1)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

int x=10;

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

pid\_t pid = fork();

if(pid==0){//child executes this

x+=1;

printf("x= %d\n",x);

printf("I am the child\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

else{//parent executes this

x+=2;

printf("x= %d\n",x);

printf("I am the parent\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

printf("Both execute this\n");

}

2)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

int x=10;

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

pid\_t pid = fork();

if(pid==0){//child executes this

x+=1;

printf("x= %d\n",x);

printf("I am the child\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

else{//parent executes this

wait(NULL);

x+=2;

printf("x= %d\n",x);

printf("I am the parent\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

printf("Both execute this\n");

}

3)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

if(execlp("/bin/ls","ls","-al",NULL)<0)

//nothing is executed after this point if exec is successful

printf("exec failed\n");

}

4)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

int x=10;

int status;

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

pid\_t pid = fork();

if(pid==0){//child executes this

x+=1;

printf("x= %d\n",x);

printf("I am the child\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

exit(3);

}

else{//parent executes this

wait(&status);

printf("the child exit status is %d\n",WEXITSTATUS(status));

x+=2;

printf("x= %d\n",x);

printf("I am the parent\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

printf("Both execute this\n");

5)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

int x=10;

int status;

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

pid\_t pid = fork();

if(pid==0){//child executes this

x+=1;

printf("x= %d\n",x);

printf("I am the child\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

exit(3);

}

else{//parent executes this

int p;

while((p=waitpid(pid,&status,WNOHANG))==0);

printf("The process with PID %d is finished\n",p);

printf("the child exit status is %d\n",WEXITSTATUS(status));

x+=2;

printf("x= %d\n",x);

printf("I am the parent\n");

printf("My PID is %d\n",getpid());

printf("My parent PID is %d\n",getppid());

}

printf("Both execute this\n");

}

6)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

// fork();//p and c1

// fork();//c2 and c1\_1

int i;

for(i=0;i<2;i++)

fork();

printf("My PID is %d\tMy parent PID is %d\n",getpid(),getppid());

}

/\*

i processes output

0 p, c1

1 c2, c1\_1

2 4 times

\*/

7)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

// fork();//p and c1

// fork();//c2 and c1\_1

int i;

for(i=0;i<2;i++){

fork();

printf("My PID is %d\tMy parent PID is %d\n",getpid(),getppid());

}

}

/\*

i processes output

0 p, c1 2 times

1 c2, c1\_1 4 times

2

\*/

8)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void main(){

char str[30];

int pfd[2],r,w;

if(pipe(pfd)<0){

printf("pipe failed\n");

exit(1);

}

if(!fork()){//child

close(pfd[0]);

w=write(pfd[1],"test",sizeof("test"));

}

else{//parent

close(pfd[1]);

r=read(pfd[0],str,sizeof(str));

printf("parent read %s from the pipe\n",str);

}

}

9)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

//ls -al | wc

//ls -al | sort | wc

//comm0 | comm1 | comm2 | comm3| ... | commn-1

//commk stdin is pipe k-1 and stout is pipe k

//exception for comm0 stdin is 0 stdout is pipe0

//exception for commn-1 stdin is pipe n-2 stdout is 1

//read string from user

//tokenize with "|". tokens are the commands saved in an array comm[] of size n

//create array of pipes of size n-1. pipe[][], this is a 2-D array. rows = n-1 col=2

// pipe[k][0] has file descriptor of the read end of pipe k

// pipe[k][1] has file descriptor of the write end of pipe k

//create a function to execute a command execute\_command()

//call execute\_command() n times inside a loop. each time to execute one command from comm[]

//execute\_command() receives the command to execute comm[k] and the index of the command 0, 1, ..., n-1

//in execute\_command() fork a child

//in execute\_command() close all pipes' ends. either using for or by calling a function to do so

//in execute\_command() for the child we created, change stdin and stdout based on the index of the command

//in execute\_command() execute the command using exec...

void main(){

//stdout is 1 and stdin is 0

int pfd[2];

if(pipe(pfd)<0){

printf("pipe failed\n");

exit(1);

}

if(!fork()){//child 1

//stdout is 1 and stdin is 0

//change stdout to pfd[1] instead of 1

//stdout is pfd[1]

// close(1);

// dup(pfd[1]);

dup2(pfd[1],1);

close(pfd[0]);close(pfd[1]);

execlp("/bin/ls","ls","-al",NULL);

printf("first exec failed\n");

}

else if(!fork()){//child 2

//stdout is 1 and stdin is 0

//change stdin to pfd[0] instead of 0

//stdin is pfd[0]

// close(0);

// dup(pfd[0]);

dup2(pfd[0],0);

close(pfd[0]);close(pfd[1]);

execlp("wc","wc",NULL);

printf("second exec failed\n");

}

//parent

close(pfd[0]);close(pfd[1]);

wait(NULL);wait(NULL);

}

10)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

//siganls

void quit\_handler(int siginfo){

printf("Received the signal SIGTERM. You can't quit me.\n");

}

void main(){

//[Ctrl-z] results in SIGTSTP

//[Ctrl-q] results in SIGQUIT

//[Ctrl-c] results in SIGINT

signal(SIGTSTP,SIG\_DFL);

signal(SIGQUIT,quit\_handler);

signal(SIGINT,SIG\_IGN);//Changes the handler of SIGINT in this process

printf("Press any of [Ctrl-z], [Ctrl-q],[Ctrl-c]\n");

while(1)

pause();//suspend until a signal is received

}

11)

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

//siganls

void int\_handler(int siginfo){

signal(SIGINT,int\_handler);

printf("Received the signal SIGTINT. You can't terminate me.\n");

}

void main(){

//[Ctrl-z] results in SIGTSTP

//[Ctrl-q] results in SIGQUIT

//[Ctrl-c] results in SIGINT

signal(SIGINT,int\_handler);//Changes the handler of SIGINT in this process

pid\_t pid = fork();

if(pid == 0){//child

signal(SIGTSTP,SIG\_DFL);

signal(SIGQUIT,SIG\_IGN);

// signal(SIGINT,int\_handler);//Changes the handler of SIGINT in this process

while(1)

pause();

}

else{//parent to send signals to the child

raise(SIGINT);//kill(getpid(),SIGINT);

sleep(1);//1 sec

kill(pid,SIGINT);

sleep(1);//1 sec

kill(pid,SIGINT);

sleep(1);//1 sec

kill(pid,SIGQUIT);

sleep(1);//1 sec

kill(pid,SIGTSTP);

}

}