EX NO: 1B	**SYSTEM CALLS OF UNIX OPERATING SYSTEM**
DATE:	

I. CLOSE() AIM:

To write the program to implement the system calls close().

ALGORITHM:

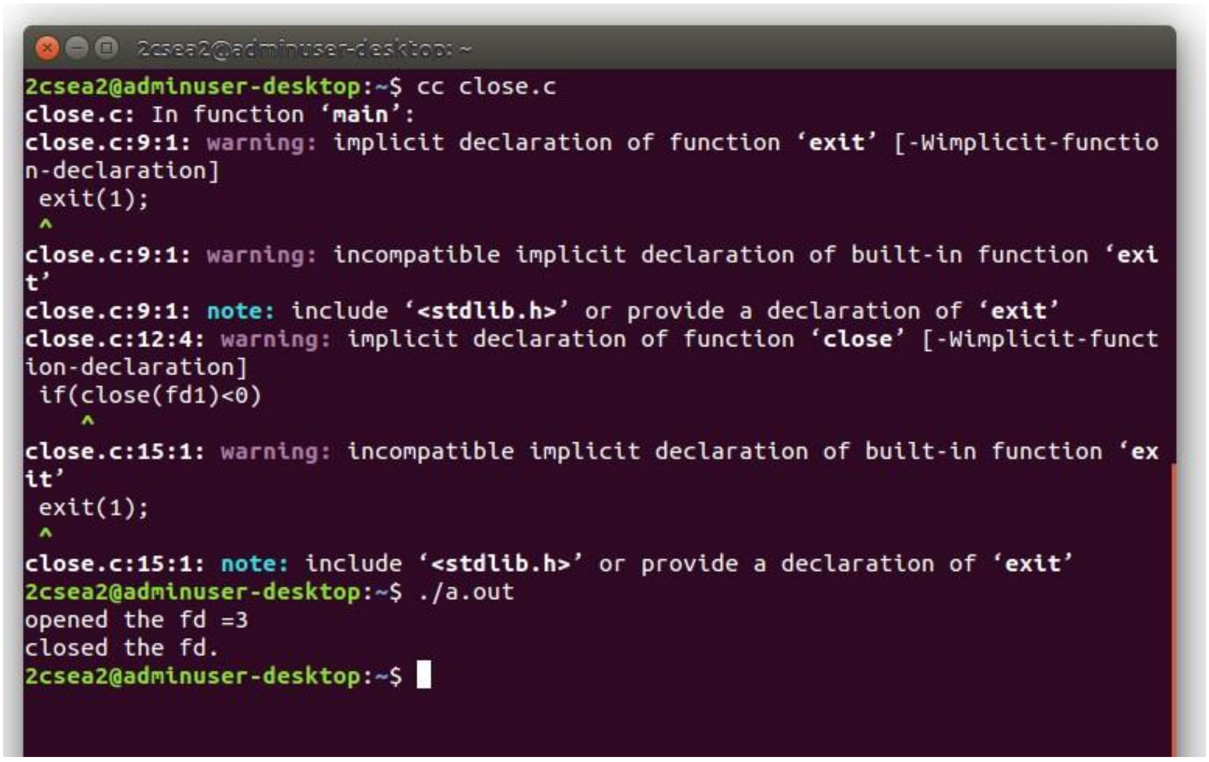
Step 1: Start
Step 2: In the main function pass the arguments.
Step 3: Create structure as stat buff and the variables as integer. Step
4: Use the for loop initialization.
Step 5: Stop.

PROGRAM:

3

```
#include<stdio.h>

#include<fcntl.h>

int main() {

int fd1 = open("foo.txt",O_RDONLY);

if(fd1<0)

{

perror("c1");
exit(1); } printf("opened the fd

=%d\n",fd1); if(close(fd1)<0) {

perror("c1"); exit(1); }

printf("closed the fd.\n");

}
```

OUTPUT:



```
2csea2@adminuser-desktop:~$ cc close.c
close.c: In function ‘main’:
close.c:9:1: warning: implicit declaration of function ‘exit’ [-Wimplicit-functio
n-declaration]
 exit(1);
 ^
close.c:9:1: warning: incompatible implicit declaration of built-in function ‘exi
t’
close.c:9:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
close.c:12:4: warning: implicit declaration of function ‘close’ [-Wimplicit-funct
ion-declaration]
 if(close(fd1)<0)
    ^
close.c:15:1: warning: incompatible implicit declaration of built-in function ‘ex
it’
 exit(1);
 ^
close.c:15:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
2csea2@adminuser-desktop:~$ ./a.out
opened the fd =3
closed the fd.
2csea2@adminuser-desktop:~$
```

RESULT:

Thus the program was executed and verified successfully.

II. GETPID()

AIM:

To write the program to implement the system calls getpid()

ALGORITHIM:

```
Step 1: Start
Step 2: Get the process id integer value by using the system call getpid()
Step 3: It returns the process id of the calling process.
Step 4: After getting the pid value it prints the process id number an exists.
Step 5: Then compile the program either with the gcc or cc command.
Step 6: Run the program.
Step 7: Stop
```

PROGRAM:

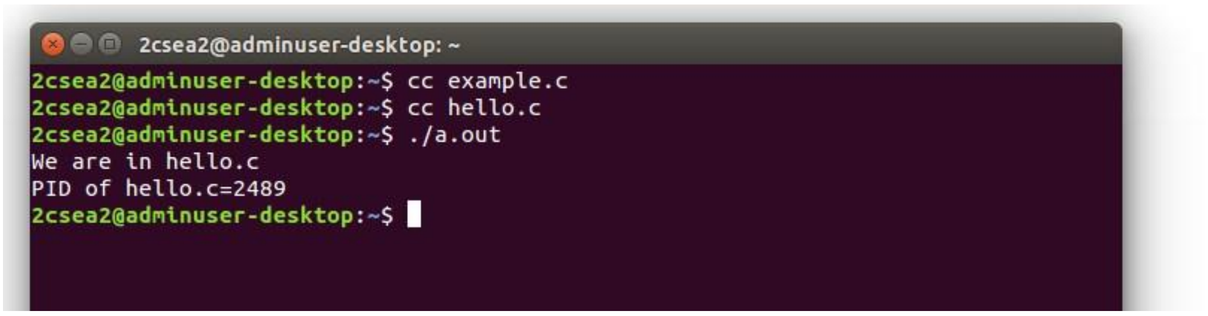
Example.c

```
#include<stdio.h>

#include<unistd.h>

#include<stdlib.h> int

main(int argc,char *argv[])

{ printf("PID of example.c=%d\n",getpid()); char

*args[]={"hello", "c","programming",NULL};

execv("./hello",args);

printf("BACK TO EXAMPLE.C");

}
```

Hello.c

```
#include<stdio.h>

#include<unistd.h>

#include<stdlib.h> int main

(int argc, char *argv[])

{    printf("We    are    in    hello.c\n");

printf("PID                                 of

hello.c=%d\n",getpid()); return 0;

}
```

OUTPUT:



```
2csea2@adminuser-desktop:~$ cc example.c
2csea2@adminuser-desktop:~$ cc hello.c
2csea2@adminuser-desktop:~$ ./a.out
We are in hello.c
PID of hello.c=2489
2csea2@adminuser-desktop:~$
```

RESULT:

Thus the program for getpid() system call has been executed and verified successfully.

III. FORK()

AIM:

To write the program to create a Child Process using system call fork().

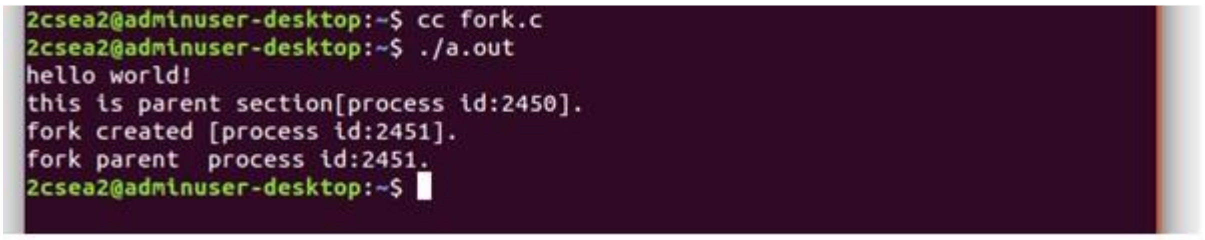
ALGORITHM:

Step 1: Declare the variable pid.
Step 2: Get the pid value using system call fork().
Step 3: If pid value is less than zero then print as “Fork failed”.
Step 4: Else if pid value is equal to zero include the new process in the system‟s file using execlp system call.
Step 5: Else if pid is greater than zero then it is the parent process and it waits till the child completes using the system call wait() Step 6: Then print “Child complete”.

PROGRAM:

```
#include  <stdio.h>

#include<unistd.h

>  int  main()  {  int

id;

printf("hello world!\n"); id=fork(); if (id>0) { printf("this

is parent section[process id:%d].\n",getpid());
} else

if(id==0) {

printf("fork created [process id:%d].\n",getpid());

printf("fork parent  process id:%d.\n",getpid());

} else { printf("fork created

failed!!\n");

}

return

0;

}
```

OUTPUT:



```
2csea2@adminuser-desktop:~$ cc fork.c
2csea2@adminuser-desktop:~$ ./a.out
hello world!
this is parent section[process id:2450].
fork created [process id:2451].
fork parent  process id:2451.
2csea2@adminuser-desktop:~$ 
```

RESULT:

Thus the program for fork() system call has been executed and verified successfully.

IV. OPEN()

AIM:

To write the program to implement the system call open( ).

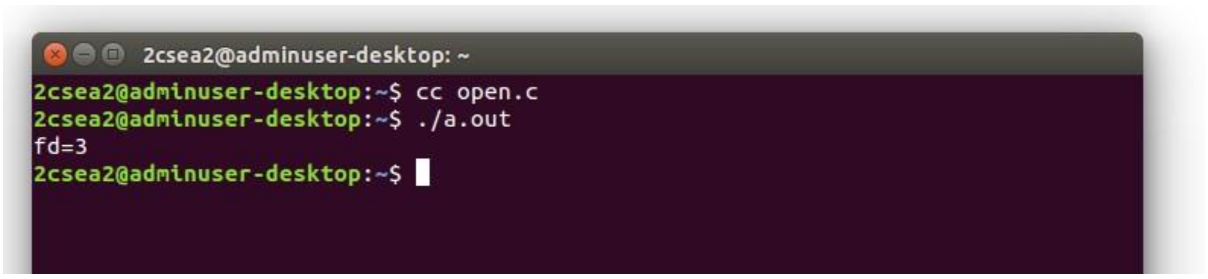
ALGORITHM:

Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.

PROGRAM:

```
#include<stdio.h>

#include<fcntl.h>

#include<errno.h>

extern int errno;

int main()

{

int fd=open("foo.txt",O_RDONLY|O_CREAT);

printf("fd=%d\n",fd);

if(fd ==-1)

{

printf("Eror no.=%d\n",errno);

printf("Program");

}

return 0;

}
```

OUTPUT:



```
2csea2@adminuser-desktop:~$ cc open.c
2csea2@adminuser-desktop:~$ ./a.out
fd=3
2csea2@adminuser-desktop:~$
```

RESULT:

Thus the program for open() system call has been executed and verified successfully.

V. READ()

AIM:

To write the program to implement the system call read( ).

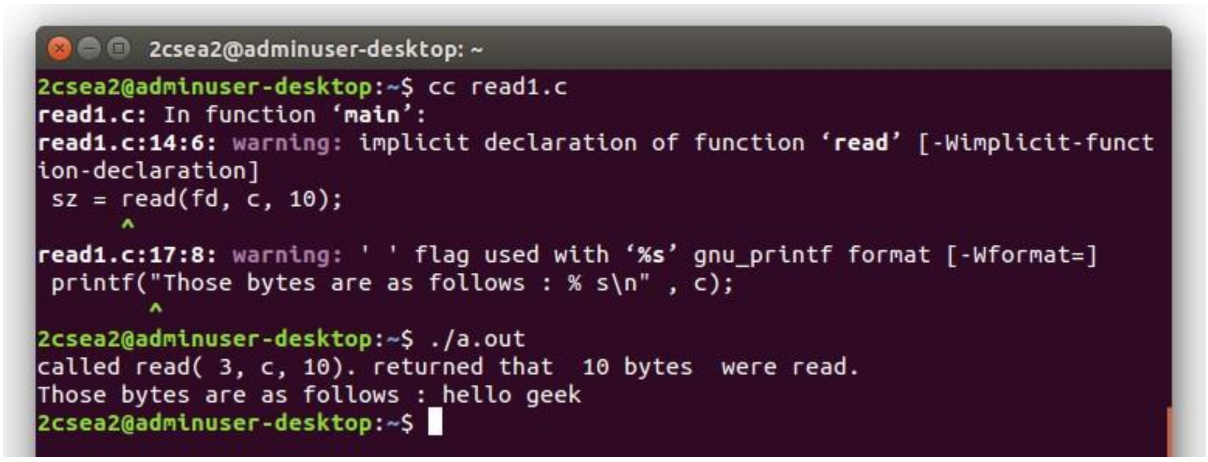
ALGORITHM:

```
Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.
```

PROGRAM:

```
#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h

> int main() { int

fd, sz;

char *c = (char *) calloc(100,sizeof(char)); fd

= open("foo.txt", O_RDONLY);
if(fd < 0) { perror("r1"); exit(1); } sz = read(fd, c, 10); printf("called read(% d,

c, 10). returned that" " % d bytes  were read.\n" , fd, sz);

c[sz] = '\0';

printf("Those bytes are as follows : % s\n" , c);

}
```

OUTPUT:



```
2csea2@adminuser-desktop: ~
2csea2@adminuser-desktop:~$ cc read1.c
read1.c: In function ‘main’:
read1.c:14:6: warning: implicit declaration of function ‘read’ [-Wimplicit-funct
ion-declaration]
 sz = read(fd, c, 10);
      ^
read1.c:17:8: warning: ' ' flag used with ‘%s’ gnu_printf format [-Wformat=]
 printf("Those bytes are as follows : % s\n" , c);
        ^
2csea2@adminuser-desktop:~$ ./a.out
called read( 3, c, 10). returned that  10 bytes  were read.
Those bytes are as follows : hello geek
2csea2@adminuser-desktop:~$ █
```

RESULT:

Thus the program for read() system call has been executed and verified successfully.

VI. WRITE ()

AIM:

To write the program to implement the system call write( ).

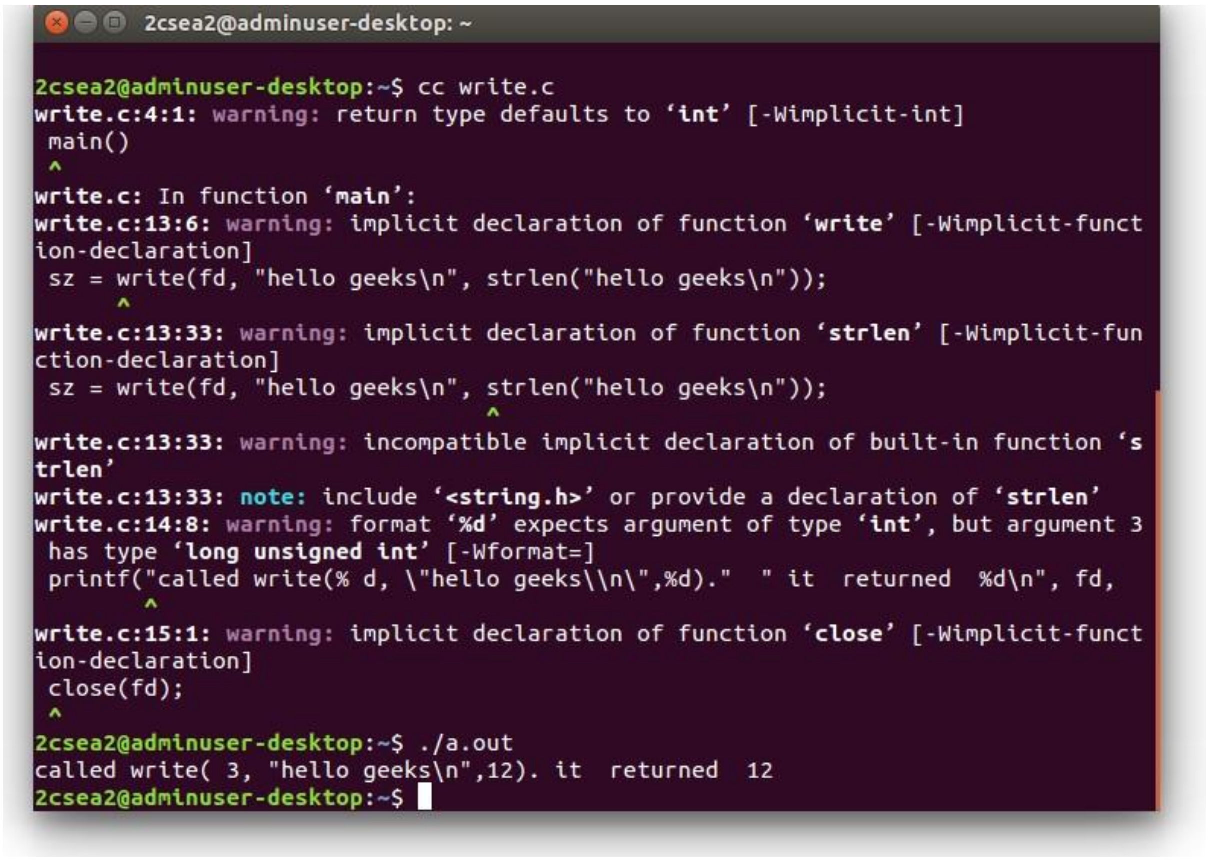
ALGORITHM:

```
Step 1 : Declare the structure elements.
Step 2 : Create a temporary file named temp1.
Step 3 : Open the file named “test” in a write mode.
Step 4 : Enter the strings for the file.
Step 5 : Write those strings in the file named “test”.
Step 6 : Create a temporary file named temp2.
Step 7 : Open the file named “test” in a read mode.
Step 8 : Read those strings present in the file “test” and save it in temp2.
Step 9 : Print the strings which are read.
```

PROGRAM:

```
#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h

> main() {

int  sz;

int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
if(fd < 0) { perror("r1"); exit(1); } sz = write(fd,

"hello geeks\n", strlen("hello geeks\n"));

printf("called write(% d, \"hello geeks\\n\",%d)."  " it  returned  %d\n", fd,
strlen("hello geeks\n"), sz);  close(fd);

}
```

OUTPUT:



```
2csea2@adminuser-desktop:~$ cc write.c
write.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
write.c: In function ‘main’:
write.c:13:6: warning: implicit declaration of function ‘write’ [-Wimplicit-funct
ion-declaration]
 sz = write(fd, "hello geeks\n", strlen("hello geeks\n"));
      ^
write.c:13:33: warning: implicit declaration of function ‘strlen’ [-Wimplicit-fun
ction-declaration]
 sz = write(fd, "hello geeks\n", strlen("hello geeks\n"));
                                 ^
write.c:13:33: warning: incompatible implicit declaration of built-in function ‘s
trlen’
write.c:13:33: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
write.c:14:8: warning: format ‘%d’ expects argument of type ‘int’, but argument 3
 has type ‘long unsigned int’ [-Wformat=]
 printf("called write(% d, \"hello geeks\\n\",%d)."  " it  returned  %d\n", fd,
        ^
write.c:15:1: warning: implicit declaration of function ‘close’ [-Wimplicit-funct
ion-declaration]
 close(fd);
 ^
2csea2@adminuser-desktop:~$ ./a.out
called write( 3, "hello geeks\n",12). it  returned  12
2csea2@adminuser-desktop:~$ 
```

RESULT:

Thus the program for write() system call has been executed and verified successfully.