OS Lab Week – 8

1. **WAP program to demonstrate the use of setuid(), setgid() system calls and sticky bit.**

Ans.

AIM: A program to demonstrate the user of setuid(), setgid() and sticky bit system calls

PROCEDURE:

Setuid(): This bit is present for files which have executable permissions. The setuid bit simply indicates that when running the executable, it will set its permissions to that of the user who created it (owner), instead of setting it to the user who launched it. Similarly, there is a setgid bit which does the same for the gid.

To locate the setuid, look for an 's' instead of an 'x' in the executable bit of the file permissions**.**

Setgid(): The setgid affects both files as well as directories. When used on a file, it executes with the privileges of the group of the user who owns it instead of executing with those of the group of the user who executed it.
When the bit is set for a directory, the set of files in that directory will have the same group as the group of the parent directory, and not that of the user who created those files. This is used for file sharing since they can be now modified by all the users who are part of the group of the parent directory.

To locate the setgid bit, look for an 's' in the group section of the file permissions

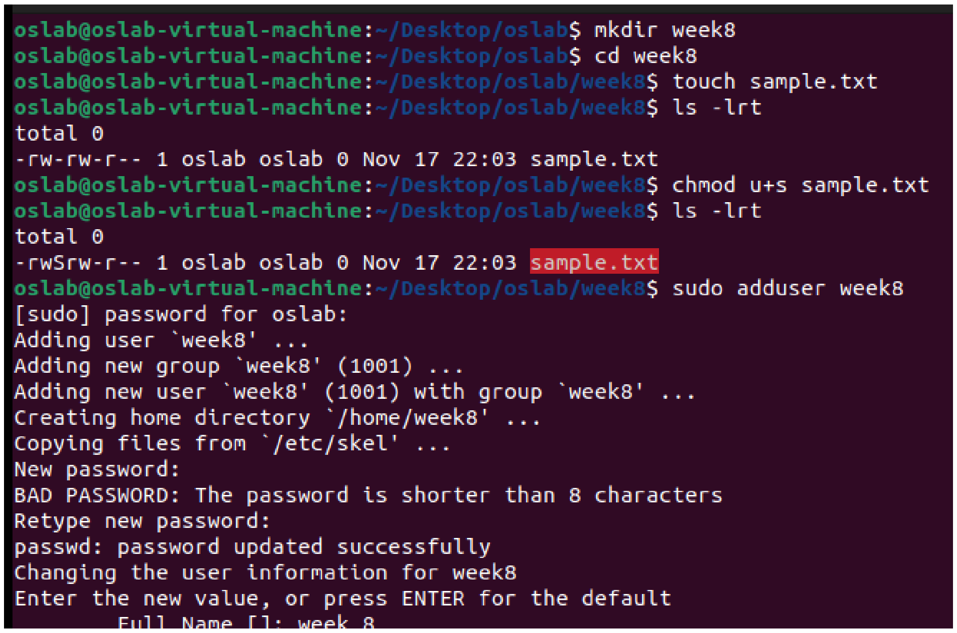
Sticky bit: The sticky bit was initially introduced to 'stick' an executable program's text segment in the swap space even after the program has completed execution, to speed up the subsequent runs of the same program. However, these days the sticky bit means something entirely different.

When a directory has the sticky bit set, its files can be deleted or renamed only by the file owner, directory owner and the root user. Simply look for a 't' character in the file permissions to locate the sticky bit. The snippet below shows how we can set the sticky bit for some directory "Gatos", and how it prevents the new user from deleting a file in the directory.

CODE:

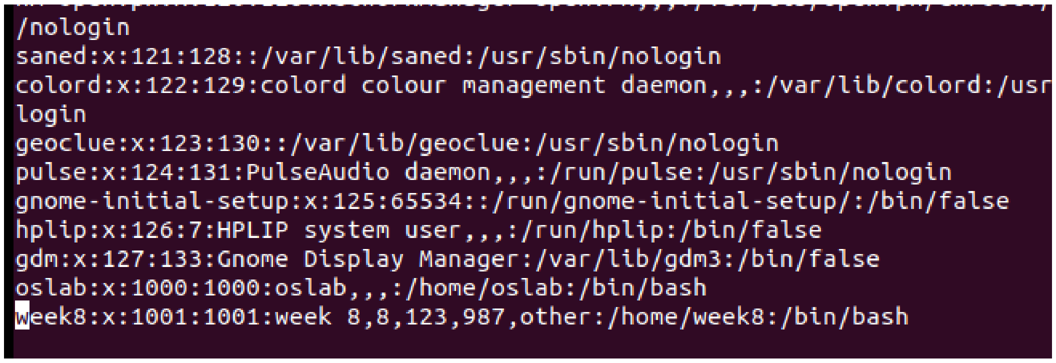
1. Setuid()

Setting a user set bit using 'chmod u+s <filename' command



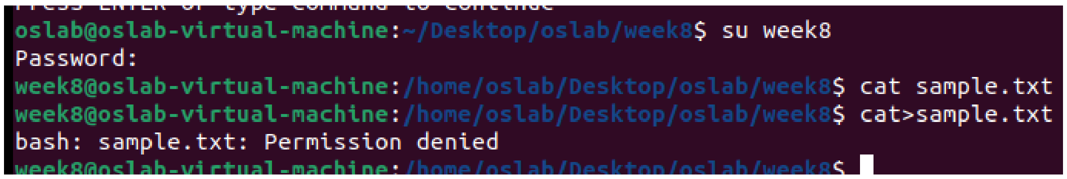
```
oslab@oslab-virtual-machine:~/Desktop/oslab$ mkdir week8
oslab@oslab-virtual-machine:~/Desktop/oslab$ cd week8
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ touch sample.txt
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ls -lrt
total 0
-rw-rw-r-- 1 oslab oslab 0 Nov 17 22:03 sample.txt
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ chmod u+s sample.txt
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ls -lrt
total 0
-rwSrw-r-- 1 oslab oslab 0 Nov 17 22:03 sample.txt
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ sudo adduser week8
[sudo] password for oslab:
Adding user `week8' ...
Adding new group `week8' (1001) ...
Adding new user `week8' (1001) with group `week8' ...
Creating home directory `/home/week8' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for week8
Enter the new value, or press ENTER for the default
        Full Name []: week 8
```

Creating a new user and checking him in /etc/passwd (in users file)



```
/nologin
saned:x:121:128::/var/lib/saned:/usr/sbin/nologin
colord:x:122:129:colord colour management daemon,,,:/var/lib/colord:/usr
login
geoclue:x:123:130::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:124:131:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:125:65534::/run/gnome-initial-setup/:/bin/false
hplip:x:126:7:HPLIP system user,,,:/run/hplip:/bin/false
gdm:x:127:133:Gnome Display Manager:/var/lib/gdm3:/bin/false
oslab:x:1000:1000:oslab,,,:/home/oslab:/bin/bash
week8:x:1001:1001:week 8,8,123,987,other:/home/week8:/bin/bash
```

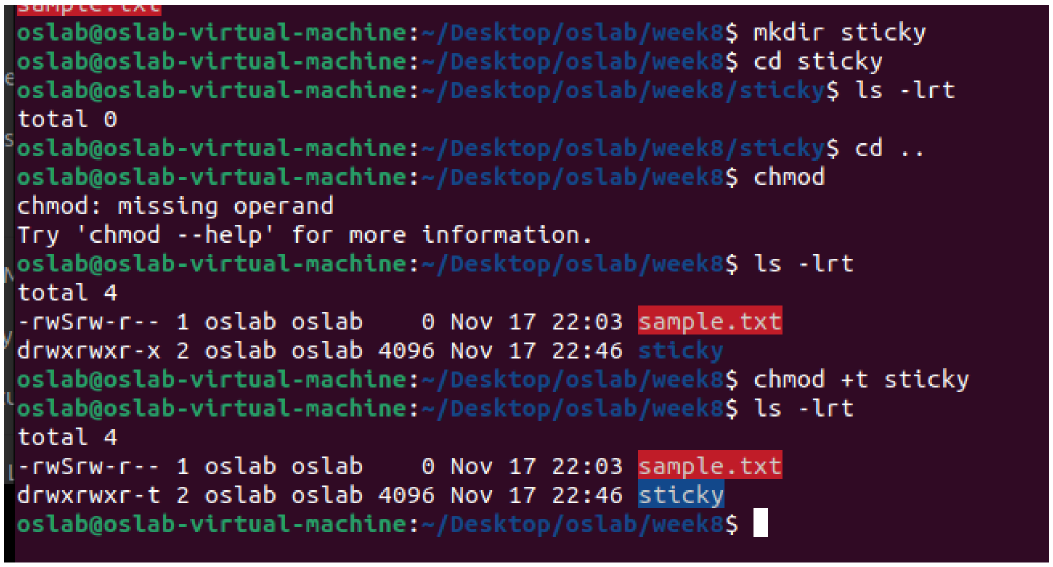
Switching to other user and trying to modify the file which has the set bit turned on



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ su week8
Password:
week8@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$ cat sample.txt
week8@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$ cat>sample.txt
bash: sample.txt: Permission denied
```

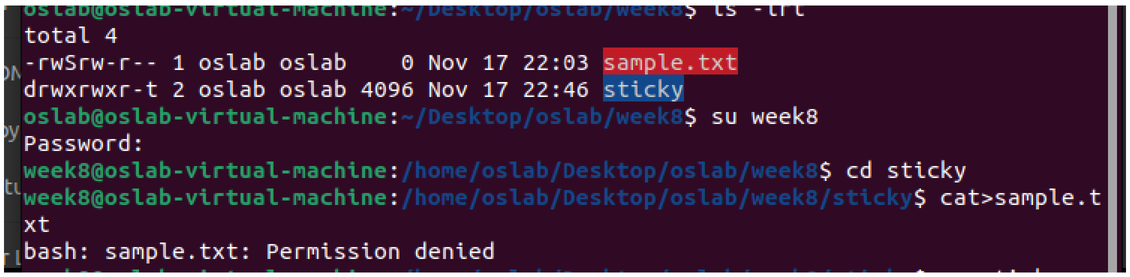
2. Sticky bit

Creating a directory and making its sticky bit on by command ‘chmod +t <filename>’



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ mkdir sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ cd sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8/sticky$ ls -lrt
total 0
oslab@oslab-virtual-machine:~/Desktop/oslab/week8/sticky$ cd ..
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ chmod
chmod: missing operand
Try 'chmod --help' for more information.
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ls -lrt
total 4
-rwSrw-r-- 1 oslab oslab    0 Nov 17 22:03 sample.txt
drwxrwxr-x 2 oslab oslab 4096 Nov 17 22:46 sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ chmod +t sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ls -lrt
total 4
-rwSrw-r-- 1 oslab oslab    0 Nov 17 22:03 sample.txt
drwxrwxr-t 2 oslab oslab 4096 Nov 17 22:46 sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$
```

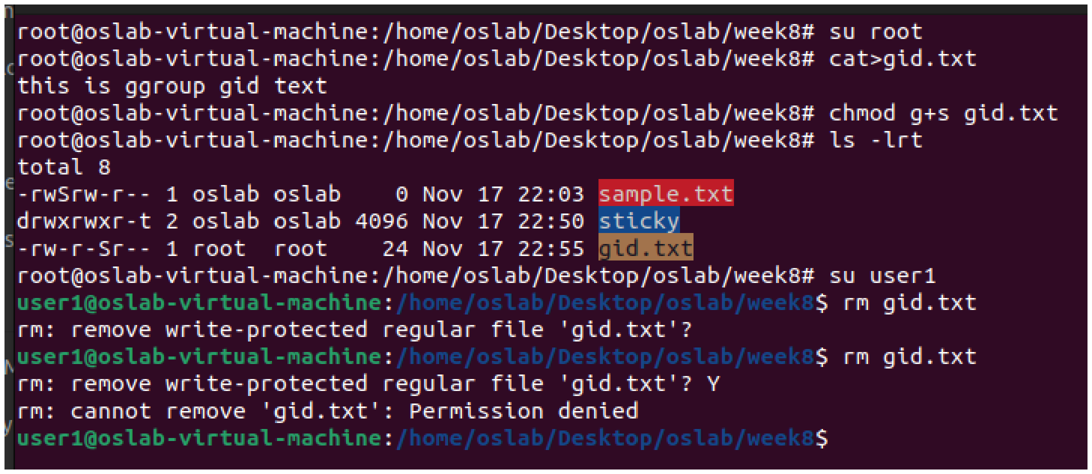
Switching to other user and modifying the files



```
total 4
-rwSrw-r-- 1 oslab oslab    0 Nov 17 22:03 sample.txt
drwxrwxr-t 2 oslab oslab 4096 Nov 17 22:46 sticky
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ su week8
Password:
week8@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$ cd sticky
week8@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8/sticky$ cat>sample.t
xt
bash: sample.txt: Permission denied
```

3. Setgid()

Setting gid using command ‘chmod g+s <filename>’ and checking to modify from other user



```
root@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8# su root
root@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8# cat>gid.txt
this is ggroup gid text
root@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8# chmod g+s gid.txt
root@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8# ls -lrt
total 8
-rwSrw-r-- 1 oslab oslab    0 Nov 17 22:03 sample.txt
drwxrwxr-t 2 oslab oslab 4096 Nov 17 22:50 sticky
-rw-r-Sr-- 1 root  root    24 Nov 17 22:55 gid.txt
root@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8# su user1
user1@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$ rm gid.txt
rm: remove write-protected regular file 'gid.txt'?
user1@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$ rm gid.txt
rm: remove write-protected regular file 'gid.txt'? Y
rm: cannot remove 'gid.txt': Permission denied
user1@oslab-virtual-machine:/home/oslab/Desktop/oslab/week8$
```

2. **WAP program to demonstrate the use of setuid(), setguid(), seteuid(), getgid(), geteuid(), getegid()**

Ans.

AIM: Program to demonstrate the use of setuid(), setgid(), seteuid(), getgid(), geteuid(), getegid()

PROCEDURE:

Setuid(): This bit is present for files which have executable permissions. The setuid bit simply indicates that when running the executable, it will set its permissions to that of the user who created it (owner), instead of setting it to the user who launched it. Similarly, there is a setgid bit which does the same for the gid.

To locate the setuid, look for an ‘s’ instead of an ‘x’ in the executable bit of the file permissions**.**

Setgid(): The setgid affects both files as well as directories. When used on a file, it executes with the privileges of the group of the user who owns it instead of executing with those of the group of the user who executed it.
When the bit is set for a directory, the set of files in that directory will have the same group as the group of the parent directory, and not that of the user who created those files. This is used for file sharing since they can be now modified by all the users who are part of the group of the parent directory.

To locate the setgid bit, look for an ‘s’ in the group section of the file permissions

Seteuid(): The process effective user ID is reset if the *UID* parameter is equal to either the current real or saved user IDs or if the effective user ID of the process is the root user.

Getgid(): The getuid subroutine returns the real user ID of the current process.

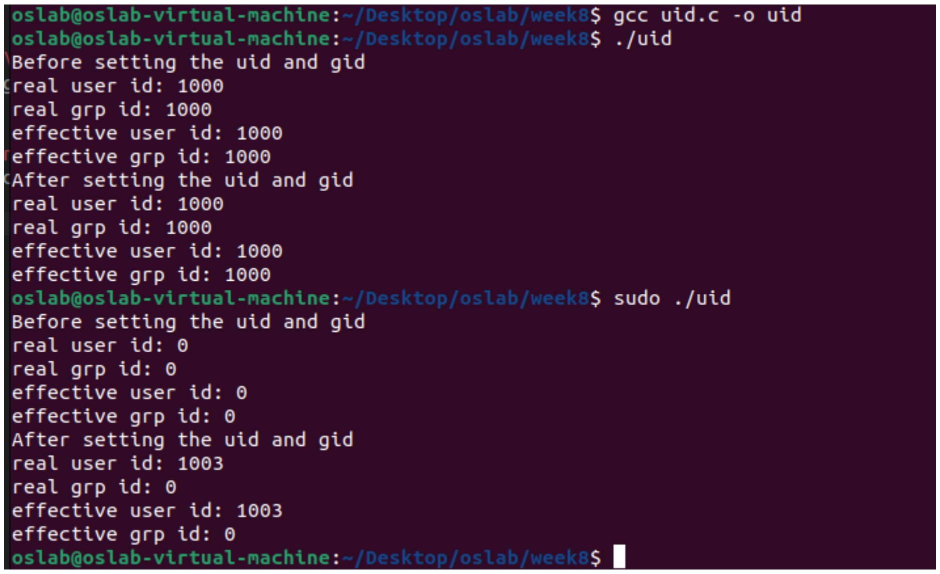
Geteuid(): The geteuid subroutine returns the effective user ID of the current process.

Getegid(): The getegid subroutine returns the effective group ID of the current process.

CODE:

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

int main()
{
        uid_t uid;
        printf("Before setting the uid and gid\nreal user id: %d\nreal grp id: 
%d\neffective user id: %d\neffective grp id: 
%d\n",getuid(),getgid(),geteuid(),getegid());
        setuid(1003);
        setgid(1002);
        printf("After setting the uid and gid\nreal user id: %d\nreal grp id: 
%d\neffective user id: %d\neffective grp id: 
%d\n",getuid(),getgid(),geteuid(),getegid());
        return 0;
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ gcc uid.c -o uid
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ./uid
Before setting the uid and gid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
After setting the uid and gid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ sudo ./uid
Before setting the uid and gid
real user id: 0
real grp id: 0
effective user id: 0
effective grp id: 0
After setting the uid and gid
real user id: 1003
real grp id: 0
effective user id: 1003
effective grp id: 0
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ 
```

3. **WAP to demonstrate the use of set setresuid(), setresgid(), setfsuid() and setfsgid()**

Ans.

AIM: A program to demonstrate the use of setresuid(), setresgid(), setfsuid() and setfsgid()

PROCEDURE:

Setresuid(): setresuid() sets the real user ID, the effective user ID, and the saved set-user-ID of the calling process. An unprivileged process may change its real UID, effective UID, and saved set-user-ID, each to one of: the current real UID, the current effective UID, or the current saved set-user-ID

Setresgid(): setresgid() sets the real GID, effective GID, and saved set-group-ID of the calling process (and always modifies the file system GID to be the same as the effective GID), with the same restrictions for unprivileged processes.

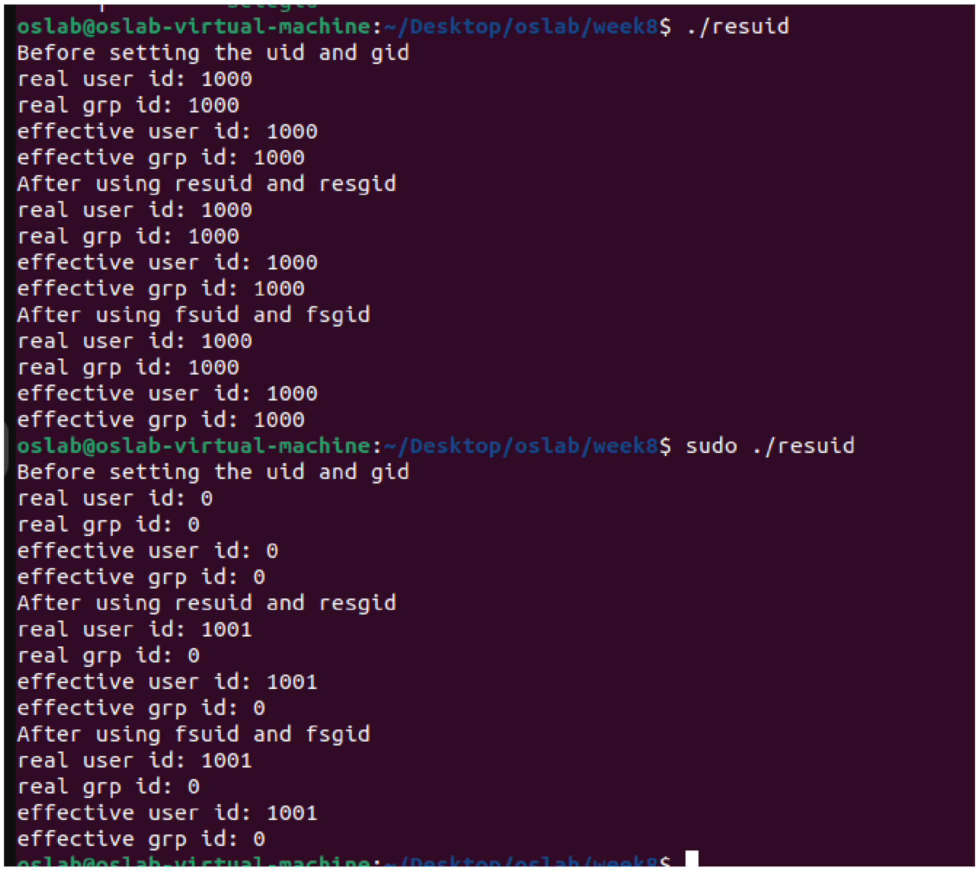
Setfsuid(): The system call setfsuid() sets the user ID that the Linux kernel uses to check for all accesses to the file system. Normally, the value of *fsuid* will shadow the value of the effective user ID. In fact, whenever the effective user ID is changed, *fsuid* will also be changed to the new value of the effective user ID.

Setfsgid(): The system call setfsgid() sets the group ID that the Linux kernel uses to check for all accesses to the file system. Normally, the value of *fsgid* will shadow the value of the effective group ID. In fact, whenever the effective group ID is changed, *fsgid* will also be changed to the new value of the effective group ID.

CODE:

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

int main()
{
        uid_t uid;
        printf("Before setting the uid and gid\nreal user id: %d\nreal grp id: 
%d\neffective user id: %d\neffective grp id: 
%d\n",getuid(),getgid(),geteuid(),getegid());
        setresuid(1001,1001,1001);
        setresgid(1002,1002,1002);
        printf("After using resuid and resgid\nreal user id: %d\nreal grp id: 
%d\neffective user id: %d\neffective grp id: 
%d\n",getuid(),getgid(),geteuid(),getegid());
        setfsuid(1001);
        setfsgid(1002);
        printf("After using fsuid and fsgid\nreal user id: %d\nreal grp id: 
%d\neffective user id: %d\neffective grp id: 
%d\n",getuid(),getgid(),geteuid(),getegid());
        return 0;
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ ./resuid
Before setting the uid and gid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
After using resuid and resgid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
After using fsuid and fsgid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$ sudo ./resuid
Before setting the uid and gid
real user id: 0
real grp id: 0
effective user id: 0
effective grp id: 0
After using resuid and resgid
real user id: 1001
real grp id: 0
effective user id: 1001
effective grp id: 0
After using fsuid and fsgid
real user id: 1001
real grp id: 0
effective user id: 1001
effective grp id: 0
oslab@oslab-virtual-machine:~/Desktop/oslab/week8$
```

4. **Write the difference between**
    a. **Effective user ID**
    b. **Effective group ID**
    c. **Real user ID**
    d. **Real group ID**
    e. **File system user ID**
    f. **File system group ID**

Ans.

Effective user ID and Effective Group ID

In Unix the effective user ID and group ID, in conjunction with the supplementary group IDs, are used to determine the permissions granted to a process when it tries to perform various operations (i.e., system calls). For example, these identifiers determine the permissions granted to a process when it accesses resources such as files and System V interprocess communication (IPC) objects, which themselves have associated user and group IDs determining to whom they belong. A process whose effective user ID is 0 (the user ID of root) has all of the privileges of the super user. Such a process is referred to as a privileged process. Certain system calls can be executed only by privileged processes

Real user ID and Real group ID

The real user ID and group ID identify the user and group to which the process belongs. As part of the login process, a login shell gets its real user and group IDs from the third and fourth fields of the user's password record in the /etc/passwd file. When a new process is created (e.g., when the shell executes a program), it inherits these identifiers from its parent.

File system user ID and File system group ID

On Linux, it is the file-system user and group IDs, rather than the effective user and group IDs, that are used (in conjunction with the supplementary group IDs) to determine permissions when performing file-system operations such as opening files, changing file ownership, and modifying file permissions. Normally, the file-system user and group IDs have the same values as the corresponding effective IDs (and thus typically are the same as the corresponding real IDs). Furthermore, whenever the effective user or group ID is changed, either by a system call or by execution of a set-user-ID or set-group-ID program, the corresponding file-system ID is also changed to the same value

OS Lab Week – 9

1. **WAP to demonstrate Paging**

Ans.

AIM: A program to demonstrate paging

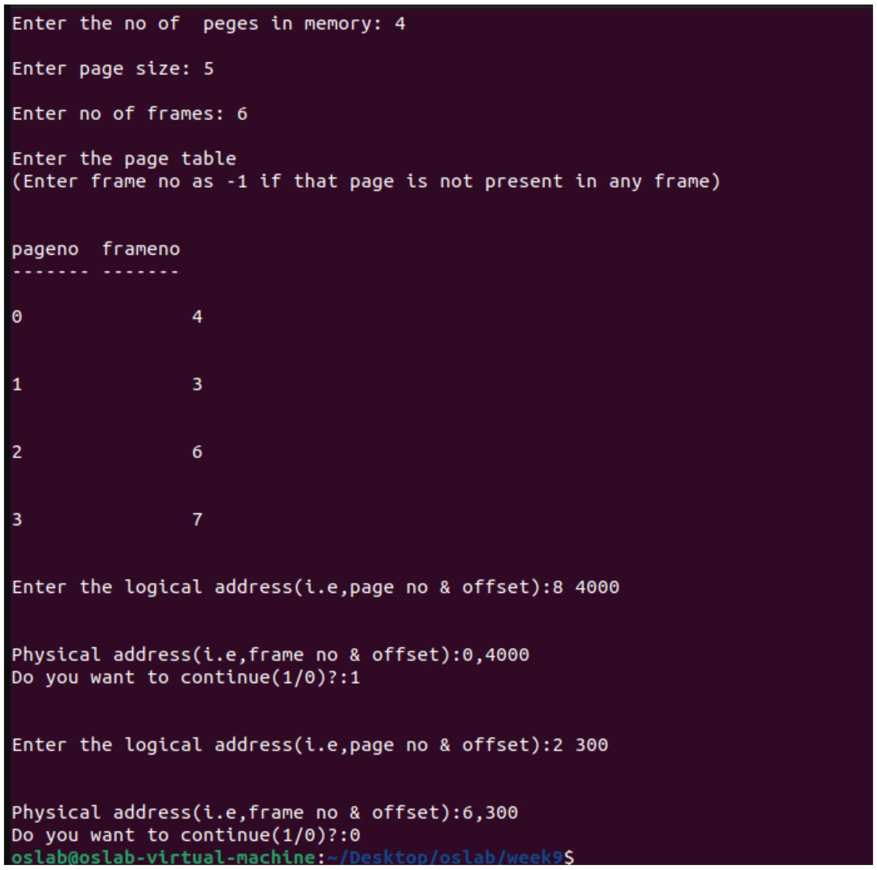
PROCEDURE:

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames. One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes. Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage. Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

CODE:

```
#include<stdio.h>

#define MAX 50

int main()

{

int page[MAX],i,n,f,ps,off,pno;

int choice=0;

printf("\nEnter the no of  peges in memory: ");

scanf("%d",&n);

printf("\nEnter page size: ");

scanf("%d",&ps);

printf("\nEnter no of frames: ");

scanf("%d",&f);

for(i=0;i<n;i++)

page[i]=-1;

printf("\nEnter the page table\n");

printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");

printf("\npageno\tframeno\n-------\t-------");

for(i=0;i<n;i++)

{

printf("\n\n%d\t\t",i);

scanf("%d",&page[i]);
}
do
{
printf("\n\nEnter the logical address(i.e,page no & offset):");
scanf("%d%d",&pno,&off);
if(page[pno]==-1)
printf("\n\nThe required page is not available in any of frames");
else
printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
return 1;
}
```

OUTPUT:



```
Enter the no of  peges in memory: 4

Enter page size: 5

Enter no of frames: 6

Enter the page table
(Enter frame no as -1 if that page is not present in any frame)


pageno  frameno
------- -------

0               4


1               3


2               6


3               7


Enter the logical address(i.e,page no & offset):8 4000


Physical address(i.e,frame no & offset):0,4000
Do you want to continue(1/0)?:1


Enter the logical address(i.e,page no & offset):2 300


Physical address(i.e,frame no & offset):6,300
Do you want to continue(1/0)?:0
oslab@oslab-virtual-machine:~/Desktop/oslab/week9$
```

2. **WAP to demonstrate segmentation**

Ans.

AIM: A program to demonstrate segmentation

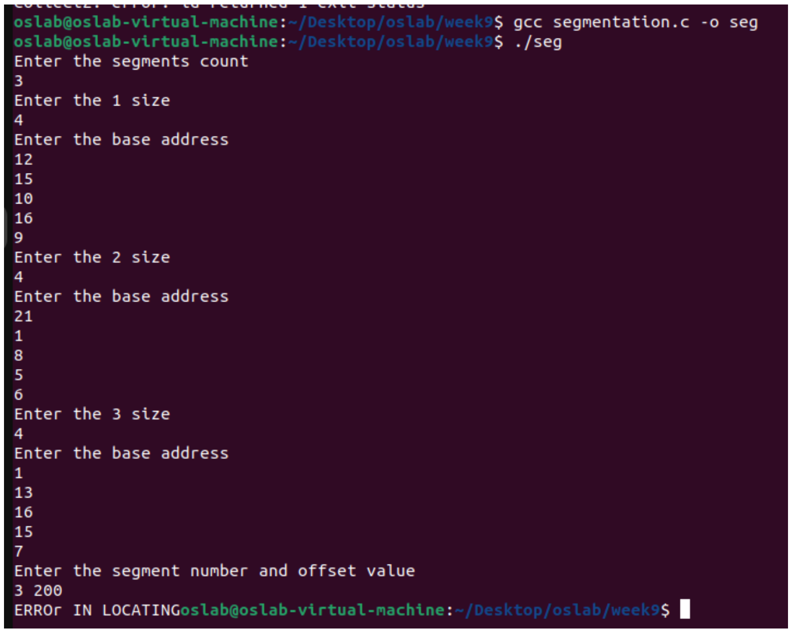
PROCEDURE:

Segmentation is a memory management technique that splits up the virtual address space of an application into chunks. By splitting the memory up into manageable chunks, the operating system can track which parts of the memory are in use and which parts are free. This makes allocating and deallocating memory much faster and simpler for the operating system. The segments are of unequal size and are not placed in a contiguous way. As it’s a non-contiguous memory allocation technique, internal fragmentation doesn’t occur. The length is decided on the base of the purpose of the segment in a user program. Segmentation is a memory management technique which divides the program from the user’s view of memory. That means the program is divided into modules/segments, unlike paging in which the program was divided into different pages, and those pages may or may not be loaded into the memory simultaneously. Segmentation prevents internal fragmentation.

CODE:

```
#include<stdio.h>

int main()

{

int a[10][10],b[100],i,j,n,x,base,size,seg,off;

printf("Enter the segments count\n");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter the %d size \n",i+1);

scanf("%d",&size);

a[i][0]=size;

printf("Enter the base address\n");

scanf("%d",&base);

a[i][1]=base;


for(j=0;j<size;j++)

        {

        x=0;

        scanf("%d",&x);

//      b[base]=x;

        base++;

        b[base]=x;

        }

}
printf("Enter the segment number and offset value \n");
scanf("%d%d",&seg,&off);
if(off<a[seg][0])
{
int abs=a[seg][1]+off;
printf("the offset is less tha %d",a[seg][0]);
printf("\n %d + %d = %d\n",a[seg][1],off,abs);
printf("the element %d is at %d ",b[abs+1],abs);
}
else
{
printf("Error in locating\n");
}
}
```

OUTPUT:



```
collect2: error: ld returned 1 exit status
oslab@oslab-virtual-machine:~/Desktop/oslab/week9$ gcc segmentation.c -o seg
oslab@oslab-virtual-machine:~/Desktop/oslab/week9$ ./seg
Enter the segments count
3
Enter the 1 size
4
Enter the base address
12
15
10
16
9
Enter the 2 size
4
Enter the base address
21
1
8
5
6
Enter the 3 size
4
Enter the base address
1
13
16
15
7
Enter the segment number and offset value
3 200
ERROr IN LOCATINGoslab@oslab-virtual-machine:~/Desktop/oslab/week9$ 
```

3. **WAP to demonstrate Banker's algorithm (Resource Allocation)**

Ans.

AIM: A program to demonstrate Banker's algorithm (Resource Allocation)

PROCEDURE:

It is a banker algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. For example, suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

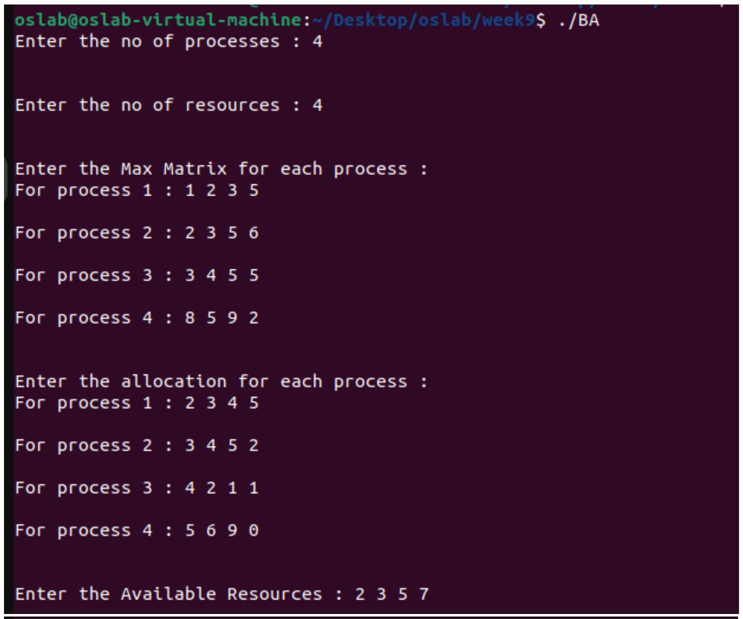
Similarly, it works in an operating system. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

CODE:

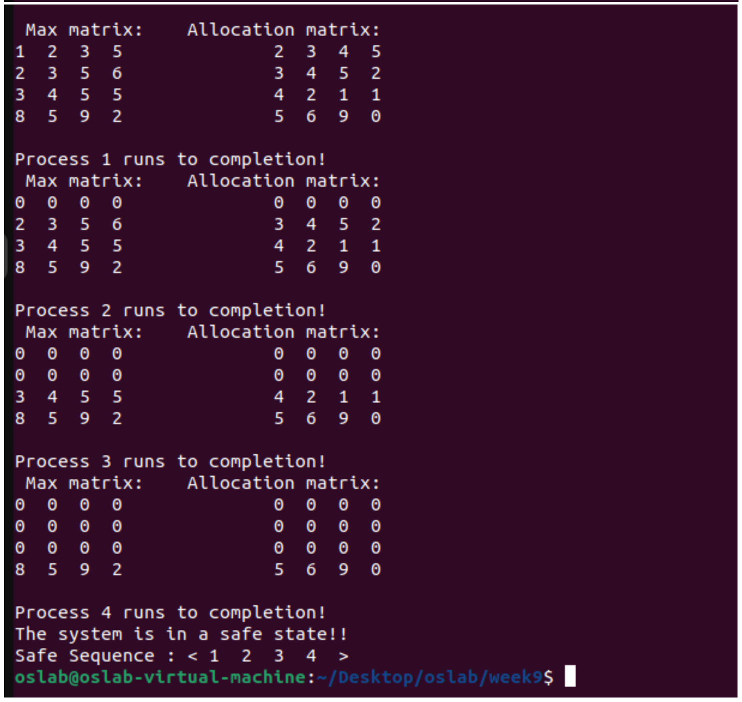
```
#include <stdio.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

for(i = 0; i< p; i++)

    completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

for(i = 0; i < p; i++)

{

    printf("\nFor process %d : ", i + 1);

    for(j = 0; j < r; j++)

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

}

printf("\n\nEnter the allocation for each process : ");

for(i = 0; i < p; i++)

{

    printf("\nFor process %d : ",i + 1);

    for(j = 0; j < r; j++)
```

```
        scanf("%d", &alloc[i][j]);
}
printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
        scanf("%d", &avail[i]);
    for(i = 0; i < p; i++)
        for(j = 0; j < r; j++)
            need[i][j] = Max[i][j] - alloc[i][j];
do
{
    printf("\n Max matrix:\tAllocation matrix:\n");
    for(i = 0; i < p; i++)
    {
        for( j = 0; j < r; j++)
            printf("%d  ", Max[i][j]);
        printf("\t\t");
        for( j = 0; j < r; j++)
            printf("%d  ", alloc[i][j]);
        printf("\n");
    }
    process = -1;
    for(i = 0; i < p; i++)
    {
        if(completed[i] == 0)//if not completed
        {
            process = i ;
            for(j = 0; j < r; j++)
            {
                if(avail[j] < need[i][j])
                {
                    process = -1;
                    break;
                }
            }
        }
        if(process != -1)
            break;
    }
    if(process != -1)
    {
        printf("\nProcess %d runs to completion!", process + 1);
        safeSequence[count] = process + 1;
        count++;
        for(j = 0; j < r; j++)
        {
            avail[j] += alloc[process][j];
            alloc[process][j] = 0;
            Max[process][j] = 0;
            completed[process] = 1;
        }
    }
}while(count != p && process != -1);
if(count == p)
{
    printf("\nThe system is in a safe state!!\n");
    printf("Safe Sequence : < ");
    for( i = 0; i < p; i++)
            printf("%d  ", safeSequence[i]);
    printf(">\n");
}
else
    printf("\nThe system is in an unsafe state!!");
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week9$ ./BA
Enter the no of processes : 4


Enter the no of resources : 4


Enter the Max Matrix for each process :
For process 1 : 1 2 3 5

For process 2 : 2 3 5 6

For process 3 : 3 4 5 5

For process 4 : 8 5 9 2


Enter the allocation for each process :
For process 1 : 2 3 4 5

For process 2 : 3 4 5 2

For process 3 : 4 2 1 1

For process 4 : 5 6 9 0


Enter the Available Resources : 2 3 5 7
```



```
 Max matrix:    Allocation matrix:
1  2  3  5             2  3  4  5
2  3  5  6             3  4  5  2
3  4  5  5             4  2  1  1
8  5  9  2             5  6  9  0

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0  0  0  0             0  0  0  0
2  3  5  6             3  4  5  2
3  4  5  5             4  2  1  1
8  5  9  2             5  6  9  0

Process 2 runs to completion!
 Max matrix:    Allocation matrix:
0  0  0  0             0  0  0  0
0  0  0  0             0  0  0  0
3  4  5  5             4  2  1  1
8  5  9  2             5  6  9  0

Process 3 runs to completion!
 Max matrix:    Allocation matrix:
0  0  0  0             0  0  0  0
0  0  0  0             0  0  0  0
0  0  0  0             0  0  0  0
8  5  9  2             5  6  9  0

Process 4 runs to completion!
The system is in a safe state!!
Safe Sequence : < 1  2  3  4  >
oslab@oslab-virtual-machine:~/Desktop/oslab/week9$
```

OS Lab Week – 10

1. **WAP to demonstrate the Linked File allocation techniques**

Ans.

AIM: A program to demonstrate the Linked File allocation techniques

PROCEDURE:

It is easy to allocate the files because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consisting of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that’s why it is said to be linked allocation. We can avoid the external fragmentation.

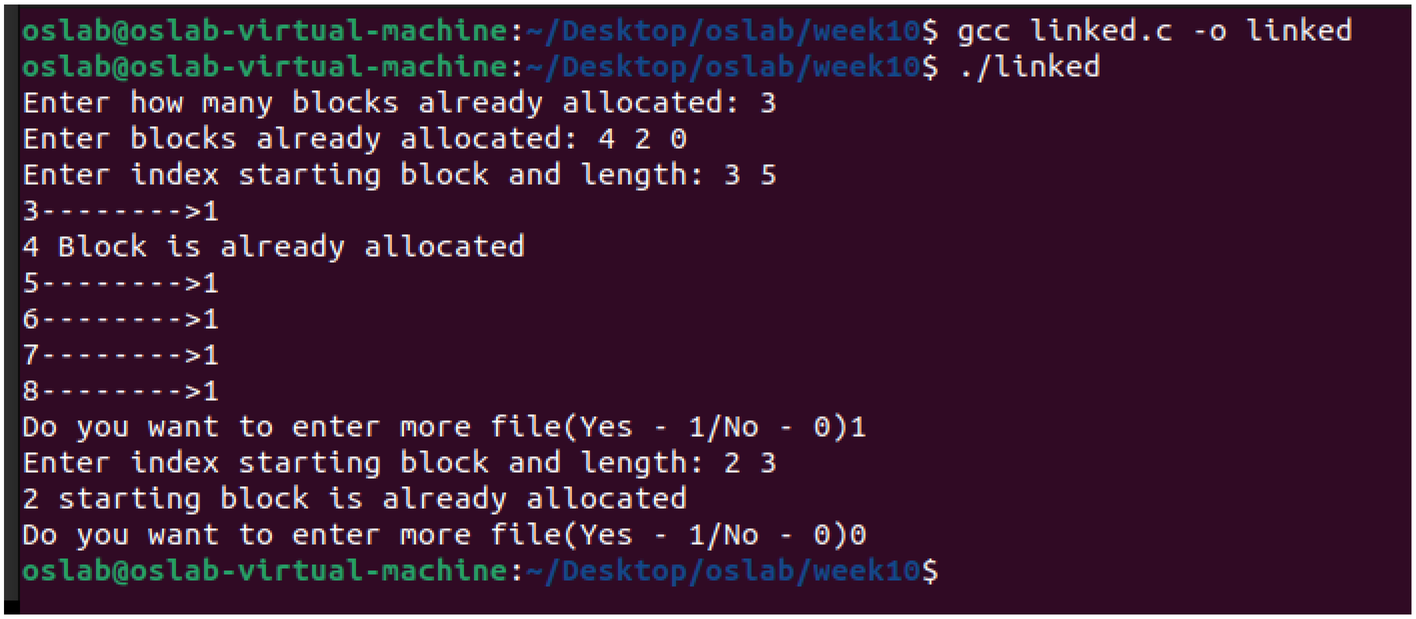
Algorithm for Linked File Allocation:

```
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page at the head of the queue
Step 3: Now the new page is inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of the stack
Step 6: Stop the allocation.
```

CODE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int f[50], p,i, st, len, j, c, k, a;
//clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
return 0;
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc linked.c -o linked
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./linked
Enter how many blocks already allocated: 3
Enter blocks already allocated: 4 2 0
Enter index starting block and length: 3 5
3-------->1
4 Block is already allocated
5-------->1
6-------->1
7-------->1
8-------->1
Do you want to enter more file(Yes - 1/No - 0)1
Enter index starting block and length: 2 3
2 starting block is already allocated
Do you want to enter more file(Yes - 1/No - 0)0
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

2. **WAP to demonstrate Indexed file allocation techniques**

Ans.

AIM: A program to demonstrate Indexed file allocation techniques

PROCEDURE:

Indexed allocation supports both sequential and direct access files. The file indexes are not physically stored as a part of the file allocation table. Whenever the file size increases, we can easily add some more blocks to the index. In this strategy, the file allocation table contains a single entry for each file. The entry consisting of one index block, the index blocks having the pointers to the other blocks. No external fragmentation.

Algorithm for Indexed File Allocation:

Step 1: Start.

Step 2: Let n be the size of the buffer

Step 3: check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the buffer

Step 6: If the buffer is full the producer has to wait

Step 7: Check there is any consumer.If yes check whether the buffer is empty

Step 8: If no the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait.

Step 10: Repeat checking for the producer and consumer till required

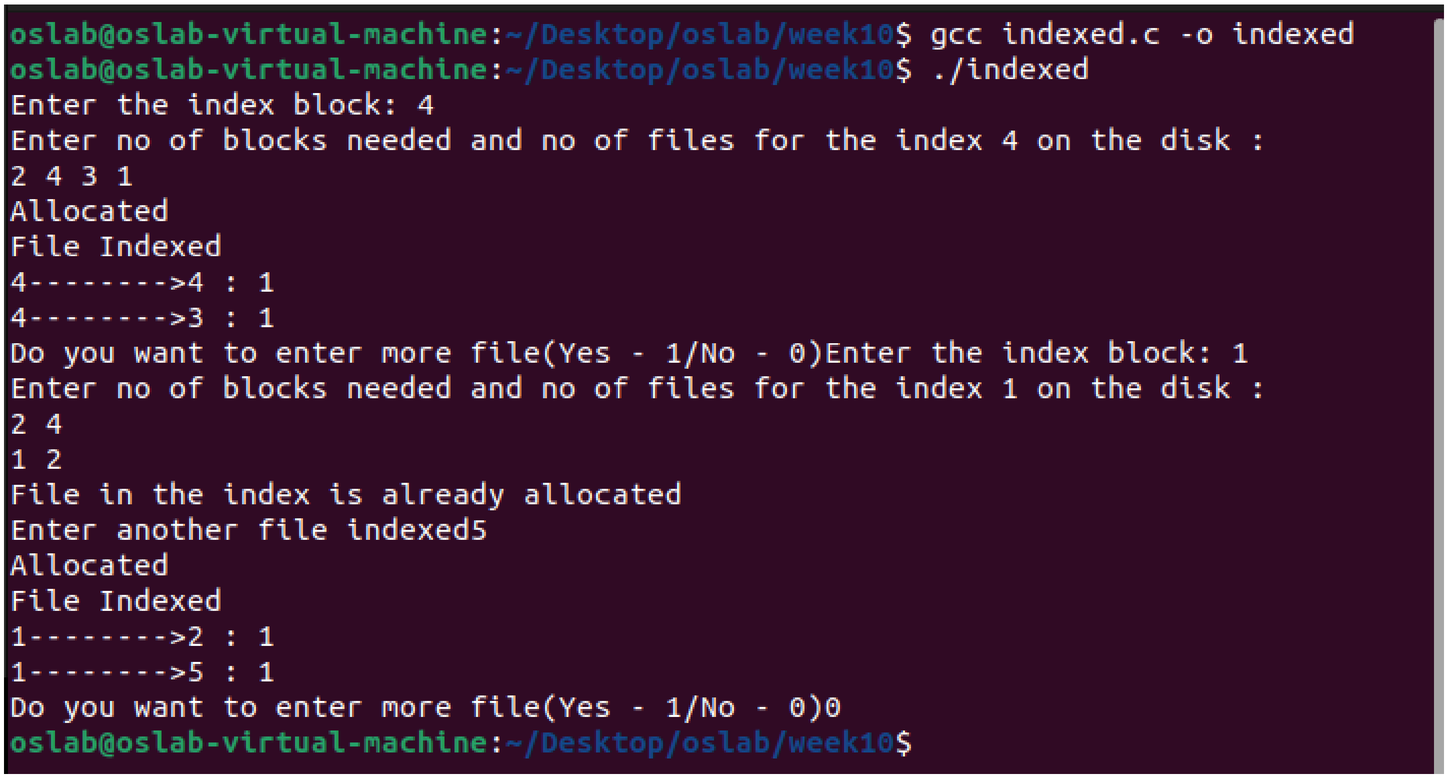
Step 11: Terminate the process.

CODE:

```
#include<stdio.h>

#include<conio.h>
```

```
#include<stdlib.h>
int main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
//clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
return 0;
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc indexed.c -o indexed
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./indexed
Enter the index block: 4
Enter no of blocks needed and no of files for the index 4 on the disk :
2 4 3 1
Allocated
File Indexed
4-------->4 : 1
4-------->3 : 1
Do you want to enter more file(Yes - 1/No - 0)Enter the index block: 1
Enter no of blocks needed and no of files for the index 1 on the disk :
2 4
1 2
File in the index is already allocated
Enter another file indexed5
Allocated
File Indexed
1-------->2 : 1
1-------->5 : 1
Do you want to enter more file(Yes - 1/No - 0)0
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

3. **WAP to demonstrate the contiguous file allocation techniques.**

Ans.

AIM: A program to demonstrate the contiguous file allocation techniques.

PROCEDURE:

In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files.

Algorithm for Sequential File Allocation:

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates it.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.

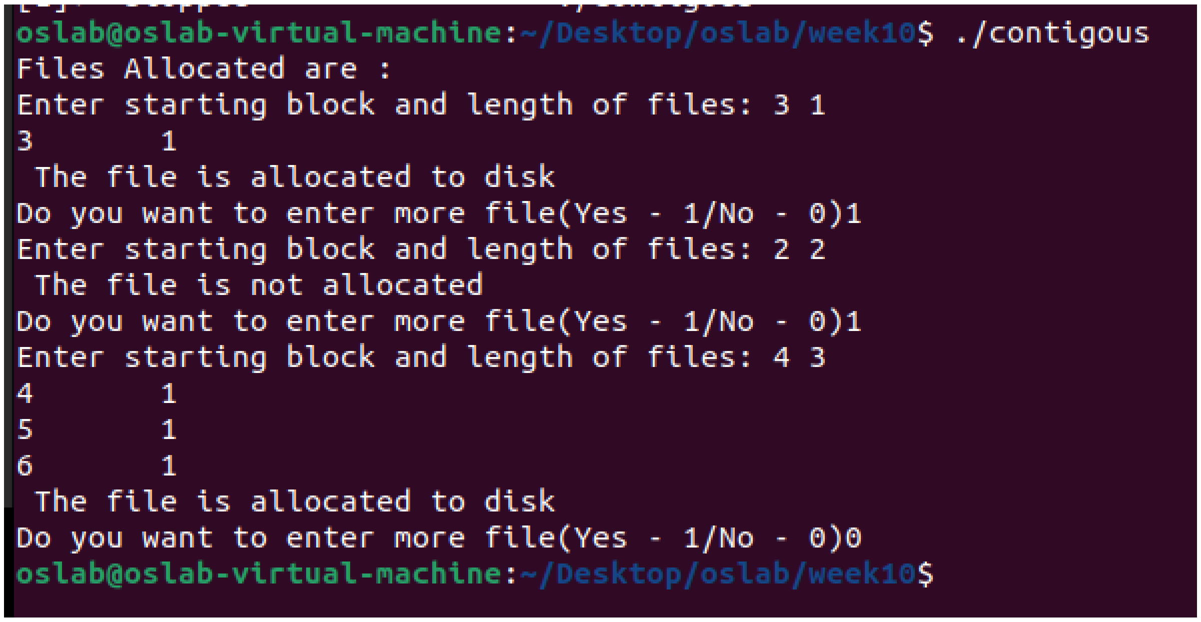
Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

CODE:

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int main()

{

int f[50], i, st, len, j, c, k, count = 0;

//clrscr();

for(i=0;i<50;i++)

f[i]=0;

printf("Files Allocated are : \n");

x: count=0;

printf("Enter starting block and length of files: ");

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;
if(len==count)
{
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("%d\t%d\n",j,f[j]);
}
if(j!=(st+len-1))
printf(" The file is allocated to disk\n");
}
else
printf(" The file is not allocated \n");
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
return 0;
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./contigous
Files Allocated are :
Enter starting block and length of files: 3 1
3       1
 The file is allocated to disk
Do you want to enter more file(Yes - 1/No - 0)1
Enter starting block and length of files: 2 2
 The file is not allocated
Do you want to enter more file(Yes - 1/No - 0)1
Enter starting block and length of files: 4 3
4       1
5       1
6       1
 The file is allocated to disk
Do you want to enter more file(Yes - 1/No - 0)0
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

4. WAP to demonstrate the use of signal – IPC (user defined Handler)

Ans.

AIM: A program to demonstrate the use of signal - IPC

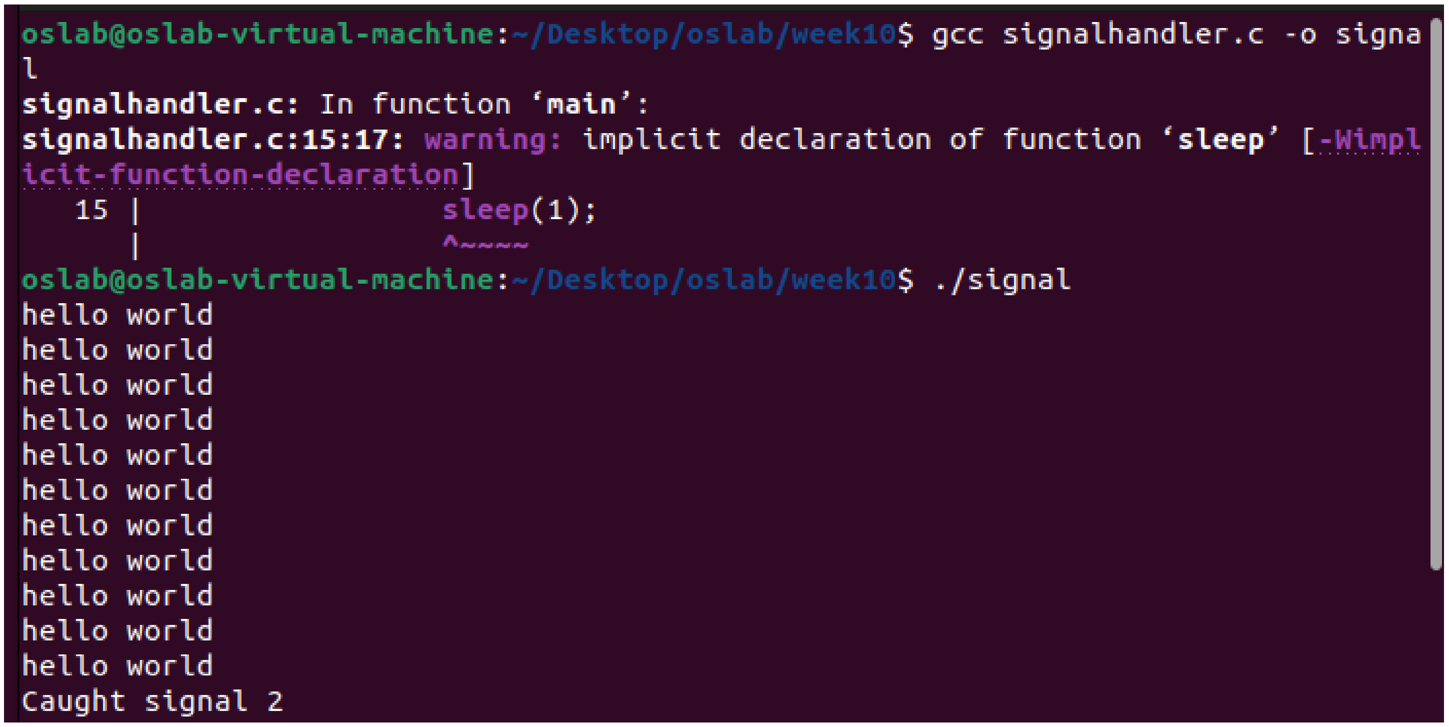
DESCRIPTION:

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. There are fix set of signals that can be sent to a process. signal are identified by integers. Signal number have symbolic names. For example SIGCHLD is number of the signal sent to the parent process when child terminates.

CODE:

```
#include<stdio.h>

#include<signal.h>

// Handler for SIGINT, caused by

// Ctrl-C at keyboard

void handle_sigint(int sig)    // Signal Handler

{

   printf("Caught signal %d\n", sig);

}

int main()

{

   signal(SIGINT, handle_sigint); // Registering a Signal

   while (1)

   {

      printf(“hello world\n”);

      sleep(1);

   }

   return 0;

}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc signalhandler.c -o signa
l
signalhandler.c: In function ‘main’:
signalhandler.c:15:17: warning: implicit declaration of function ‘sleep’ [-Wimpl
icit-function-declaration]
   15 |                 sleep(1);
      |                 ^~~~~
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./signal
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
Caught signal 2
```

5. **WAP to demonstrate the SIGCHLD and SIGINT handling**

Ans.

AIM: A program to demonstrate the SIGCHLD and SIGINT handling

DESCRIPTION:

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. There are fix set of signals that can be sent to a process. signal are identified by integers. Signal number have symbolic names. For example SIGCHLD is number of the signal sent to the parent process when child terminates.

CODE:

SIGINT handling:

```
#include<stdio.h>

#include<signal.h>

// Handler for SIGINT, caused by

// Ctrl-C at keyboard

void handle_sigint(int sig)    // Signal Handler

{

   printf("Caught signal %d\n", sig);

}

int main()

{
```

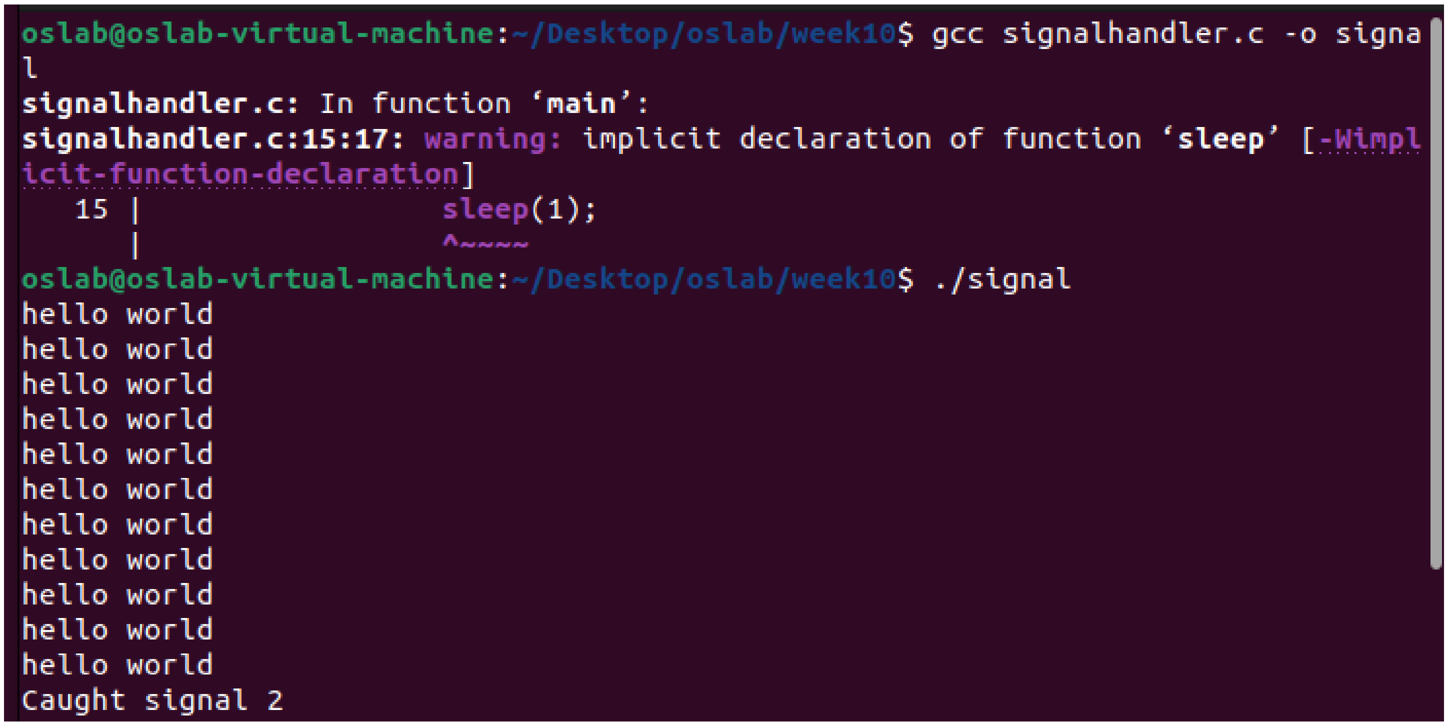
```
  signal(SIGINT, handle_sigint); // Registering a Signal
  while (1)
  {
     printf(“hello world\n”);
     sleep(1);
  }
  return 0;
}
```

SIGCHLD handling:

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
// Handler for SIGINT, caused by
// Ctrl-C at keyboard
void handle_sigchild(int sig)    // Signal Handler
{
  printf("Inside child signal %d \n", sig);
  //signal(SIGINT,SIG_DFL); //De registering Signal
}
int main()
{
         signal(SIGCHLD, handle_sigchild); // Registering a Signal
         int p=fork();
         if(p==0)
         {
          printf("Inside Child process \n");
          //
          printf("End of child Process \n");
          }
          else if(p>=0)
          {
            printf("Inside parent process \n");
            wait();
            printf("End of parent process \n");
           }
        return 0;
}
```

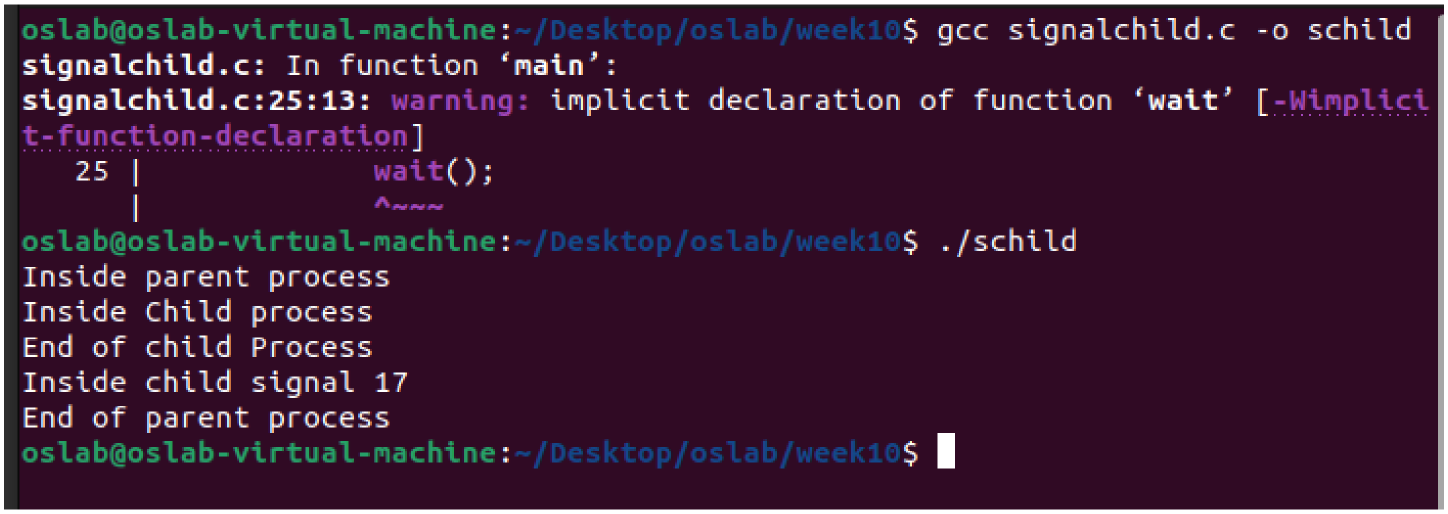
OUTPUT:

SIGINT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc signalhandler.c -o signa
l
signalhandler.c: In function ‘main’:
signalhandler.c:15:17: warning: implicit declaration of function ‘sleep’ [-Wimpl
icit-function-declaration]
   15 |                 sleep(1);
      |                 ^~~~~
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./signal
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
Caught signal 2
```

SIGCHLD:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc signalchild.c -o schild
signalchild.c: In function ‘main’:
signalchild.c:25:13: warning: implicit declaration of function ‘wait’ [-Wimplici
t-function-declaration]
   25 |             wait();
      |             ^~~~
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./schild
Inside parent process
Inside Child process
End of child Process
Inside child signal 17
End of parent process
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ 
```

6. WAP to demonstrate the use of shared memory – IPC

Ans.

AIM: A program to demonstrate the use of shared memory

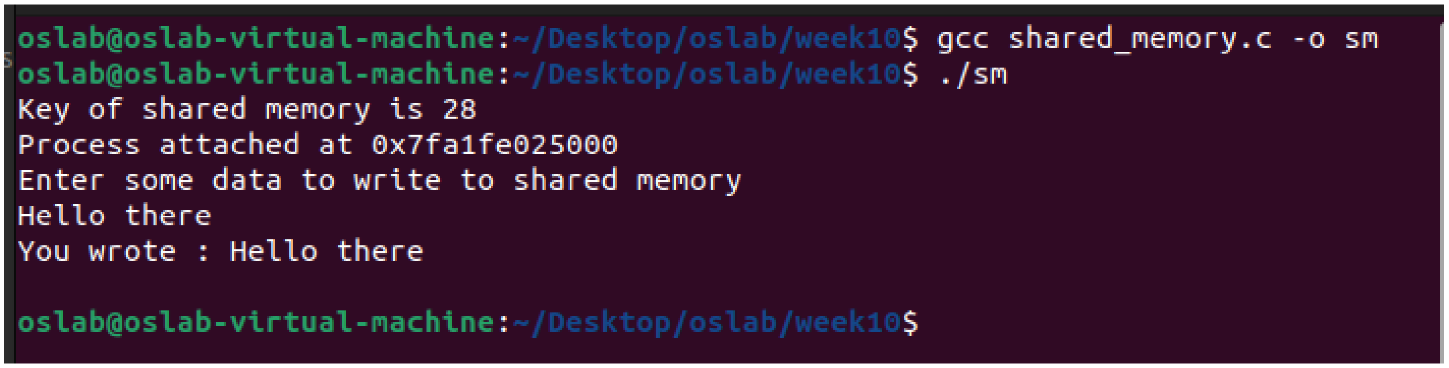
DESCRIPTION:

Shared Memory is the fastest inter-process communication (IPC) method. The operating system maps a memory segment in the address space of several processes so that those processes can read and write in that memory segment. Two functions:shmget() and shmat() are used for IPC using shared memory. shmget() function is used to create the shared memory segment while shmat() function is used to attach the shared segment with the address space of the process.

CODE:

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void *shared_memory;

char buff[100];

int shmid;

shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);


/*creates shared memory segment with key 2345, having size 1024 bytes. IPC_CREAT is used to create the shared
segment if it does not exist. 0666 are the permisions on the shared segment*/


printf("Key of shared memory is %d\n",shmid);

shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment

printf("Process attached at %p\n",shared_memory); //this prints the address where the segment is attached with this
process

printf("Enter some data to write to shared memory\n");

read(0,buff,100); //get some input from user

strcpy(shared_memory,buff); //data written to shared memory

printf("You wrote : %s\n",(char *)shared_memory);

}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc shared_memory.c -o sm
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./sm
Key of shared memory is 28
Process attached at 0x7fa1fe025000
Enter some data to write to shared memory
Hello there
You wrote : Hello there

oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

7. **WAP to demonstrate the use of semaphore – IPC**

Ans.

AIM: A program to demonstrate the use of semaphore - IPC

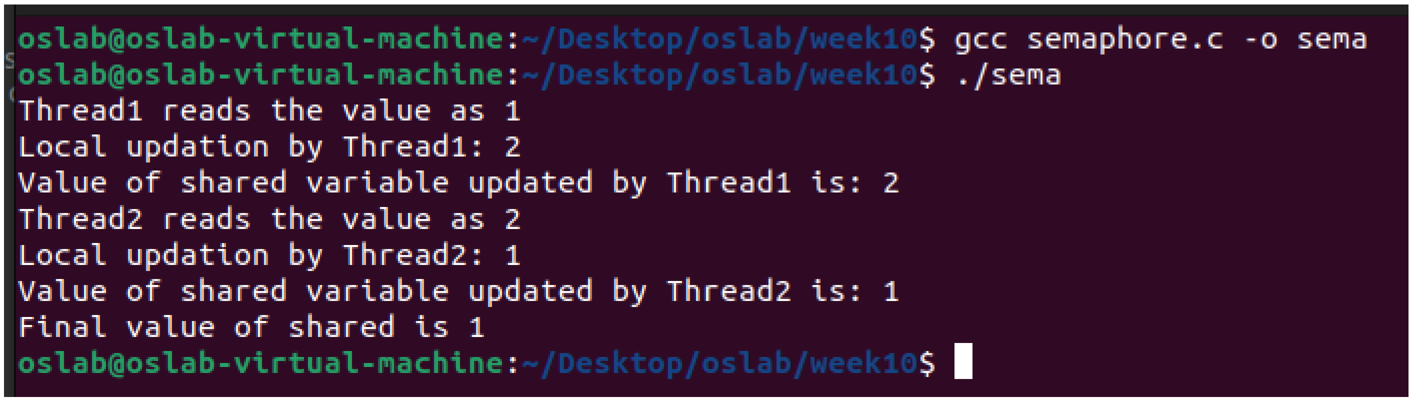
DESCRIPTION:

Semaphore is an integer variable which is accessed or modified by using two atomic operations: *wait()* and *signal()*. In C program the corresponding operations are *sem_wait()* and *sem_post()*. Here, we write a Program for Process Synchronization using Semaphores to understand the implementation of *sem_wait()* and *sem_signal()* to avoid a race condition.

CODE:

```
#include<pthread.h>

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

void *fun1();

 void *fun2();

 int shared=1; //shared variable

 sem_t s; //semaphore variable

 int main()

 {

 sem_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is number of processes sharing semaphore, 3rd argument is the initial value of semaphore variable

 pthread_t thread1, thread2;

 pthread_create(&thread1, NULL, fun1, NULL);

 pthread_create(&thread2, NULL, fun2, NULL);

 pthread_join(thread1, NULL);

 pthread_join(thread2,NULL);

 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable
 }
void *fun1()
{
   int x;
   sem_wait(&s); //executes wait operation on s
   x=shared;//thread1 reads value of shared variable
   printf("Thread1 reads the value as %d\n",x);
   x++;  //thread1 increments its value
   printf("Local updation by Thread1: %d\n",x);
   sleep(1); //thread1 is preempted by thread 2
   shared=x; //thread one updates the value of shared variable
   printf("Value of shared variable updated by Thread1 is: %d\n",shared);
   sem_post(&s);
}
void *fun2()
{
   int y;
   sem_wait(&s);
   y=shared;//thread2 reads value of shared variable
   printf("Thread2 reads the value as %d\n",y);
   y--;  //thread2 increments its value
   printf("Local updation by Thread2: %d\n",y);
   sleep(1); //thread2 is preempted by thread 1
   shared=y; //thread2 updates the value of shared variable
   printf("Value of shared variable updated by Thread2 is: %d\n",shared);
   sem_post(&s);
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc semaphore.c -o sema
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./sema
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

8. **WAP to demonstrate the use of Mutex – IPC**

Ans.

AIM: A program to demonstrate the use of mutex - IPC

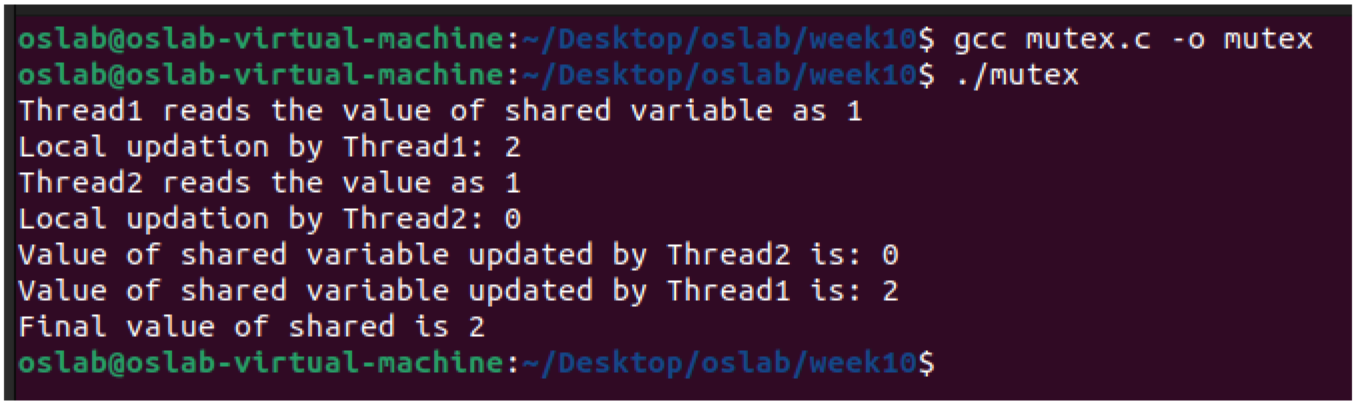
DESCRIPTION:

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called a *race condition*. This Program uses threads to simulate race condition.

CODE:

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1; //shared variable
int main()
 {
 pthread_t thread1, thread2;
 pthread_create(&thread1, NULL, fun1, NULL);
 pthread_create(&thread2, NULL, fun2, NULL);
 pthread_join(thread1, NULL);
 pthread_join(thread2,NULL);
 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable
 }
void *fun1()
{
   int x;
   x=shared;//thread one reads value of shared variable
   printf("Thread1 reads the value of shared variable as %d\n",x);
   x++;  //thread one increments its value
   printf("Local updation by Thread1: %d\n",x);
   sleep(1);  //thread one is preempted by thread 2
   shared=x; //thread one updates the value of shared variable
   printf("Value of shared variable updated by Thread1 is: %d\n",shared);
}
void *fun2()
 {
    int y;
    y=shared;//thread two reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread two increments its value
    printf("Local updation by Thread2: %d\n",y);
    sleep(1); //thread two is preempted by thread 1
    shared=y; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
 }
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc mutex.c -o mutex
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./mutex
Thread1 reads the value of shared variable as 1
Local updation by Thread1: 2
Thread2 reads the value as 1
Local updation by Thread2: 0
Value of shared variable updated by Thread2 is: 0
Value of shared variable updated by Thread1 is: 2
Final value of shared is 2
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

9. **WAP to demonstrate the use of message queue – IPC**

Ans.

AIM: A program to demonstrate the use of message queue - IPC

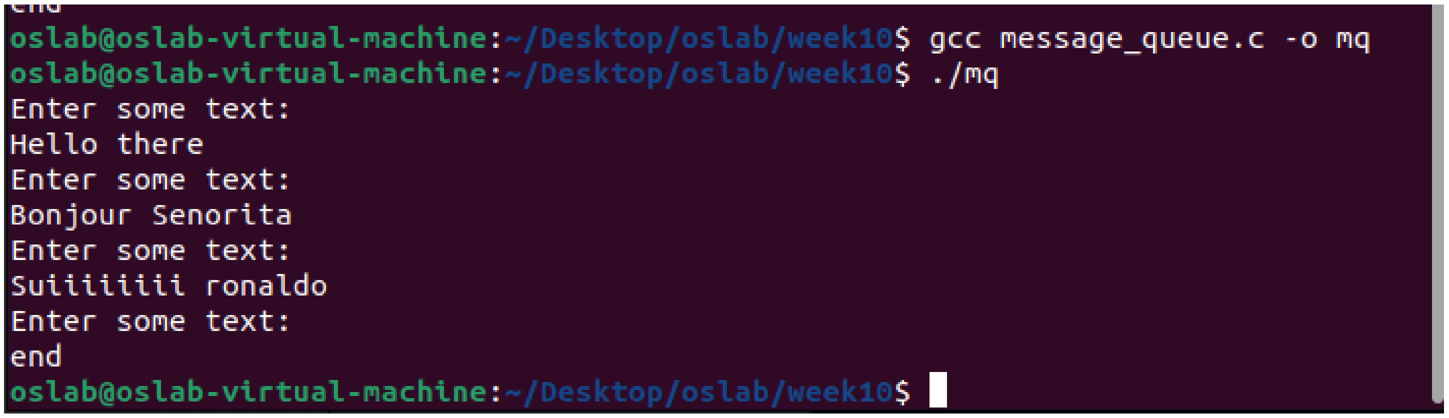
DESCRIPTION:

Program for IPC using Message queues are almost similar to named pipes with the exception that they do not require the opening and closing of pipes. But, they face one similar problem like named pipes; blocking on full pipes. Message queues send blocks of data from one process to another. Each block of data is considered to have a type. There is an upper limit on the maximum size of each block and also a limit on the maximum total size of all blocks on all queues in the system.

CODE:

```
#include<stdlib.h>
 #include<stdio.h>
 #include<string.h>
 #include<unistd.h>
 #include<sys/types.h>
 #include<sys/ipc.h>
 #include<sys/msg.h>
 #define MAX_TEXT 512   //maximum length of the message that can be sent allowed
 struct my_msg{
    long int msg_type;
    char some_text[MAX_TEXT];
 };
 int main()
 {
    int running=1;
    int msgid;
    struct my_msg some_data;
    char buffer[50]; //array to store user input
    msgid=msgget((key_t)14534,0666|IPC_CREAT);
    if (msgid == -1) // -1 means the message queue is not created
    {
        printf("Error in creating queue\n");
        exit(0);
    }

    while(running)
    {
        printf("Enter some text:\n");
        fgets(buffer,50,stdin);
        some_data.msg_type=1;
        strcpy(some_data.some_text,buffer);
        if(msgsnd(msgid,(void *)&some_data, MAX_TEXT,0)==-1) // msgsnd returns -1 if the message is not sent
        {
            printf("Msg not sent\n");
        }
        if(strncmp(buffer,"end",3)==0)
        {
            running=0;
        }
    }
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ gcc message_queue.c -o mq
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./mq
Enter some text:
Hello there
Enter some text:
Bonjour Senorita
Enter some text:
Suiiiiiii ronaldo
Enter some text:
end
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$
```

10. WAP to demonstrate the use of socket – IPC

Ans.

AIM: A program to demonstrate the use of socket - IPC

DESCRIPTION:

A socket is **a bi-directional data transfer mechanism**. They are used to transfer data between two processes. The two processes can be running on the same system as Unix-domain or loopback sockets, or on different systems as network sockets. Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

CODE:

SOCKET:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(connfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

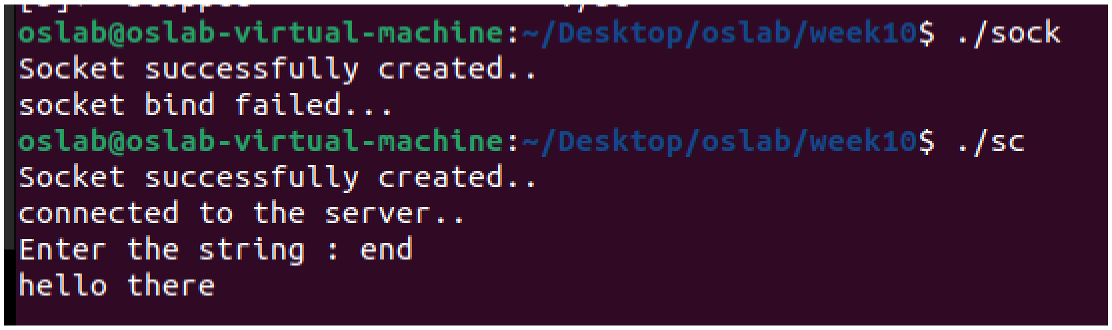
CLIENT:

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
   char buff[MAX];
   int n;
   for (;;) {
      bzero(buff, sizeof(buff));
      printf("Enter the string : ");
      n = 0;
      while ((buff[n++] = getchar()) != '\n')
         ;
      write(sockfd, buff, sizeof(buff));
      bzero(buff, sizeof(buff));
      read(sockfd, buff, sizeof(buff));
      printf("From Server : %s", buff);
      if ((strncmp(buff, "exit", 4)) == 0) {
         printf("Client Exit...\n");
         break;
      }
   }
}

int main()
{
   int sockfd, connfd;
```

```
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

OUTPUT:



```
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./sock
Socket successfully created..
socket bind failed...
oslab@oslab-virtual-machine:~/Desktop/oslab/week10$ ./sc
Socket successfully created..
connected to the server..
Enter the string : end
hello there
```