

```

for(;it!=procs.end();it++){

    curr_match=*it;
    vector<int>:: iterator si =
        find_if(blocks.begin()+offset,blocks.end(),predicate_first_fit);

    if (si==blocks.end())
        cout<<"\n"<<*it<<"K must wait for memory.";

    else{
        cout<<"\n"<<*it<<"K is put in "<<*si<<"K partition";
        int index_of_block = si-blocks.begin();
        offset=index_of_block+1;
        if (offset==blocks.size())offset=0;

        blocks[index_of_block]-=*it;
    }
    cout<<"\t Free Blocks: "; display_block(blocks);
}
cout<<endl<<endl;
}
int main(){

    vector<int> blocks,procs;
    string size,buf;
    cout<<"Enter the partition details: ";

    getline(cin,size);
    stringstream ss(size);
    while(ss>>buf)
        blocks.push_back(stoi(buf));

    cout<<"Enter the memory required by processes: ";

    getline(cin,size);
    stringstream ss1(size);
    while(ss1>>buf)
        procs.push_back(stoi(buf));

    first_fit(blocks,procs);
}

```

```

    best_fit(blocks,procs);
    worst_fit(blocks,procs);
    next_fit(blocks,procs);

    return 0;
}

```

OUTPUT:

```

Enter the partition details: 150 100 170 300 50
Enter the memory required by processes: 142 452 150 50 50 70 30
FIRST FIT MEMORY ALLOCATION:      Free Blocks:  150 100 170 300 50

142K is put in 150K partition      Free Blocks:  8 100 170 300 50
452K must wait for memory.         Free Blocks:  8 100 170 300 50
150K is put in 170K partition      Free Blocks:  8 100 20 300 50
50K is put in 100K partition       Free Blocks:  8 50 20 300 50
50K is put in 50K partition        Free Blocks:  8 0 20 300 50
70K is put in 300K partition       Free Blocks:  8 0 20 230 50
30K is put in 230K partition       Free Blocks:  8 0 20 200 50

BEST FIT MEMORY ALLOCATION:      Free Blocks:  8 0 20 200 50

142K is put in 200K partition      Free Blocks:  0 8 20 50 58
452K must wait for memory.         Free Blocks:  0 8 20 50 58
150K must wait for memory.         Free Blocks:  0 8 20 50 58
50K is put in 58K partition        Free Blocks:  0 8 20 50 8
50K must wait for memory.          Free Blocks:  0 8 8 20 50
70K must wait for memory.          Free Blocks:  0 8 8 20 50
30K is put in 50K partition        Free Blocks:  0 8 8 20 20

WORST FIT MEMORY ALLOCATION:      Free Blocks:  0 8 8 20 20

142K must wait for memory.         Free Blocks:  0 8 8 20 20
452K must wait for memory.         Free Blocks:  0 8 8 20 20
150K must wait for memory.         Free Blocks:  0 8 8 20 20
50K must wait for memory.          Free Blocks:  0 8 8 20 20
50K must wait for memory.          Free Blocks:  0 8 8 20 20
70K must wait for memory.          Free Blocks:  0 8 8 20 20
30K must wait for memory.          Free Blocks:  0 8 8 20 20

NEXT FIT MEMORY ALLOCATION:      Free Blocks:  0 8 8 20 20

142K must wait for memory.         Free Blocks:  0 8 8 20 20
452K must wait for memory.         Free Blocks:  0 8 8 20 20
150K must wait for memory.         Free Blocks:  0 8 8 20 20
50K must wait for memory.          Free Blocks:  0 8 8 20 20
50K must wait for memory.          Free Blocks:  0 8 8 20 20
70K must wait for memory.          Free Blocks:  0 8 8 20 20
30K must wait for memory.          Free Blocks:  0 8 8 20 20

```

EXPERIMENT-12.

To demonstrate deadlock detection in operating systems (Banker's Algorithm)

```
#include <bits/stdc++.h>
using namespace std;

// Demonstrate Deadlock avoidance in Operating Systems by using
// bankers's algorithm

int main(){

    int num_proc,resources;

    cout<<"\nNumber of processes "; cin>>num_proc;
    cout<<"\nNumber of resource types "; cin>>resources;

    int avail_res[resources];
    int max_needed[num_proc][resources];
    int allocated[num_proc][resources];

    cout<<"\nAvailable units of resources: ";
    for(int i=0; i<resources; i++)cin>>avail_res[i];

    cout<<"\nMaximum resources the processes take: \n";
    for(int i=0; i<num_proc; i++){
        for(int j=0; j<resources; j++)
            cin >> max_needed[i][j];
    }

    cout<<"\nAllocated resources the processes have: \n";
    for(int i=0; i<num_proc; i++){
        for(int j=0; j<resources; j++)
            cin >> allocated[i][j];
    }

    //Check for enough resources
    //See if the processes release enough resources for a particular
    //process's need
```



```

//Compute a sum of all allocated resources
vector<int>total_allocated(resources);

for(int j=0; j<resources; j++){
    total_allocated[j]= avail_res[j];
    for(int i= 0; i < num_proc; i++)
        total_allocated[j]+=allocated[i][j];
}

//For each process, if need for each resource <= sum of all
//other processes's allocated resource
bool insufficient_resources = false;
vector <int> resource_heavy;

for(int i= 0; i < num_proc; i++){
    bool enough_allocated = true;
    for(int j=0; j<resources; j++){
        if (max_needed[i][j]-allocated[i][j] >
            total_allocated[j]-allocated[i][j])
            enough_allocated = false;
    }
    if(!enough_allocated){
        insufficient_resources = true;
        resource_heavy.push_back(i+1);
    }
}

if (insufficient_resources){
    cout<<"\n Requested greater resources than available: ";
    cout<<"\n Heavy processes is/are : ";
    vector<int>:: iterator si = resource_heavy.begin();
    for(; si!=resource_heavy.end(); si++)
        cout<<"-->"<<*si;
    cout<<endl;
    return -1;
}

// Create a sequence to hold the no deadlock allocations
vector<int>final_sequence;

```

```

while(final_sequence.size()!=num_proc){
    // Possible bug: if the released resources till now do not
    // suffice the minimum need of the
    // remaining processes

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

        if(find(final_sequence.begin(),final_sequence.end(),i+1)!=final_sequence.end())
            continue;

        bool possible = true;

        for(int j=0; j<resources; j++)
            if (avail_res[j] < (max_needed[i][j]-allocated[i][j]))
                possible = false;

        if (possible){
            // Greedily allocate resource to the process
            final_sequence.push_back(i+1);

            // Release the allocated resources
            for(int j=0; j<resources; j++)
                avail_res[j] += allocated[i][j];
        }
    }
}

cout<<"\nThe sequence to avoid deadlock is : ";
vector<int>:: iterator si = final_sequence.begin();
for(; si!=final_sequence.end(); si++)
    cout<<"-->"<<"P("<<*si-1<<")";

cout<<endl;

return 0;
}

```

OUTPUT:

```
Number of processes 4
Number of resource types 3
Available units of resources: 3 3 2
Maximum resources the processes take:
7 5 3
3 2 2
9 0 4
2 2 2
Allocated resources the processes have:
0 1 0
2 0 0
4 0 1
2 1 1
The sequence to avoid deadlock is : -->2-->4-->1-->3
```

```
Number of processes 5
Number of resource types 3
Available units of resources: 0 1 0
Maximum resources the processes take:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocated resources the processes have:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2
Requested greater resources than available:
Heavy processes is/are : -->1-->3
```

EXPERIMENT-13.

To simulate mutex synchronization in an Operating System

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

int global_sum=0;

/* A program to print the sum of a numbers given using command
   line arguments
   using separate threads and display the final result. Use mutex
   locks to guarantee synchronisation.
*/

typedef struct thread_info{
    pthread_t thread_id;
    int thread_no;
    void *argument;
}thread;

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

void* thread_sum(void *x){

    printf("\n Thread with arg %d awaiting lock\n",*(int *)x);
    pthread_mutex_lock(&mutex);

    printf(" Thread with arg %d holding lock\n",*(int *)x);
    global_sum=0;
    int y = *(int *)x;
    for(int i=1; i<=y; i++){

        global_sum+=i;
    }

    printf(" mThread with arg %d releasing lock\n",*(int *)x);
    *(int *) x = global_sum;
```

```

    pthread_mutex_unlock(&mutex);
    pthread_exit(x);
}

void print_error(char *message){
    fprintf(stderr, "%s\n",message);
    exit(EXIT_FAILURE);
}

int main(int argc,char **argv)
{
    int s;
    if(argc<2)
        print_error("Arguments !> 2");

    pthread_attr_t attr;                                     //Set
        of attribites for our own thread structure
    s=pthread_attr_init(&attr);
        //initialise the attributes for the thread stack
        //assert(s==0);

    thread *tid=(thread *)calloc(argc-1,sizeof(thread));
        //Allocate memory for threads
    if(tid==NULL)
        print_error("Couldn't Allocate Memory");

    int arg_threads[argc - 1];

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

        arg_threads[i] = atoi(argv[i+1]);
        tid[i].argument = &arg_threads[i];
        //Set the
        argument as the input
        tid[i].thread_no=i+1;

        printf("\n Creating Thread %d with arg:
            %d\n",tid[i].thread_no,arg_threads[i]);
        s=pthread_create(&tid[i].thread_id,&attr,thread_sum,tid[i].argument);

```

```

        //Create the thread in background
        assert (s==0);
    }

    s=pthread_attr_destroy(&attr);           //We don't need the
        attributes now as the threads are up and running.
    assert(s==0);

    void * status ;                          //To get the value from the
        thread

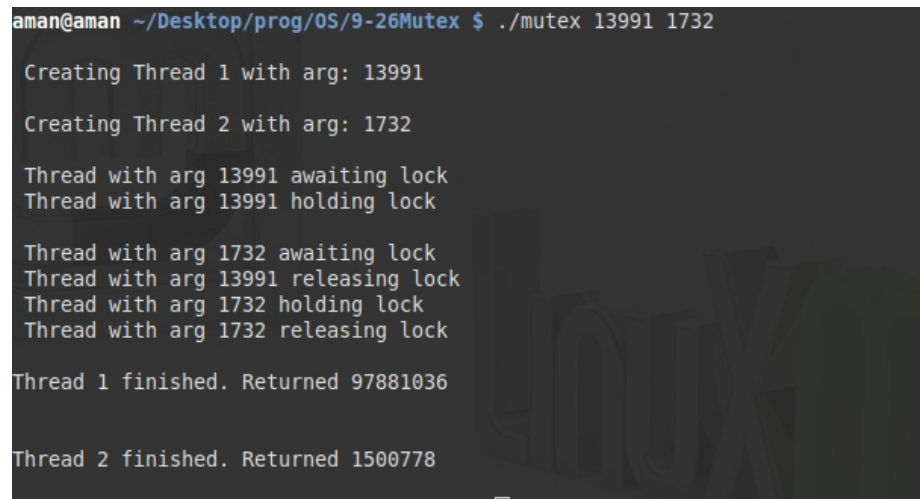
    for(int i=0; i<argc-1; i++){
        s=pthread_join(tid[i].thread_id,&status);
        //make calling thread wait for the called thread

        printf("\n Thread %d finished. Returned %d
            \n\n",tid[i].thread_no ,*(int *)status);
    }

    return 0;
}

```

OUTPUT:



```

aman@aman ~/Desktop/prog/05/9-26Mutex $ ./mutex 13991 1732

Creating Thread 1 with arg: 13991
Creating Thread 2 with arg: 1732

Thread with arg 13991 awaiting lock
Thread with arg 13991 holding lock

Thread with arg 1732 awaiting lock
Thread with arg 13991 releasing lock
Thread with arg 1732 holding lock
Thread with arg 1732 releasing lock

Thread 1 finished. Returned 97881036

Thread 2 finished. Returned 1500778

```

EXPERIMENT-14.

To simulate file allocation strategies

```
#include <iostream>
#include <map>
#include <set>
#include <list>
#include <vector>
#include <cstdlib>

using namespace std;
#define clear system("clear")

typedef map<int, pair< pair<int,int>,bool > > map_cont;
typedef map<int,list<int> > map_link;
typedef map<int,pair<int,vector<int> > > map_index;

map_cont cont_create(int file_no,map_cont &directory,int size,int
    &free_space){

    // Employ first fit if possible
    map_cont::iterator mit = directory.begin();
    bool file_created = false;

    if (mit==directory.end() && size<free_space){
        directory.insert(make_pair(file_no,
            make_pair(make_pair(0,size) ,true) ) );
        file_created = true;
        free_space -= size;
        return directory;
    }

    for(;mit!=directory.end(); mit++){
        //Look for the first hole possible to accomodate

        if (mit->second.second==true) continue;
        else if (mit->second.second== false ){
            if (mit->second.first.second - mit->second.first.first
                >= size){
```

```

        mit->second.first.second = mit->second.first.first+
            size;
        mit->second.second = true;
        // mit->first = file_no;
        file_created = true;
        free_space -= size;
        break;
    }
}

}

if (!file_created && size < free_space){
    mit = --directory.end();
    int last_limit = mit->second.first.second;
    free_space -= size;
    directory.insert(make_pair(file_no,
        make_pair(make_pair(last_limit, last_limit+size) , true) )
    );
}

else if(!file_created)
    cout<<"\n File cannot be created now. Try defragmentation";

return directory;
}

int cont_defragment(map_cont& directory, int & size_hole){

    cout<<"\n\n Current hole size: "+ size_hole;
    cout<<"\n Trying defragmentation: \n";

    map_cont:: iterator mit = directory.begin();
    if(mit==directory.end())
        return size_hole;

    int last_upper_bound = directory.begin()->second.first.second;
    //Stores the last bound of first file
    mit++;

    for(; mit!=directory.end(); mit++){

```



```

        if (mit->second.first.first != last_upper_bound ) {
            size_hole -= mit->second.first.first - last_upper_bound;
            int actual_size = mit->second.first.second -
                mit->second.first.first;
            mit->second.first.first = last_upper_bound;
            mit->second.first.second = mit->second.first.first +
                actual_size;
        }
        last_upper_bound = mit->second.first.second;
    }

    cout<<"\n The new hole size: "<<size_hole;
    return size_hole;
}

void contiguous_mode(){

    clear;
    int max_size,choice = 1;
    cout<<"\n Enter maximum size of allocation "; cin>>max_size;
    int free_space = max_size;
    int max_block_possible = free_space;
    int file_count = 0,size_hole = 0;
    map_cont directory;

    while(choice) {
        clear;
        cout<<"\n\n FREE(Actual): "<<free_space<<"\t\t TOTAL:
            "<<max_size;
        cout<<"\n\n Directory Structure : \n";

        size_hole = cont_directory(directory);

        cout<<"\n\n 1. Create a file ";
        cout<<"\n 2: Delete a file ";
        cout<<"\n 3: Defragment ";
        cout<<"\n 0: Return \t\t";
        cin >> choice;

        switch(choice){

```

```

        case 1:
            int size;
            cout<<"\n Enter the size of the file: ";
            cin>>size;
            cont_create(++file_count,directory,size,free_space);
            break;

        case 2:
            int number;
            cout<<"\n Enter the file number to delete: ";
            cin>>number;
            cont_delete(number,directory,free_space);
            break;

        case 3: size_hole = cont_defragment(directory,
            size_hole);
    }
}

map_link linked_create(int file_no, map_link &directory, int
    size,int &free_space,set<int>&pointers,const int & max_size){

    if(size>=free_space){
        cout<<"\n File too large.";
        return directory;
    }

    list<int> file_pointers;
    while(file_pointers.size()!=size){
        int temp_index = rand()%max_size;
        if (pointers.find(temp_index)==pointers.end()){
            pointers.insert(temp_index);
            file_pointers.push_back(temp_index);
        }
    }

    directory.insert(make_pair(file_no,file_pointers));
    free_space -= size;
    return directory;
}

```

```

}

map_index index_create(int file_no, map_index &directory, int
    size,int &free_space, set<int>&pointers,const int & max_size){

    //Accomodate size of the index block pointer
    if(size+1 >=free_space){
        cout<<"\n File too large.";
        return directory;
    }

    //Set a previously unused index block address
    int index = rand()%max_size;
    while (pointers.find(index)!=pointers.end())
        index = rand()%max_size;

    pointers.insert(index);

    //Now set the actual file blocks
    vector<int> file_pointers;

    while(file_pointers.size()!=size){
        int temp_index = rand()%max_size;
        if (pointers.find(temp_index)==pointers.end()){
            pointers.insert(temp_index);
            file_pointers.push_back(temp_index);
        }
    }

    directory.insert(make_pair(file_no,make_pair(index,file_pointers)
        ) );
    free_space -= size + 1;
    return directory;
}

map_index index_delete(int file,map_index &directory,int &
    free_space,set<int>&pointers){

    if(directory.find(file)==directory.end()){
        cout<<"\n No such file exists";
    }
}

```

```

        return directory;
    }

    pair<int, vector<int> > index_block =
        directory.find(file)->second;
    vector<int>:: iterator it = index_block.second.begin();

    free_space += index_block.second.size() +1 ;

    // Delete all the blocks in the index block referred by the
    // index block
    for(; it!=index_block.second.end(); it++)
        pointers.erase(*it);

    //Delete the address of the index block itself as well as the
    // map entry
    pointers.erase(index_block.first);
    directory.erase(directory.find(file));

    return directory;
}

int main(){

    auto int mode;
    while(1) {

        clear;
        cout<<"\n Specify file allocation mode: \n";
        cout<<"\n 1: Contiguous ";
        cout<<"\n 2: Linked ";
        cout<<"\n 3: Indexed ";
        cout<<"\n 0: Exit \t";
        cin >> mode;

        switch (mode){

            case 1: contiguous_mode(); break;
            case 2: linked_mode(); break;
            case 3: indexed_mode(); break;

```

```
        case 0: exit(0);          break;
        default: main();
    }
}
```

OUTPUT(Contiguous):

```
Specify file allocation mode:

1: Contiguous
2: Linked
3: Indexed
0: Exit      1

FREE(Actual): 38          TOTAL: 96

Directory Structure :

Name : 1      Location: (0,12)      Valid: 1
Name : 2      Location: (12,46)     Valid: 1
Name : 3      Location: (46,58)     Valid: 1

Hole Size: 0

1. Create a file
2. Delete a file
3. Defragment
0: Return      
```

```

FREE(Actual): 38                TOTAL: 96

Directory Structure :

Name : 1      Location: (0,12)    Valid: 1
Name : 2      Location: (12,46)   Valid: 1
Name : 3      Location: (46,58)   Valid: 1

Hole Size: 0

1. Create a file
2: Delete a file
3: Defragment
0: Return      2

Enter the file number to delete: 2

```

```

FREE(Actual): 72                TOTAL: 96

Directory Structure :

Name : 1      Location: (0,12)    Valid: 1
Name : 2      Location: (12,46)   Valid: 0
Name : 3      Location: (46,58)   Valid: 1

Hole Size: 34

1. Create a file
2: Delete a file
3: Defragment
0: Return      1

Enter the size of the file: 6

```

```

FREE(Actual): 66          TOTAL: 96

Directory Structure :

Name : 1      Location: (0,12)      Valid: 1
Name : 2      Location: (12,18)     Valid: 1
Name : 3      Location: (46,58)     Valid: 1

Hole Size: 28

1. Create a file
2: Delete a file
3: Defragment
0: Return

```

OUTPUT(Linked):

```

FREE(Actual): 40          TOTAL: 40

Directory Structure :

1. Create a file
2: Delete a file
0: Return          1

Enter the size of the file: 13

FREE(Actual): 8          TOTAL: 40

Directory Structure :

Name : 1      Location: 23->6->17->35->33->15->26->12->9->21->2->27->10->
Name : 2      Location: 19->3->20->16->11->8->7->29->22->18->13->
Name : 3      Location: 39->24->38->4->36->1->25->5->

1. Create a file
2: Delete a file
0: Return          2

Enter the file number to delete: 2

```

```

FREE(Actual): 19          TOTAL: 40

Directory Structure :

Name : 1          Location: 23->6->17->35->33->15->26->12->9->21->2->27->10->
Name : 3          Location: 39->24->38->4->36->1->25->5->

1. Create a file
2: Delete a file
0: Return

```

OUTPUT(Indexed):

```

FREE(Actual): 56          TOTAL: 56

Directory Structure :

1. Create a file
2: Delete a file
0: Return          1

Enter the size of the file: 12

```

```

FREE(Actual): 43          TOTAL: 56

Directory Structure :

Name : 1          Block Address: 15          Blocks 46->9->19->1->31->10->44->45->2->34->11->27->
1. Create a file
2: Delete a file
0: Return

```

```

FREE(Actual): 23          TOTAL: 56

Directory Structure :

Name : 1          Block Address: 15          Blocks 46->9->19->1->31->10->44->45->2->34->11->27->
Name : 2          Block Address: 14          Blocks 52->50->16->3->0->47->37->
Name : 3          Block Address: 30          Blocks 26->54->43->33->42->38->23->51->29->53->5->

1. Create a file
2: Delete a file
0: Return          2

Enter the file number to delete: 2

```

```

FREE(Actual): 31          TOTAL: 56

Directory Structure :

Name : 1          Block Address: 15          Blocks 46->9->19->1->31->10->44->45->2->34->11->27->
Name : 3          Block Address: 30          Blocks 26->54->43->33->42->38->23->51->29->53->5->

1. Create a file
2: Delete a file
0: Return

```


Experiment - 15

To configure the DHCP server and DHCP client

AIM: To set up DHCP server & to automate allocation of IP addresses.

PROCEDURE:

1. Install dhcp3-server; execute the following commands, and follow the prompts:

- aptitude install dhcp3-server



2. When asked you what connection you want to run the DHCP server on, use 'eth0'.

3. Open your DHCP server's configuration as root, using your text editor.

```
sudo gedit /etc/dhcp3/dhcpd.conf
```

4. Write the following code into dhcpd.conf. Create a backup too.

```
subnet 192.168.1.0 netmask 255.255.255.0 {  
    range 192.168.1.1 192.168.1.15;  
    option domain-name "fox.net";  
    option domain-name-servers 208.67.222.222, 208.67.220.220;  
    option broadcast-address 192.168.1.255;  
    option routers 192.168.1.1;  
    option subnet-mask 255.255.255.0; }
```

Experiment – 16

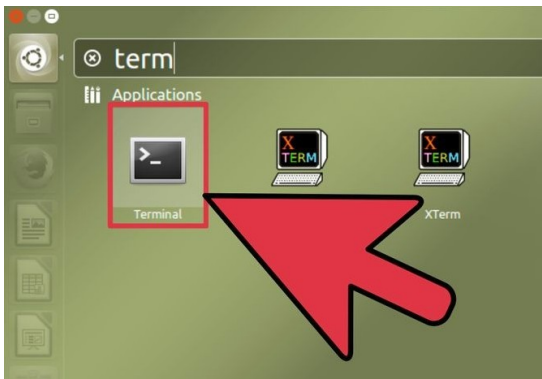
To configure the NFS server and NFS client

AIM: Write steps to set up NFS server and client and share files & directories between ubuntu LINUX operating systems.

PROCEDURE:-

NFS (Network File System) is used to share files between Linux computers on a local network. When sharing files with NFS, there are two sides: the server and the clients. The server is the computer that is actually storing the files, while the clients are the computers that are accessing the shared folder by mounting the shared folder as a virtual drive.

Open the terminal on the server computer. This is the computer that will be hosting the shared files. The server computer will need to be turned on and logged in in order for clients to mount the shared folder. NFS requires using the terminal to install and configure both the server and client.



Type:

```
sudo apt-get install nfs-kernel-server nfs-common portmap
```

This will begin downloading and installing the NFS files on your computer.

```
wiki@wiki:~$ sudo apt-get install nfs-kernel-server nfs
-common portmap
[sudo] password for wiki:
```

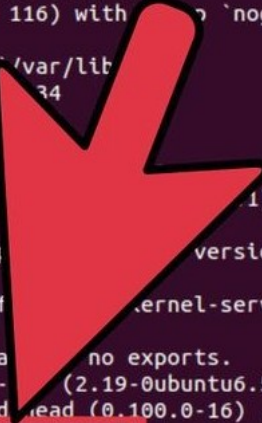


After installation, type,
dpkg-reconfigure portmap.

Select "No" from the menu that appears. This will enable other computers on the network to connect to your shared folder.

```
wiki@wiki: ~
ersion
Adding system user `statd' (UID 116) ...
Adding new user `statd' (UID 116) with shell /usr/sbin/nologin `nogroup'
...
Not creating home directory /var/lib
statd start/running, process 34
gssd stop/pre-start, process
idmapd start/running, proces
Processing triggers for urea
Setting up nfs-kernel-server (1.3.0-1) ...

Creating config file /etc/exports with new version
Creating config file /etc/default/nfs-kernel-server with
h new version
* Not starting NFS kernel daemon: no exports.
Processing triggers for libc-bin (2.19-0ubuntu6.5) ...
Processing triggers for ureadhead (0.100.0-16) ...
wiki@wiki:~$ dpkg-reconfigure portmap
```



Type ,
sudo /etc/init.d/portmap restart
to restart the portmap service. This will ensure that your changes
take effect.

```
wiki@wiki:~$ sudo /etc/init.d/portmap restart
[sudo] password for wiki:
```



Make a dummy directory that will be used to share the data. This is an empty directory that will direct the clients to the actual shared directory. This will allow you to change the shared directory on your server later without having to make any changes to the clients.

```
mkdir -p /export/dummysname
```



```
wiki@wiki:~$ sudo mkdir -p /export/dummysname
[sudo] password for wiki:
wiki@wiki:~$
```

Open the `./etc/exports` file. You will need to add your dummy directory as well as the IPs that are allowed to access it to this file. Use the following format to share with all the IP addresses on your local



```
wiki@wiki:~$ sudo gedit /etc/exports
[sudo] password for wiki:
```

Use the `.sudo /etc/init.d/nfs-kernel-server restart` command to restart the NFS server.


```
wiki@wiki:~$ sudo /etc/init.d/nfs-kernel-server restart
[sudo] password for wiki:
* Stopping NFS kernel daemon:                               [ OK ]
* Unexporting directories for NFS kernel daemon...          [ OK ]
* Not starting NFS kernel daemon: no exports.
wiki@wiki:~$
```



Client Computer

`sudo apt-get install portmap nfs-common`

```
wiki@wiki:~$ sudo apt-get install portmap nfs-common
[sudo] password for wiki:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'rpcbind' instead of 'portmap'
nfs-common is already the newest version.
rpcbind is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 316 not upgraded.
wiki@wiki:~$
```



Create the directory that the shared files will be mounted in

```
wiki@wiki:~$ sudo mkdir /sharedFile
[sudo] password for wiki:
wiki@wiki:~$
```



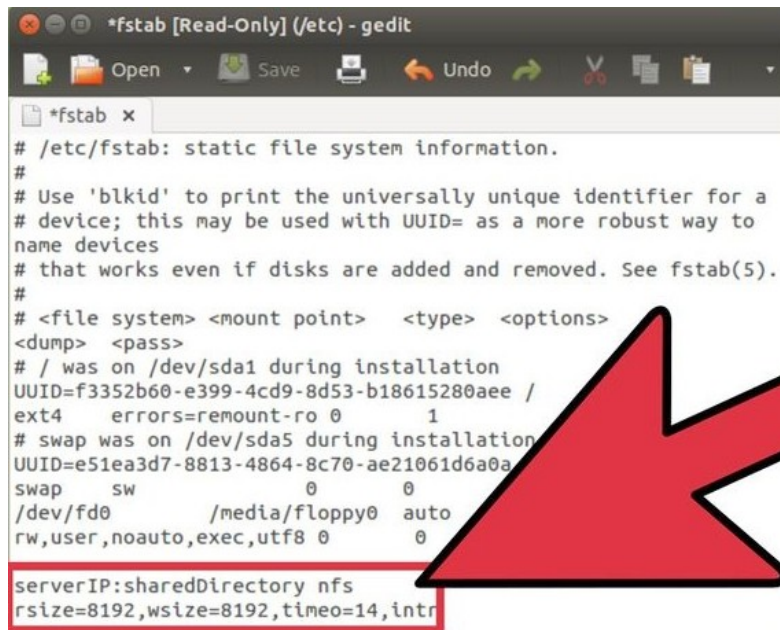
Add

```
serverIP:sharedDirectory nfs  
size=8192,ws=8192,timeo=14,intr.
```

Replace *serverIP* with the IP address of the NFS server computer.

Replace *sharedDirectory* with the dummy directory

- Using the above examples, the line might look like:
192.168.1.5:/export/Shared
/sharedFilesnfs=8192,ws=8192,timeo=14,intr

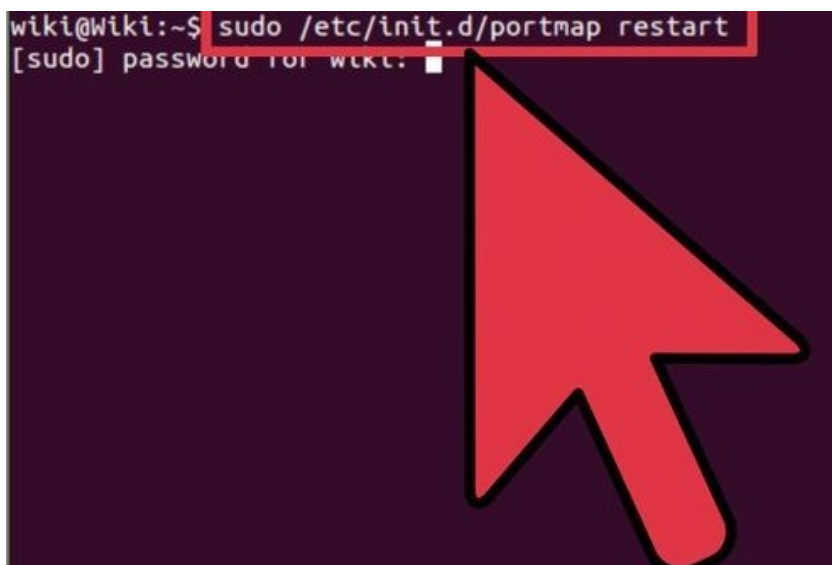


```
*fstab [Read-Only] (/etc) - gedit  
# /etc/fstab: static file system information.  
#  
# Use 'blkid' to print the universally unique identifier for a  
# device; this may be used with UUID= as a more robust way to  
# name devices  
# that works even if disks are added and removed. See fstab(5).  
#  
# <file system> <mount point> <type> <options>  
# <dump> <pass>  
# / was on /dev/sda1 during installation  
UUID=f3352b60-e399-4cd9-8d53-b18615280aee /  
ext4 errors=remount-ro 0 1  
# swap was on /dev/sda5 during installation  
UUID=e51ea3d7-8813-4864-8c70-ae21061d6a0a  
swap sw 0 0  
/dev/fd0 /media/floppy0 auto  
rw,user,noauto,exec,utf8 0 0  
  
serverIP:sharedDirectory nfs  
size=8192,ws=8192,timeo=14,intr
```

Type

```
sudo /etc/init.d/portmap restart
```

to restart portmap and use the new settings. The drive will automatically mount each time the computer reboots.



```
wiki@wiki:~$ sudo /etc/init.d/portmap restart  
[sudo] password for wkt:
```


Experiment - 17

To configure the Samba Server to share files

- 1) To get Samba open terminal and run:

sudo apt-get install samba

```
beau@beau-desktop:~$ sudo apt-get install samba
[sudo] password for beau:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.28-11 nvidia-kernel-common linux-headers-2.6.28-11-generic libgtkglext1
Use 'apt-get autoremove' to remove them.
Suggested packages:
  openssh-server inet-superserver smbldap-tools ldb-tools
The following NEW packages will be installed:
  samba
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 4527kB of archives.
After this operation, 12.7MB of additional disk space will be used.
Get:1 http://au.archive.ubuntu.com jaunty/main samba 2:3.3.2-lubuntu3 [4527kB]
Fetched 4527kB in 28s (161kB/s)
Preconfiguring packages ...
Selecting previously deselected package samba.
(Reading database ... 132980 files and directories currently installed.)
Unpacking samba (from .../samba_2:3.3.2-lubuntu3_i386.deb) ...
Processing triggers for man-db ...
Processing triggers for ufw ...
Setting up samba (2:3.3.2-lubuntu3) ...
Generating /etc/default/samba...
tdbsam_open: Converting version 0 database to version 4.
account_policy_get: tdb_fetch_uint32 failed for field 1 (min password length), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 2 (password history), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 3 (user must logon to change password), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 4 (maximum password age), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 5 (minimum password age), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 6 (lockout duration), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 7 (reset count minutes), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 8 (bad lockout attempt), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 9 (disconnect time), returning 0
account_policy_get: tdb_fetch_uint32 failed for field 10 (refuse machine password change), returning 0
Importing account for nobody...ok
Importing account for beau...ok
Adding group 'sambashare' (GID 125) ...
Done.
Adding user 'beau' to group 'sambashare' ...
Adding user beau to group sambashare
Done.
* Starting Samba daemons
```

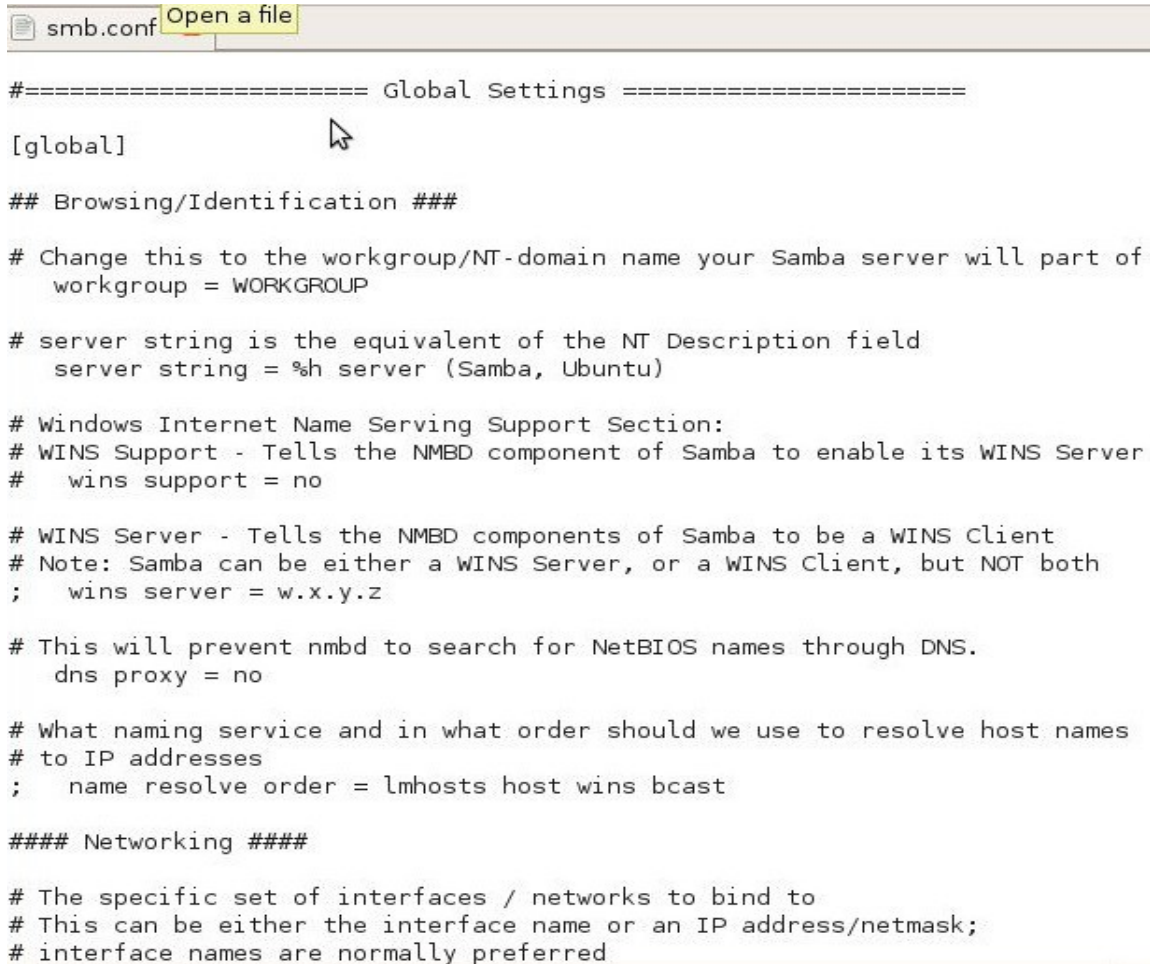
[OK]

- 2) To open and edit Samba's configuration file.

sudo gedit /etc/samba/smb.conf

- 3) Change WORKGROUP to any workgroup name.
Add netbios name = server and replace server with servers name
- 4) Scroll down to Share Definitions in the .conf file. Change yes next to read-only to no if you want to be able to write to that drive
- 5) If you want to add more drives just repeat those options e.g.
[public]
comment = Data

path = /export
force user = thermoelectric
force group = users
read only = No
Path is where that shared drive is located.



```
#===== Global Settings =====  
[global]  
  
## Browsing/Identification ###  
  
# Change this to the workgroup/NT-domain name your Samba server will part of  
workgroup = WORKGROUP  
  
# server string is the equivalent of the NT Description field  
server string = %h server (Samba, Ubuntu)  
  
# Windows Internet Name Serving Support Section:  
# WINS Support - Tells the NMBD component of Samba to enable its WINS Server  
# wins support = no  
  
# WINS Server - Tells the NMBD components of Samba to be a WINS Client  
# Note: Samba can be either a WINS Server, or a WINS Client, but NOT both  
; wins server = w.x.y.z  
  
# This will prevent nmbd to search for NetBIOS names through DNS.  
dns proxy = no  
  
# What naming service and in what order should we use to resolve host names  
# to IP addresses  
; name resolve order = lmhosts host wins bcast  
  
#### Networking ####  
  
# The specific set of interfaces / networks to bind to  
# This can be either the interface name or an IP address/netmask;  
# interface names are normally preferred
```

- 6) Add users to Ubuntu by typing this into terminal e.g
sudo useradd -c "Thermoelectric Rules" -m -g users -p password Thermoelectric
- 7) You replace password with that users password. You replace Thermoelectric Rules with your real name. You replace Thermoelectric with your user name.
- 8) Repeat that until you have made a account for all of your users
- 9) Then add the users to Samba by typing this into terminal e.g
sudo smbpasswd -a Thermoelectric


```
smb.conf

#===== Share Definitions =====

# Un-comment the following (and tweak the other settings below to suit)
# to enable the default home directory shares. This will share each
# user's home directory as \\server\username
[homes]
;   comment = Home Directories
;   browseable = no

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
;   read only = yes

# File creation mask is set to 0700 for security reasons. If you want to
# create files with group=rw permissions, set next parameter to 0775.
;   create mask = 0700

# Directory creation mask is set to 0700 for security reasons. If you want to
# create dirs. with group=rw permissions, set next parameter to 0775.
;   directory mask = 0700

# By default, \\server\username shares can be connected to by anyone
# with access to the samba server. Un-comment the following parameter
# to make sure that only "username" can connect to \\server\username
# This might need tweaking when using external authentication schemes
;   valid users = %S

# Un-comment the following and create the netlogon directory for Domain Logons
# (you need to configure Samba to act as a domain controller too.)
[netlogon]
;   comment = Network Logon Service
;   path = /home/samba/netlogon
```

- 10) Start Samba by executing this in terminal
sudo nmbd; smbd;
- 11) Configure the /export directory:
sudo mkdir /export
sudo chown Thermoelectric.users /export
sudo chmod u=rwx,g=rwx,o=rwx /export
- 12) Check that Samba is running correctly:
sudo smbclient -L localhost -U%
- 13) Connect to SERVER (netbios name) as Thermoelectric (your user name):
sudo smbclient //SERVER/Thermoelectric -UThermoelectric%password

```
beau@beau-desktop:~$ sudo useradd -c "thermoelectric Rules" -m -g users -p passw
ord Thermoelectric
[sudo] password for beau:
beau@beau-desktop:~$ sudo smbpasswd -a Thermoelectric
New SMB password:
Retype new SMB password:
Added user Thermoelectric.
beau@beau-desktop:~$
```

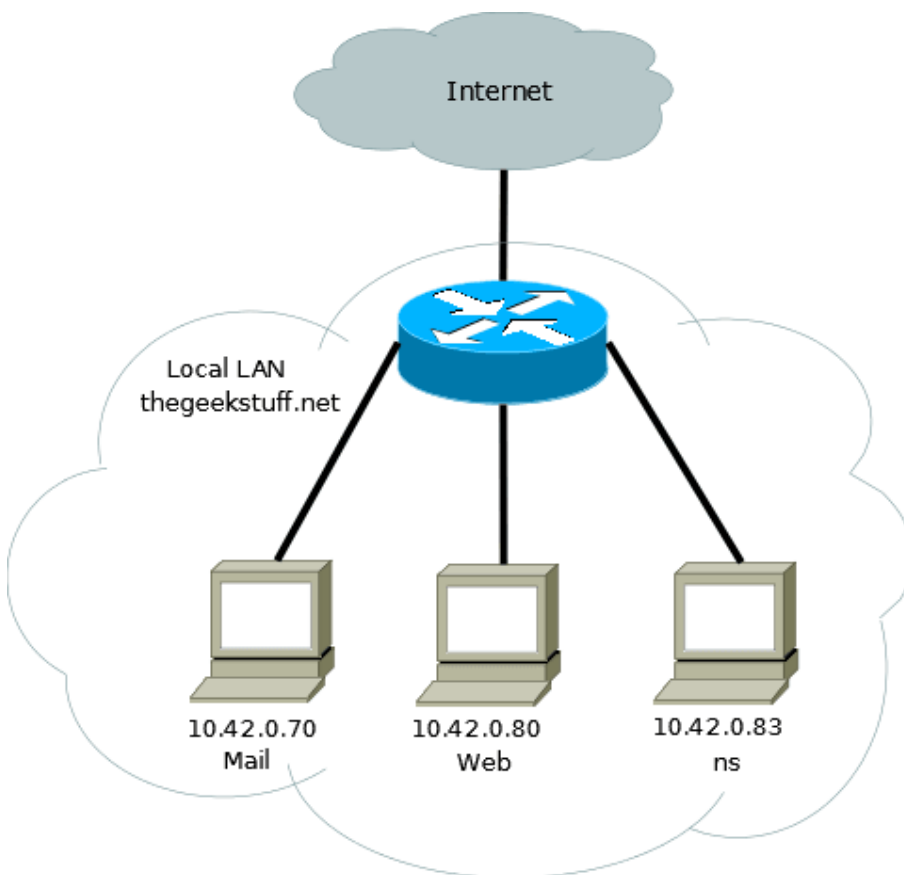
Experiment - 18

To configure and set up the DNS Server

Domain Name Service (DNS) is an internet service that maps IP addresses to fully qualified domain names (FQDN) and vice versa.

BIND stands for Berkley Internet Naming Daemon. BIND is the most common program used for maintaining a name server on Linux.

In this tutorial, we will explain how to install and configure a DNS server. The DNS chosen is on “10.42.0.83”.



1). Install Bind

```
$ sudo apt-get install bind9
```

All the DNS configurations are stored under /etc/bind directory. The primary configuration is /etc/bind/named.conf which will include other needed files. The file named /etc/bind/db.root describes the root nameservers in the world.

2). Configure Cache NameServer

The job of a DNS caching server is to query other DNS servers and cache the response. Next time when the same query is given, it will provide the response from the cache. The cache will be updated periodically. To configure a Cache NameServer, add your ISP (Internet Service Provider)'s DNS server or any OpenDNS server to the file /etc/bind/named.conf.options

```
Forwarders {  
  
    8.8.8.8;        8.8.4.4;  
  
};
```

After the above change, restart the DNS server.

```
$ sudo service bind9 restart
```

3) Test the Cache NameServer

```
$ dig ubuntu.com
```

4) Build the Forward Resolution for Primary/Master NameServer

Now we will add the details which is necessary for forward resolution into /etc/bind/db.dns_stuff.net.

```
$ sudo cp /etc/bind/db.local /etc/bind/db.dns_stuff.net
```

Next, edit the /etc/bind/db.dns_stuff.net and replace the following.

```
$TTL    604800

@ IN SOA ns.dns_stuff.net. lak.localhost. (

    1024      ; Serial

    604800    ; Refresh

    86400     ; Retry

    2419200   ; Expire

    604800 )  ; Negative Cache TTL

;
```

5) Build the Reverse Resolution for Primary/Master NameServer

```
$ sudo cp /etc/bind/db.127 /etc/bind/db.10

$ sudo service bind9 restart
```

6) Test the DNS server

On web.dns_stuff.net server, add the following to /etc/resolv.conf

```
nameserver 10.42.0.83

$ ping mail.dns_stuff.net
```

Appendix A

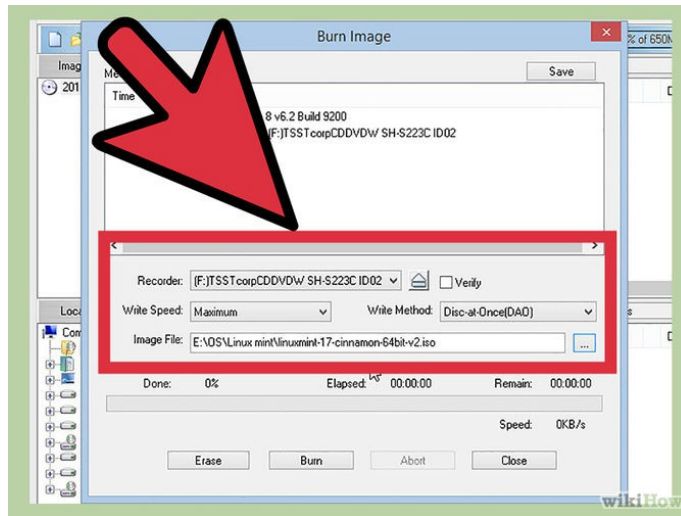
Installation of Linux Mint

1) Download the Linux Mint ISO.

An ISO file is a disk image that you can burn to a DVD.

2) Burn the image to a disc/USB

Using any disk burning tool, create a bootable disk with the image downloaded



3) Boot from the bootable disk

Specify the system to boot from the bootable in the BIOS menu

4) Start the Linux Mint live System

Select the Linux mint live system available in the menu

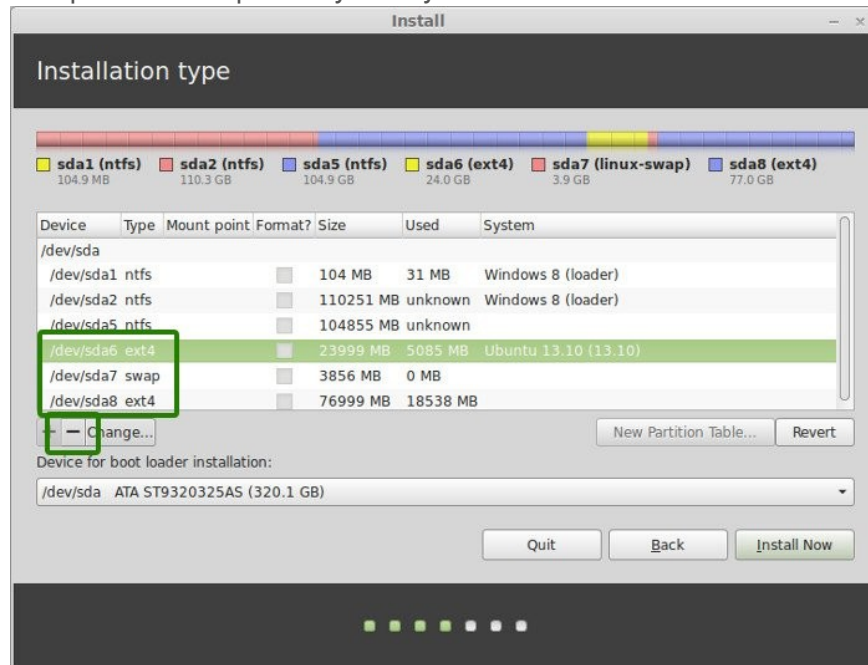


5) Start the installation

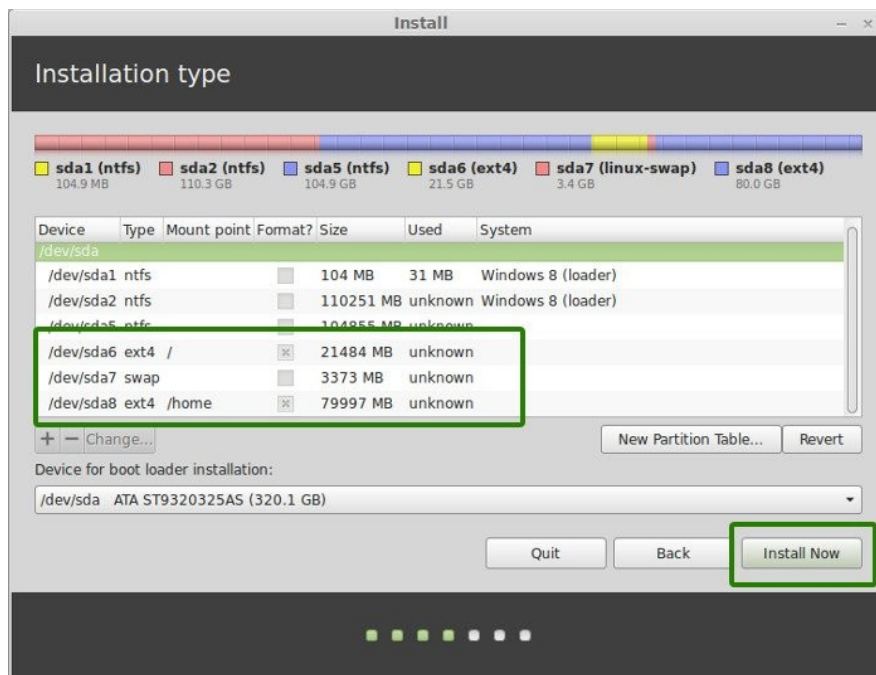
Click on 'Install Linux Mint' to continue with the installation

6) Prepare the Partitions

Set the mount points as required by the system



7) Create Root/Home/Swap Partitions



7) Create User and follow trivial instructions

Configure user details and set up a root password

8) Restart

When finished, restart the system and boot into the Linux installation

Appendix B

Basic Linux Commands

1. tar

Create a new tar archive.

```
$ tar cvf archive_name.tar dirname/
```

2. grep

Search for a given string in a file (case in-sensitive search).

```
$ grep -i "the" demo_file
```

Print the matched line, along with the 3 lines after it.

```
$ grep -A 3 -i "example" demo_text
```

3. find

Find files using file-name (case in-sensitve find)

```
# find -iname "MyCProgram.c"
```

4. diff

Compare two files line by line. Ignore white space while comparing.

```
# diff -w name_list.txt name_list_new.txt
```

5. ls

Display file size in human readable format (e.g. KB, MB etc.,)

```
$ ls -lh  
  
-rw-r----- 1 ramesh team-dev 8.9M Jun 12 15:27 arch-linux.txt.gz
```

6. ps

Used to display information about the processes that are running in the system.

```
$ ps -ef | more
```

7. rm command examples

Get confirmation before removing the file.

```
$ rm -i filename.txt
```

8. cp command examples

Copy file1 to file2 preserving the mode, ownership and timestamp.

```
$ cp -p file1 file2
```

9. cat command examples

You can view multiple files at the same time. Following example prints the content of file1 followed by file2 to stdout.

```
$ cat file1 file2
```