Operating System Lab Manual.

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");
        }

}
```