**To copy from one file to another.**

#include<stdio.h>

FILE \*fs,\*ft;

void main(int argc,char \*argv[])

{

char c;

if(argc!=3)

{

printf("Invalid numbers of arguments.");

exit(1);

}

fs=fopen(argv[1],"r");

if(fs==NULL)

{

printf("Can't find the source file.");

exit(1);

}

ft=fopen(argv[2],"w");

if(ft==NULL)

{

printf("Can't open target file.");

fclose(fs);

exit(1);

}

fork();

rdwrt();

fclose(fs);

fclose(ft);

exit(0);

}

void rdwrt()

{

for(;;)

{

if(read(fdrd,&c,1) !=1))

return;

write(fdwt,&c,1);

}

}

**Signal handling**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

void sigcatcher()

{

printf("Caught signal and pid is %d \n", getpid());

signal(SIGINT, sigcatcher);

// printf("Caught signal %d, coming out...\n");

}

int main () {

int ppid;

signal(SIGINT, sigcatcher);

if(fork()==0)

{

sleep(5);

ppid = getppid();

for(;;)

if(kill(ppid,SIGINT)==-1)

exit();

}

nice(10);

for(;;)

}

**Example of exit**

main()

{

int child;

if((child=fork())==0)

{

printf(“child PID %d \n”, getpid());

pause();

}

printf(“child PID %d \n “ ,child);

exit(child);

}

**example of ignoring death of child**

#include < signal.h>

main (int argc, char \*argv)

{

int i, ret\_val, ret\_code;

if (argc > = 1 )

signal (SIGCLD, SIG\_IGN) ;

for (i = 0; i < 15; i++)

if (fork() == 0)

{

/\* ignore death of children \*/

/\* child proc here \*/

printf("child proc %x \n", getpid() ) ;

exit (i) ;

}

ret\_val = wait ( &ret\_code) ;

printf(“wait ret\_val %x ret\_code %x \n”, ret\_val,ret\_code);

}

**depicting reason for death of child**

#include <signal.h>

main (argc, argv)

{

char buf[256];

if (argc !=1)

signal (SIGCLD, SIG\_IGN);

while (read(O, buf, 256) )

if (fork()== 0)

{

/\* ignore death of children \*/

/\* child proc here typically does something with buf \*/

exit(0);

}

**multiple fork commands**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// Driver code

int main()

{

    int pid, pid1, pid2;

    // variable pid will store the

    // value returned from fork() system call

    pid = fork();

    if (pid == 0) {

        // First child needs to be printed

        // later hence this process is made

        // to sleep for 3 seconds.

        sleep(3);

        // This is first child process

        // getpid() gives the process

        // id and getppid() gives the

        // parent id of that process.

        printf("child[1] --> pid = %d and ppid = %d\n",

               getpid(), getppid());

    }

    else {

        pid1 = fork();

        if (pid1 == 0) {

            sleep(2);

            printf("child[2] --> pid = %d and ppid = %d\n",

                   getpid(), getppid());

        }

        else {

            pid2 = fork();

            if (pid2 == 0) {

                // This is third child which is

                // needed to be printed first.

                printf("child[3] --> pid = %d and ppid = %d\n",

                       getpid(), getppid());

            }

            // If value returned from fork()

            // in not zero and >0 that means

            // this is parent process.

            else {

                // This is asked to be printed at last

                // hence made to sleep for 3 seconds.

                sleep(3);

                printf("parent --> pid = %d\n", getpid());

            }

        }

    }

    return 0;

}

**TRY**

#include <unistd.h>

#include <stdio.h>

int main()

{

    // Creating first child

    int n1 = fork();

    // Creating second child. First child

    // also executes this line and creates

    // grandchild.

    int n2 = fork();

    if (n1 > 0 && n2 > 0)

    {

        printf("parent\n");

        printf("%d %d \n", n1, n2);

        printf(" my id is %d \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else if (n1 == 0 && n2 > 0)

    {

        printf("First child\n");

        printf("%d %d \n", n1, n2);

        printf("my id is %d  \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else if (n1 > 0 && n2 == 0)

    {

        printf("second child\n");

        printf("%d %d \n", n1, n2);

        printf("my id is %d  \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else {

        printf("third child\n");

        printf("%d %d \n", n1, n2);

        printf(" my id is %d \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    return 0;

}