Write a C program to create child process and allow parent process to display parent and the child to display child on the screen?

Aim: Write a C program to create child process and allow parent process to display "parent" and the child to display "child" on the screen

Algorithm:

Step 1: start
Step2: call the fork() function to create a child
process fork function returns 2 values
step 3: which returns 0 to child process
step 4:which returns process id to the parent
process step 5:stop

Program

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
int pid,pid1,pid2;
        pid=fork();
        if(pid==-1)
        {
                printf("ERROR IN PROCESS CREATION \n");
                exit(0);
        }
        if(pid!=0)
        {
                pid1=getpid();
                printf("\n the parent process ID is %d", pid1);
        }
        else
        {
                pid2=getpid();
        }       printf("\n the child process ID is %d\n", pid2);
}
```

Output:

```
the child process ID is
4485 the parent process
ID is 4484
```

Write a C program to create zombie process?

Aim: Write a C program to create zombie process

Algorithm: Step 1:call fork function to create a child process
Step 2:if fork()>0
Then creation of Zombie
By applying sleep function for 10 seconds
Step 3: now terminate the child process
Step 4: exit status child process not reported to parent
Step 5: status any process which is zombie can known by
Applying ps(1) command
Step 6: stop

Program file name: 8b.c

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
int pid;
        pid=fork();
        if(pid == 0)
        {       printf("Iam child my pid is %d\n",getpid());
                printf("My parent pid is:%d\n",getppid());
                exit(0);
        }
        else
        {       printf("I am parent, my pid is %d\n",getpid());
                sleep(100);
                exit(0);
        }
}
```

Output:

```
Iam child my pid is 4732
My parent pid is:4731
I am parent, my pid is 4731
```

Checking for zombie process. Z means zombie process

Second terminal

```
[root@dba ~]# ps -el|grep a.out
0 S     0  4731  4585  0  77   0 -   384 -     pts/3    00:00:00 a.out
1 Z     0  4732  4731  0  77   0 -     0 exit  pts/3    00:00:00 a.out <defunct>
```

Write a C program to illustrate how an orphan process is created?

Aim:-Write a C program to illustrate how an orphan process is created

Algorithm:

Step 1: call the fork function to create the child process
Step 2:if (pid==0)
Then print child id and parent id
else goto step 4
Step 3:Then sleep(10)
Print child id and parent id
Step 4: Print child id and parent id
Step 5:which gives the information of orphan process
Step 6:stop

Program

```
#include <stdio.h>
#include<stdlib.h>
int main()
 {int pid;
        printf("I am the original process with PID %d and PPID %d\n",getpid(),getppid());
        pid=fork();
        if(pid == 0)
        {       printf("I am child, my pid is %d ",getpid());
                printf("My Parent pid is:%d\n",getppid());
                sleep(10);
                 printf("Now my pid is %d ",getpid());
                printf("My parent pid is:%d\n",getppid());
                exit(0);
        }
        else
        {       sleep(10);
                printf("I am parent, my pid is %d\n",getpid());
                //printf("I am going to die\n");
        }
printf("PID:%d terminates...\n",getpid());
}
```

Output

```
I am the original process with PID 5960 and PPID 5778
I am child, my pid is 5961 My Parent pid is:5960
I am parent, my pid is 5960
PID:5960 terminates...
[root@dba ~]# Now my pid is 5961 My parent pid is:1
```

Write a C program that illustrate communication between two process using unnamed pipes?

Aim:- Write a C program that illustrate communication between two process using unnamed pipes

Program file name: unnamed_pipe.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<fcntl.h>
void server(int,int);
void  client(int,int);
int main()
{
int p1[2],p2[2],pid;
        pipe(p1);
        pipe(p2);
        pid=fork();
        if(pid==0)
        {
         close(p1[1]);
        close(p2[0]);
        server(p1[0],p2[1]);
        return 0;
 }
        close(p1[0]);
        close(p2[1]);
        client(p1[1],p2[0]);
        wait();
return 0;
}


void client(int wfd,int rfd)
{
int i,j,n;
 char fname[2000];
 char buff[2000];
 printf("ENTER THE FILE NAME :");
 scanf("%s",fname);
 printf("CLIENT SENDING THE REQUEST .... PLEASE WAIT\n");
sleep(10);
write(wfd,fname,2000);
n=read(rfd,buff,2000);
buff[n]='\0';
printf("THE RESULTS OF CLIENTS ARE ..... \n");
write(1,buff,n);
}
```

```
void server(int rfd,int wfd)
 {
        int i,j,n;
       char fname[2000];
       char buff[2000];
        n=read(rfd,fname,2000);
        fname[n]='\0';
        int fd=open(fname,O_RDONLY);
        sleep(10);
        if(fd<0)
                write(wfd,"can't open",9);
        else
                n=read(fd,buff,2000);
        write(wfd,buff,n);
}
```

Write a C program that receives a message from message queue and display them?

Aim:-Write a C program that receives a message from message queue and display them

Algorithm:

Step 1:Start
Step 2:Declare a message queue structure

```
typedef struct msgbuf {
        long mtype;
        char mtext[MSGSZ];
         } message_buf;
```

Mtype =0 Retrieve the next message on the queue, regardless of its mtype.
PositiveGet the next message with an mtype equal to the specified
msgtyp.
Negative Retrieve the first message on the queue whose mtype fieldis
less than or equal to the absolute value of the msgtyp argument.
Usually mtype is set to1
mtext is the data this will be added to the queue.
Step 3:Get the message queue id for the "name" 1234, which was created by the server
key = 1234
Step 4 : if ((msqid = msgget(key, 0666< 0) Then print error
The msgget() function shall return the message queue identifier associated with the argument key.
Step 5: Receive message from message queue by using msgrcv function

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int    msgflg);
#include < sys/msg.h>
(msgrcv(msqid, &rbuf, MSGSZ, 1, 0)
```

msqid: message queue id
&sbuf: pointer to user defined structure MSGSZ: message size
Message type: 1
Message flag:The msgflg argument is a bit mask constructed by ORing together zero or
more of the following flags: IPC_NOWAIT or MSG_EXCEPT or MSG_NOERROR
Step 6:if msgrcv <0 return error
Step 7:otherwise print message sent is sbuf.mext
Step 8:stop

Message Send

Program:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXSIZE     128
void die(char *s)
{
 perror(s);
 exit(1);
}



    typedef struct msgbuf
    {
       long    mtype;
       char    mtext[MAXSIZE];
    };

    main()
    {
       int msqid;
       int msgflg = IPC_CREAT | 0666;
       key_t key;
       struct msgbuf sbuf;
       size_t buflen;

       key = 1234;

       if ((msqid = msgget(key, msgflg )) < 0) //Get
    the message queue ID for the given key
          die("msgget");

       //Message Type
       sbuf.mtype = 1;

       printf("Enter a message to add to message
    queue : ");
       scanf("%[^\n]",sbuf.mtext);
       getchar();

       buflen = strlen(sbuf.mtext) + 1 ;


    if (msgsnd(msqid, &sbuf, buflen,
IPC_NOWAIT) < 0)
    {
       printf ("%d, %d, %s, %d\n", msqid,
sbuf.mtype, sbuf.mtext, buflen);
       die("msgsnd");
    }

    else
       printf("Message Sent\n");

    exit(0);
}
```

Message Received

Program:

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


#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     128

void die(char *s)
{
  perror(s);
  exit(1);
}

typedef struct msgbuf
{
    long    mtype;
    char    mtext[MAXSIZE];
} ;
main()
{
    int msqid;
    key_t key;
    struct msgbuf rcvbuffer;

    key = 1234;

    if ((msqid = msgget(key, 0666)) < 0)
      die("msgget()");

    //Receive an answer of message type 1.
if (msgrcv(msqid, &rcvbuffer, MAXSIZE, 1, 0) < 0)
      die("msgrcv");

    printf("%s\n", rcvbuffer.mtext);
    exit(0);
}
```