Signal(single process)

Signal	When it is send
SIGINT	When Control C (CTRL C i.e. ^C) is pressed.
SIGQUIT	When Control slash (CTRL \ i.e. ^\) is pressed.
SIGFPE	When illegal mathematical operation such as division by zero is done.
SIGPIPE	When a process tries to write in a pipe with all read ends closed
SIGCHLD	When a child process terminates it send to its parent
SIGALRM	When a process calls alarm(t) then it is send to the process after time 't'
SIGKILL	When command kill <process id> is given
SIGUSR2	Open signal. Any process can send to any other process. (SIGUSR1 similar)

All signals (except following) terminate the process, to which the signal is sent.

SIGSTOP Stops execution.	**SIGCONT** Resumes execution of stopped process.	**SIGCHLD**

System Call	Operation
signal(signal name,SIG_IGN)	The corresponding signal is ignored. This does not work for SIGKILL.
signal(signal name,funtion name)	On the arrival of the signal the corresponding function is called. It is called catching the signal. It does not work for SIGKILL.
signal(signal name,SIG_DFL)	On the arrival of the signal the default action is taken.
kill(process id, signal name)	Corresponding signal is send to the process whose identifier is given. Logically it is equivalent to Send(process id, signal name)

Program A	Program B	Program C	Program D

#include<stdio.h>
#include<signal.h>
void k( )
{ printf("AA\n"); }
main( )
{ signal(SIGINT,k);
for(;;);
}
	void u( )
	{ printf("TT\n"); }
	main( )
	{ signal(SIGFPE,u);
	int a,b;
	scanf("%d",&a);
	b=3/a;
	}
		void g( )
		{ printf("AA\n");}
		main( )
		{ alarm(3);
		~~signal(SIGALRM,g);~~
		for(;;);
		}
			void a(){printf("%dX\n",getpid());}
			void b(){printf("%dY\n",getpid());}
			main( )
			{ printf("%d\n",getpid());int q=fork();
			if (q!=0) signal(SIGINT,a);
			else signal(SIGINT,b);
			for(;;);
			}

Program A: outputs AA whenever ^C is pressed. To terminate the above program use ^\ (CRTL \).

To count the number of ^C: int c=0; void k( ){ c++; printf("%d\n",c);}main( ) { signal(SIGINT,k); for(;;); }

First ^C prints AA. Second ^C terminates. void k( ){printf("AA\n"); signal(SIGINT,SIG_DFL);} main same

First ^C AA others BB void m( ){printf("BB\n");} void k( ){printf("AA\n"); signal(SIGINT,m);} main same

1. Write a program, which ignores 5 ^C's. On 6th ^C the program terminates.
2. On first ^C print PP, on 2nd QQ, 3rd PP, 4th QQ, 5th PP so on (A) On 1st ^C PP, on 2nd QQ on 3rd terminate.
3. When ^C is pressed, the program asks a question, whether you want to terminate. If user types 0 the program terminates. If user types 1 then the program does not terminate.
4. Prints X when ^C is pressed first time. Print XX when ^C is pressed second time. XXX on third and so on.
5. When ^C is pressed the program outputs its time difference with previous ^C. use sleep(1). (1st o/p garbage)
6. Write a program, which is printing Z's in an infinite loop. When ^C is pressed the program terminates.
7. Modify it. When ^C is pressed the program outputs Y and terminates. (A) outputs Y's after ^C.

Program B: For input 0(zero) the output is TT. 3.2/0 does not invoke signal.

8. Write a program which reads numbers in a loop and outputs its double. As soon as number less than 5 is given the program outputs sum of previous numbers. (no if) Assume all numbers are positive. No loop.

Program C: output Alarm Clock(terminates) is produced after 3 seconds, because a signal SIGALRM is sent to itself. ~~AA(not terminates)~~ after 3 second.

9. Modify it. Output AA is produced after every 3 seconds. (No sleep) (A) o/pAA at t=3,7,12,18,25,33….
10. When ^C is pressed output AA 3 seconds later (A) Program terminates 3 seconds after ^C is pressed.
11. Write program, which prints AAA…. for 1 second.
12. Write program, which prints 1 2 3 4 5 …. The next number is outputted after 1 sec. No sleep.

Program D: When ^C is pressed the parent prints X and child Y.

13. Parent prints X and child Y and Z alternatively. (one if)
14. Parent X and child X,Y alternatively (use 2 functions)(1 function)(no %c)