OS Lab Exercise 2 -Implementation Of System Calls -B.Ajay(21110103)

Aim:

To write C Program to implement the following system calls

- Open ()
- Read ()
- Write ()
- Wait ()
- Exec ()
- Fork ()
- Sleep ()
- Getpid ()
- Lseek ()

Descriptions:

1.Open():

Header:

```
#include <fcntl.h>
```

Syntax:

```
int open(const char *pathname, int flags);

int open(const char *pathname, int flags, mode_t mode);
```

Description:

->The open() system call opens the file specified by pathname. If the specified file does not exist, it may optionally (if O_CREAT is specified in flags) be created by open().

The return value of open() is a file descriptor, a small,

nonnegative integer that is an index to an entry in the process's

table of open file descriptors. The file descriptor is used in

subsequent system calls (read(2), write(2), lseek(2), fcntl(2),

etc.) to refer to the open file. The file descriptor returned by

a successful call will be the lowest-numbered file descriptor not

currently open for the process

2.read():

Header:

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

Syntax:

```
ssize_t read(int fd, void *buf, size_t count);
```

Description:

read() attempts to read up to count bytes from file descriptor fd

into the buffer starting at buf.

On files that support seeking, the read operation commences at

the file offset, and the file offset is incremented by the number

of bytes read. If the file offset is at or past the end of file,

no bytes are read, and read() returns zero.

If count is zero, read() may detect the errors described below.

In the absence of any errors, or if read() does not check for

errors, a read() with a count of 0 returns zero and has no other

effects.

3.write():

Header:

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

Syntax:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Description:

write() writes up to count bytes from the buffer starting at buffer

to the file referred to by the file descriptor fd.

The number of bytes written may be less than count if, for

example, there is insufficient space on the underlying physical

medium, or the RLIMIT_FSIZE resource limit is encountered

or the call was interrupted by a signal handler

after having written less than count bytes.

4.wait():

Header:

```
#include <sys/wait.h>
```

Syntax:

```
pid_t wait(int *wstatus);
```

Description:

The wait() system call suspends execution of the calling thread until one of its children terminates.

these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state

5.exec():

Header:

#include <unistd.h>

Syntax:

int execl(const char *pathname, const char *arg, ...

/*, (char *) NULL */);

Description:

The exec() family of functions replaces the current process image

with a new process image. The functions described in this manual

page are layered on top of execve(2). (See the manual page for

execve(2) for further details about the replacement of the

current process image.)

The initial argument for these functions is the name of a file

that is to be executed.

The functions can be grouped based on the letters following the

"exec" prefix.

6.Fork():

Header :

#include <unistd.h>

Syntax:

pid_t fork(void);

Description:

fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.

The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces have the same content. Memory writes, file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the processes do not affect the other.

7. sleep():

Header:

#include <unistd.h>

Syntax:

unsigned int sleep(unsigned int seconds);

Description:

sleep() causes the calling thread to sleep either until the number of real-time seconds specified in seconds have elapsed or until a signal arrives which is not ignored.

8.getpid():

Header:

#include <unistd.h>

Syntax:

pid_t getpid(void);

Description:

getpid() returns the process ID (PID) of the calling process.

(This is often used by routines that generate unique temporary filenames.)

9.lseek():

Header:

#include <unistd.h>

Syntax:

off_t lseek(int fd, off_t offset, int whence);

Description:

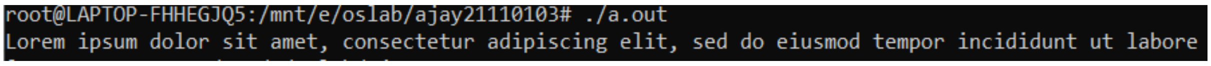
lseek() repositions the file offset of the open file description associated with the file descriptor fd to the argument offset

Programs:

1 **<u>Open:</u>**

```
#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>
#include <fcntl.h>

int main(){

int k=open("test.txt",O_RDONLY);

char buffer[100];

read(k,buffer,100);

write(1,buffer,100);

}
```

Output:



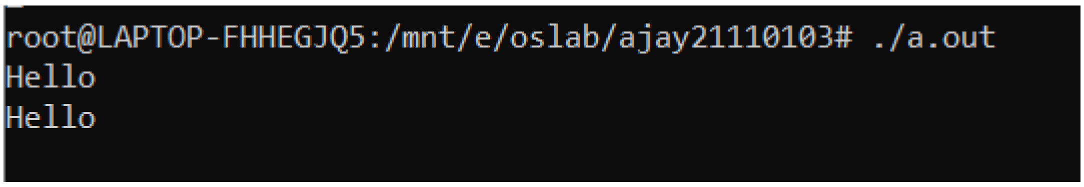
```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
```

2. Read():

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main(){
```

```
char buffer[15];

read(1,buffer,10);

printf("%s\n",buffer);

}
```

Output:

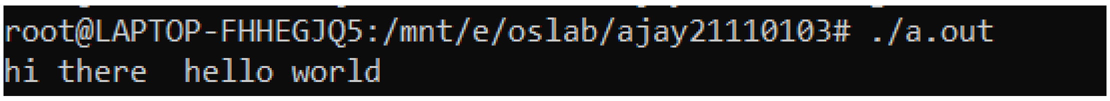


```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Hello
Hello
```

3.write():

```
#include <unistd.h>

int main(){

char bufferr[30]="hi there  hello world\n";

write(1,bufferr,30);

}
```

Output:



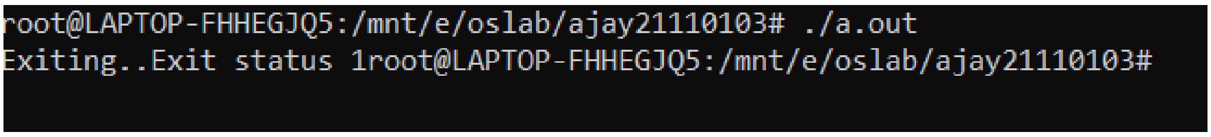
```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
hi there  hello world
```

4.wait():

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>
```

```
#include <unistd.h>

int main(){

        int status;

        if(fork()==0)

        {

        printf("Exiting..");

        exit(1);

        }else{

        wait(&status);

        }

        printf("Exit status %d",WEXITSTATUS(status));

}
```

Output:



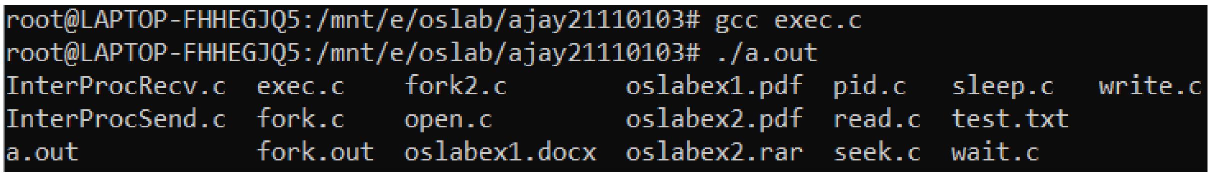
```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Exiting..Exit status 1root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103#
```

5.Exec():

```
#include <unistd.h>

int main(){
```

```
char* path="/bin/ls";

char* arg[]={path,"-la",NULL};

execl(path,"-la",NULL);

execv(path,arg);

}
```

Output:



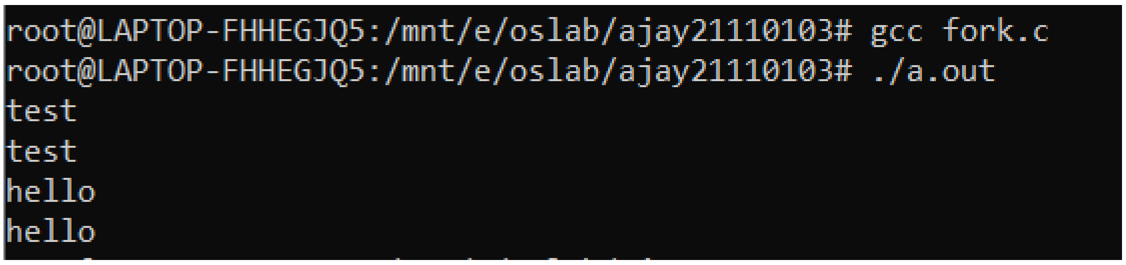
```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc exec.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
InterProcRecv.c  exec.c    fork2.c        oslabex1.pdf  pid.c   sleep.c   write.c
InterProcSend.c  fork.c    open.c         oslabex2.pdf  read.c  test.txt
a.out            fork.out  oslabex1.docx  oslabex2.rar  seek.c  wait.c
```

6. Fork():

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void test(){

fork();

//fork();

//fork();

printf("test\n");

}

int main(){
```

```
test();

//fork()&&fork()||fork();

//fork();

//fork();

//fork();

printf("hello\n");

}
```

Output:

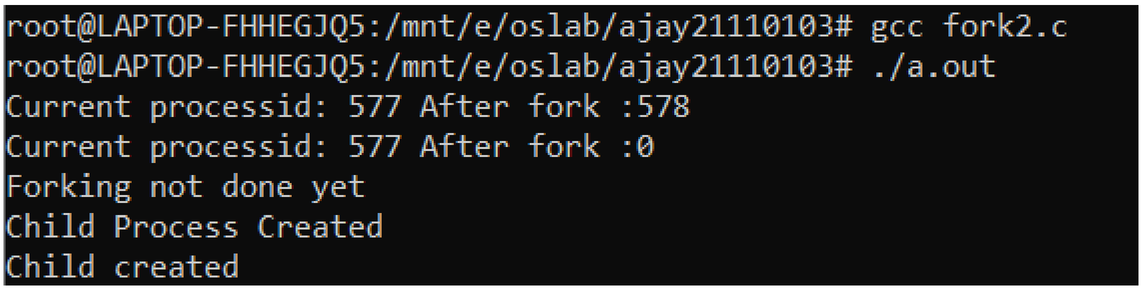


```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc fork.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
test
test
hello
hello
```

Fork():

```
#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main(){

printf("Current processid: %d",(int)getpid());

pid_t a=fork();
printf(" After fork :%d\n",(int)a);

if(a<0){

fprintf(stderr,"Error \n");



}

else if (a==0){

printf("Child Process Created\n");

}

else{

printf("Forking not done yet\n");

wait(NULL);

printf("Child created\n");

}

}
```

Output:



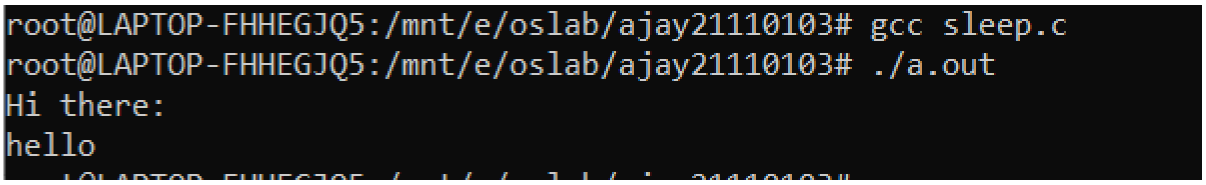
```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc fork2.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Current processid: 577 After fork :578
Current processid: 577 After fork :0
Forking not done yet
Child Process Created
Child created
```

7.sleep():

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

```
#include <unistd.h>

int main(){

printf("Hi there:\n");

sleep(3);

printf("hello\n");

}
```

Output:

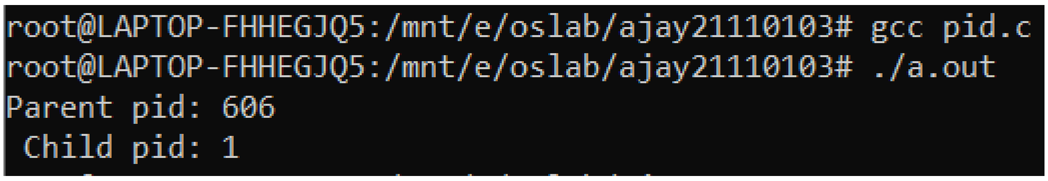


```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc sleep.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Hi there:
hello
```

8.getpid():

```
#include <unistd.h>

#include <stdio.h>

int main(){

if(fork()==0){

printf("Parent pid: %d\n ",getpid());

printf("Child pid: %d\n",getppid());

}

}
```

Output:

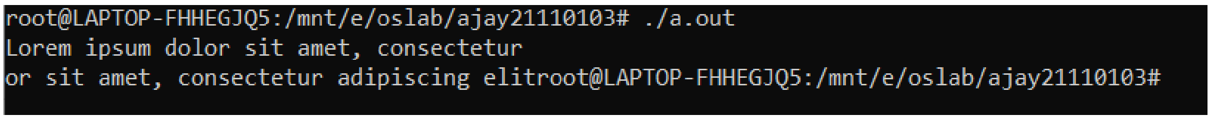


```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc pid.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Parent pid: 606
 Child pid: 1
```

9.lseek():

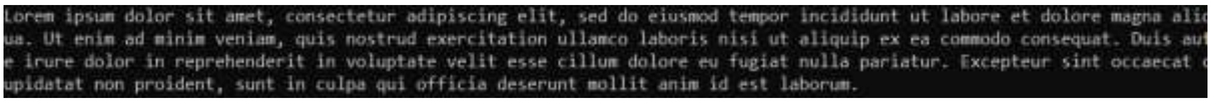
```
#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

int main(){

int buf[40];

int fd=open("test.txt",O_RDWR);

read(fd,buf,40);

write(1,buf,40);

lseek(fd,15,SEEK_SET);

write(1,"\n",1);

read(fd,buf,40);

write(1,buf,40);

}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Lorem ipsum dolor sit amet, consectetur
or sit amet, consectetur adipiscing elitroot@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103#
```

Test.txt:



```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna ali
ua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis au
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
upidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Result:

Thus the above linux system calls were implemented in c