Date: 18-10-21

signal handling

→ signal is a notification to a process that an event has occured

→ signals are known as software interrupts

→ signals are similar to H/W interrupts in that they interrupt the normal flow of execution of program.

→ in general it is not possible to predict when signal arising most of the case.

(1) →

```
main()
{
printf("hello..\n");
int *p = 0;
printf("%d\n", *p);
}
```

$ cc p1.c

$ ./a.out

hello

↳ process receiving signal (SIGSEGV) ✓

↳ segmentation fault (11) (core dump)

(2)

```
main()
{
while (1)
printf("hello--\n");
}
```

hello
hello
hello
⋮

ctrl+c

ctrl+c ↳ signal ↳ (2)

ctrl+\ ↳ sigquit ↳ (3)

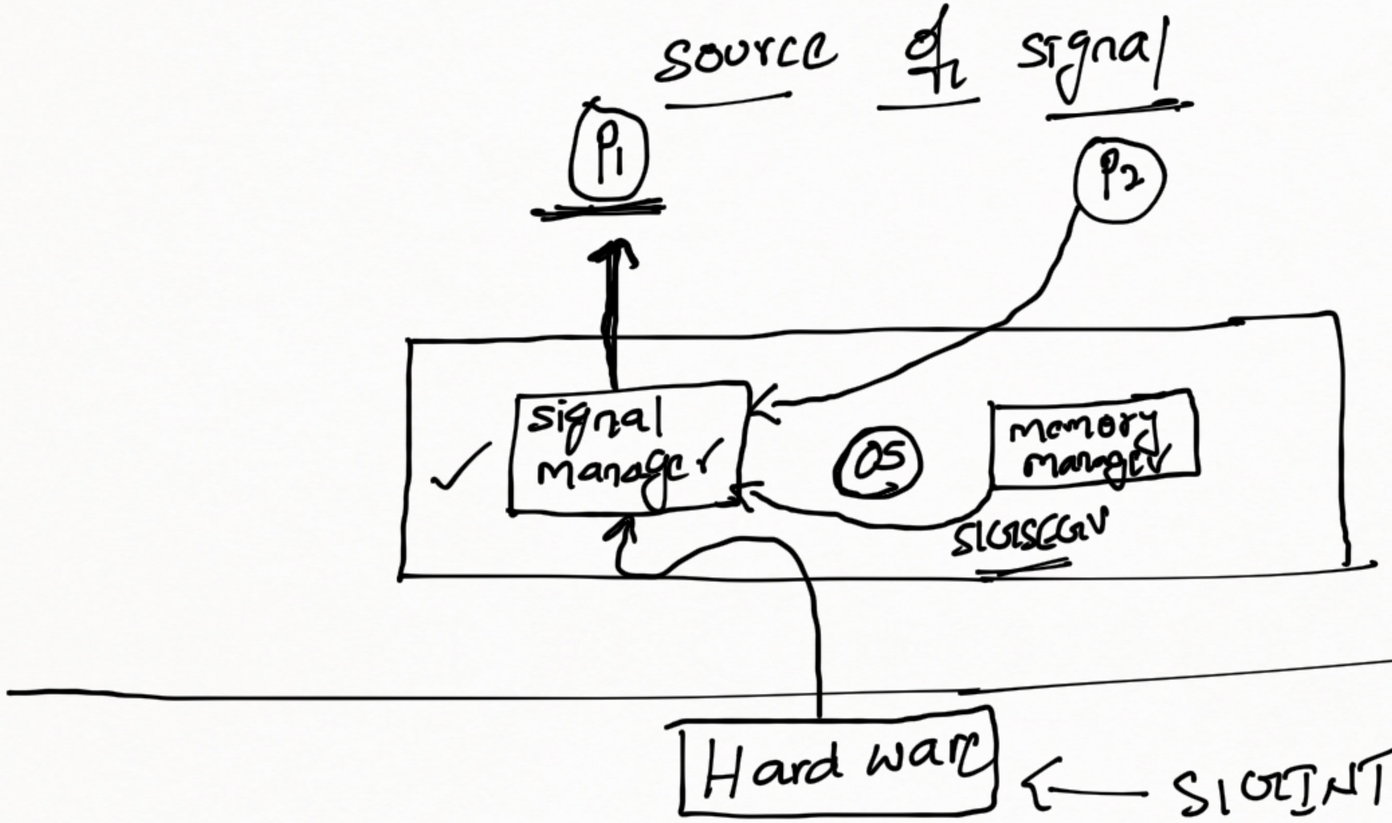
(3)

```
main()
{
    int a=10, b=0;
printf("hello--\n"); ✓
  int res;
   res = a/b;
printf("%d\n", res);
}
```

✓

→ floating point exception

SIGFPE (8)



→ upon delivery of a signal process carry out one of the following default actions.

① ignore the signal

② Termination of the process

③ coredump: A file coredump is generated and process terminated

① from the H/W ✓

② from the terminal ✓
$ kill -9 pid

③ from another process
ctrl+c

④ Stop: process stopped (suspended)

⑤ Continue: process continued (resume)

⇒ Instead of accepting default action programmer can change the action

sending and receiving signals

→ sending signals from one process to other process we can use kill() syscall

→ possible to send signal to the same process by calling raise() function

→ you can make process to sleep until any signal arrives using pause()

kill()
raise()
pause()

$ ./a.out signo pid

$ ./a.out 3 1258

o = kill(atoi(argv[2]), atoi(argv[1]))

-1 = kill(pid, signo)