Operating System concepts:

Introduction:

- Unix is written in C languages and it has Dedicate C shell for providing full-fledged scripting of C languages in unix enviornment.

Processes, Child-Parent relationship

When we boot the system special process called scheduler or swapper is created with PID 0. The swapper creates chiled called process dispatcher and swapper manages memory allocation for process. Process dispatcher n ow create shell. From now onward all the process create by us is chich of shell and decendent of dispatcher. Unix keeps track of all the process in a data structure called the process table.

Forking Processes

- processes initiated or created by user can also create a children process.
- Fork is the function to create the child process that is duplicate of a parent process.
- Child process begin execution from fork function
-

Exercise:

1.

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

void main()
{
fork();
printf("hello world");
}
```

Process Identification:

- PID is assiciated with child process
- getpid() function return the process id of that process.

Example:

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

void main()
{
int pid;
pid=getpid();
printf("process id is %d", pid);
}
```

- getppid() function return the parent ID.
- Fork function return 0 for child process and child pid for parent process.

Example:

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

```
#include <unistd.h>

void main()
{
int x=fork();
//printf("hello world");
int pid,ppid;
pid=getpid();
ppid=getppid();
if (x==0)    //   Fork function return 0 for child process
        {
        printf("process id  child %d \n", pid);
        printf("process id child's parent %d\n", ppid);
        }
else
        {
        printf("process id  parent %d \n", pid);
        printf("process id parent's parent %d\n", ppid);
        }
}
```

- Orphan Process:
  - When child process is in running state but its parent completed its execution then child process is left as an orphan process The process dispatcher immediately becomes the parent process of all such processes.

Example:

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

void main()
{
int x=fork();
if (x==0)
        {
        printf("child %d \n", getpid());
        printf("child parent %d\n", getppid());
        sleep(20);
        printf("child %d \n", getpid());
        printf("child parent %d\n", getppid());

        }
else
        {
        printf("parent %d \n", getpid());
        printf("parent's parent %d\n", getppid());
        }
}
```

Zombie Process

Unix has a concept of zombie process that are dead but have not removed from the PROCESS TABLE. Consider the example below:

```
#include<stdio.h>
#include<unistd.h>
void main()
{
        if (fork() > 0)
        {
        printf("parent");
        sleep(50);
        }
}
```

Type ps -el command to see the status of zombie process:
you will find <defunct> and z in the staus.

Sleeping Process: The second column of the process table always shows the status of the process. (R for running O for orphan S for sleeping)

Process Syncronization:
1. Parent process should held up till child process complete its execution.
2. wait() funtion is used for this purpose.
3.parent process wait() till the signal for the completion of the child process.
4. Ideally parent process should wait for the child process to complete. So wait() can be used to syncronise.
5.If there is no child for a process, wait() will return -1. If child is terminated child pid is returned to parent. If child is is running then Parent process will go to suspended state.
6. If more than one child is there and if parent want to wait for all children then sleep can be used for this purpose.
Assignment: Please verify above facts by writing the code.

Sharing data between processess using file.

```
#include<fcntl.h>
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
void main()
{
        int fp;
        char chr='A';
        int pid;
        pid = fork();
        if (pid == 0)
        {
                fp=open("abc", O_WRONLY, 0666);
                printf("In child character is %c ", chr);
```

```
                chr='B';
                write(fp, &chr, 1);
                printf("In child character after change %c", chr);
        }
        else
        {
                sleep(20);
                wait((int*)0);
                fp=open("abc", O_RDONLY);
                read(fp, &chr,1);
                printf("Character after parents reads is %c", chr);
                close(fp);
        }
```

} output: In child character is A In child character after change BCharacter after parents reads is B

Important point in above code to learn.

1. learn about file handling open function and its parameter read and write function.

Sharing of File Descriptor

A file unlike variable is never duplicated.
FILE DESCRIPTOR TABLE of a file is shared with all the children of that parent who forked the child and opened the file before forked.
So all the file opened in parent will also be opened in child.
If a file is opened then entry is made in the STSTEM FILE TABLE. File pointer and access mode are stored in this table. This table is global table. So not only file its file descriptor and access mode is shared between processes.

Code:

```
#include<fcntl.h>
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
void main()
{
        int fp;
        char buffer[10];
        int pid;
      fp=open("abc", O_RDONLY);
        pid = fork();
        if (pid == 0)
        {
                printf("child process ID %d \n", getpid());
                read(fp, buffer,10);
                buffer[10]='\0';
                printf("Child read: \n");
```

```
                //printf(" child read again %s", buffer);
                puts(buffer);
                printf("Child exiting \n");


        }
        else
        {
                sleep(20);
                read(fp, buffer, 10);
                buffer[10]='\0';
                printf("parent reading \n");
                puts(buffer);
                printf("parent exiting \n");
        }

}
```

create a file abc, put some random data in it..
Output:

- High level file functions: fopen fread ftell fseek

```
#include<fcntl.h>
#include<stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
void main()
{
        FILE *fp;
        char buffer[10];
        int pid;
     fp=fopen("abc", "r");
        pid = fork();
        if (pid == 0)
        {
                printf("initial file pointer location %ld \n", ftell(fp));
                fread(buffer, sizeof(buffer),1,fp);
                buffer[10]='\0';
                printf("Child read: %s \n", buffer);
                printf("After child read, file pointer location %ld \n", ftell(fp));
                printf("Child exiting \n");


        }
        else
        {
                sleep(20);
                printf("initial parent file pointer location %ld \n", ftell(fp));
```

```
                fread(buffer, sizeof(buffer),1,fp);
                buffer[10]='\0';
                printf("parent read: %s \n", buffer);
                printf("After parent read, file pointer location %ld \n", ftell(fp));
                printf("parent exiting \n");
        }

}
```

THE EXEC FUNCTION:

how to compile C program in linux.

gcc -o <object filename> <source file name>
ex: gcc -o p1 p1.c

- small program is simple and more beautiful. To develop complicated program there is a mechanism to call other program from current program. We can chain many small programs and develop the complicated program.

.
Execute a C program from another program.

P1.c

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

void main()
{
        printf("before exec my id is %d \n", getpid());
        printf("my parent id is %d \n", getppid());
        printf("exec starts \n");
        execl("/home/ritesh-gce/Desktop/2020/os/operating-system/p2", "p2", (char *)0);
        printf("this is not printed");
}
```

p2.c

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

        void main()
        {
                printf("after exec my id is %d", getpid());
                printf("parent id is %d", getppid());
                printf("exec ends \n");

        }
```

execute one by one: gcc -o p2 p2.c, gcc p1.c, ./a.out
Output:

```
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc exec_p1.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ ./a.out
before exec my id is 27732
my parent id is 19156
exec starts
after exec my id is 27732parent id is 19156exec ends
```

- p2 is called from p1.
- Last line of program p1 will never executed. When p2 is called it is loaded in the memory where p1 was placed. So control will never come to p1.
- Second parameter passed to execl can be divided. It can be considered as the parameters for function called through execl function.
- The execution of p1 results into the execution of p2 in same memory space, same process is in the execution for p1 and p2.
- The second parameter of execl function can be of more than one length. Below is the program:

Main function arguments: argc argv

exec_p1_2.c

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

void main(int argc, char * argv[])
{
        printf("after exec my id is %d", getpid());
        printf("parent id is %d", getppid());
        printf("Child is %s and its arguments are : %s %s", argv[0], argv[1], argv[2]);
        printf("exec ends \n");

}
```

p2_2.c

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

void main(int argc, char * argv[])
{
        printf("after exec my id is %d", getpid());
        printf("parent id is %d", getppid());
        printf("Child is %s and its arguments are : %s %s", argv[0], argv[1], argv[2]);
        printf("exec ends \n");

}
```

How to run:
gcc -o p2_2 p2_2.c
gcc -o p1.out exec_p1_2.c

./p1.out /home/ritesh-gce/Desktop/2020/os/operating-system/p2_2 p2_2 Hello Unix

```
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc -o p2_2 p2_2.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ ./p1.out /home/ritesh-gce/Desktop/2020/os/operating-system/p2_2 p2_2 Hello Unix
before exec my id is 23303
my parent id is 19156
exec starts
after exec my id is 23303parent id is 19156Child is p2_2 and its arguments are : Hello Unixexec ends
```

The execv() and execvp() Function.
execv(): In above example we have hard coded the parameter passed. If we want to change the parameters we have to use execv() function. execv takes only two arguments, program we want to execute and array of pointers that hold all the argumets.

execv.c

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

void main()

{

        char *temp[3];
        temp[0] = "ls";
        temp[1] = "-l";
        temp[2] = (char *) 0;
        execv("/bin/ls", temp);
        printf("this is not printed");
}
```

```
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc execv.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ ./a.out
total 992
-rw-r--r-- 1 ritesh-gce ritesh-gce     49 Sep  5  2019  abc
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8400 Apr  3 21:20  a.out
-rw-r--r-- 1 ritesh-gce ritesh-gce    341 Apr  3 15:04  exec_p1_2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    296 Mar 31 16:21  exec_p1.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    192 Apr  3 21:20  execv.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    542 Mar 27 18:40  filesharing.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    791 Mar 30 13:49  filesharing_new.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    363 Aug 29  2019  fork1.c
drwxrwxr-x 3 ritesh-gce ritesh-gce   4096 Mar 28 12:50  linux
-rw-r--r-- 1 ritesh-gce ritesh-gce 336392 Mar 28 17:18  linux_concepts.odt
-rw-r--r-- 1 ritesh-gce ritesh-gce 189948 Mar 28 17:25  linux_concepts.pdf
-rw-r--r-- 1 ritesh-gce ritesh-gce    343 Jun 28  2019  orphan.c
drwxr-xr-x 3 ritesh-gce ritesh-gce   4096 Jan  3 15:21  os_bash_lab
-rw-r--r-- 1 ritesh-gce ritesh-gce  98680 Apr  3 16:37  OS_concepts.odt
-rw-r--r-- 1 ritesh-gce ritesh-gce  36949 Mar 28 17:26  OS_concepts.pdf
-rw-r--r-- 1 ritesh-gce ritesh-gce  19781 Nov 30 05:29 'OS lab manual.odt'
-rw-r--r-- 1 ritesh-gce ritesh-gce 204833 Mar 28 17:04 'OS lab manual.pdf'
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8480 Apr  3 15:04  p1.out
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8432 Mar 31 16:21  p2
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8432 Apr  3 15:09  p2_2
-rw-r--r-- 1 ritesh-gce ritesh-gce    281 Apr  3 15:09  p2_2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    176 Apr  1 17:13  p2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    513 Aug 29  2019  shareing.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    662 Jul  7  2019  texput.log
-rw-r--r-- 1 ritesh-gce ritesh-gce    107 Jul  6  2019  zombie.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$
```

The execv() function takes only two parameters, the program we want to execute and the array of pointers that holds all the parameter we want to passed.

Execvp() function

execvp.c

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

void main()

{

char *temp[4];
temp[0] = "/home/ritesh-gce/Desktop/2020/os/operating-system/p22";
temp[1] = "hello";
temp[2]= "unix";
temp[3] = (char *) 0;
printf("parent id is %d \n", getpid());
printf("parent id is %d \n", getppid());
execvp(temp[0], temp);
printf("this is not printed");
}
```

p2_2.c

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

void main(int argc, char * argv[])
{
printf("after exec my id is %d", getpid());
printf("parent id is %d", getppid());
printf("Child is %s and its arguments are : %s %s", argv[0], argv[1], argv[2]);
printf("exec ends \n");

}
```

output:

```
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc -o p22 p2_2.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc execvp.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ ./a.out
parent id is 19807
parent id is 19156
after exec my id is 19807parent id is 19156Child is /home/ritesh-gce/Desktop/2020/os/operating-system/p22 and its arguments are : hello unixexec ends
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$
```

With execv(), the first argument is a path to the executable.

With execvp(), the first argument is a filename. It must be converted to a path before it can used. This involves looking for the filename in all of the directories in the PATH environment variable.

Execvp.c

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

void main()
```

```
{
        char *temp[3];
        temp[0] = "";
        temp[1] = "-l";
        temp[2] = (char *) 0;
        printf("parent id is %d \n", getpid());
        printf("parent id is %d \n", getppid());
        execvp("ls", temp );
        printf("this is not printed");
}
```

```
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ gcc execvp.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$ ./a.out
parent id is 27448
parent id is 19156
total 1212
-rw-r--r-- 1 ritesh-gce ritesh-gce     49 Sep  5  2019  abc
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8488 Apr  5 10:04  a.out
-rw-r--r-- 1 ritesh-gce ritesh-gce    341 Apr  3 15:04  exec_p1_2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    296 Mar 31 16:21  exec_p1.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    192 Apr  3 21:20  execv.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    270 Apr  5 10:04  execvp.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    542 Mar 27 18:40  filesharing.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    791 Mar 30 13:49  filesharing_new.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    363 Aug 29  2019  fork1.c
drwxrwxr-x 3 ritesh-gce ritesh-gce   4096 Mar 28 12:50  linux
-rw-r--r-- 1 ritesh-gce ritesh-gce 336392 Mar 28 17:18  linux_concepts.odt
-rw-r--r-- 1 ritesh-gce ritesh-gce 189948 Mar 28 17:25  linux_concepts.pdf
-rw-r--r-- 1 ritesh-gce ritesh-gce    343 Jun 28  2019  orphan.c
drwxr-xr-x 3 ritesh-gce ritesh-gce   4096 Jan  3 15:21  os_bash_lab
-rw-r--r-- 1 ritesh-gce ritesh-gce 322926 Apr  4 16:41  OS_concepts.odt
-rw-r--r-- 1 ritesh-gce ritesh-gce  36949 Mar 28 17:26  OS_concepts.pdf
-rw-r--r-- 1 ritesh-gce ritesh-gce  19781 Nov 30 05:29 'OS lab manual.odt'
-rw-r--r-- 1 ritesh-gce ritesh-gce 204833 Mar 28 17:04 'OS lab manual.pdf'
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8480 Apr  3 15:04  p1.out
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8432 Mar 31 16:21  p2
-rwxr-xr-x 1 ritesh-gce ritesh-gce   8432 Apr  4 16:36  p22
-rw-r--r-- 1 ritesh-gce ritesh-gce    281 Apr  4 16:35  p2_2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    176 Apr  1 17:13  p2.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    513 Aug 29  2019  shareing.c
-rw-r--r-- 1 ritesh-gce ritesh-gce    662 Jul  7  2019  texput.log
-rw-r--r-- 1 ritesh-gce ritesh-gce    107 Jul  6  2019  zombie.c
ritesh-gce@riteshgce-Vostro-3578:~/Desktop/2020/os/operating-system$
```

- execvp() uses PATH environmental variable to execute the commands.

The exec() function called through a fork().