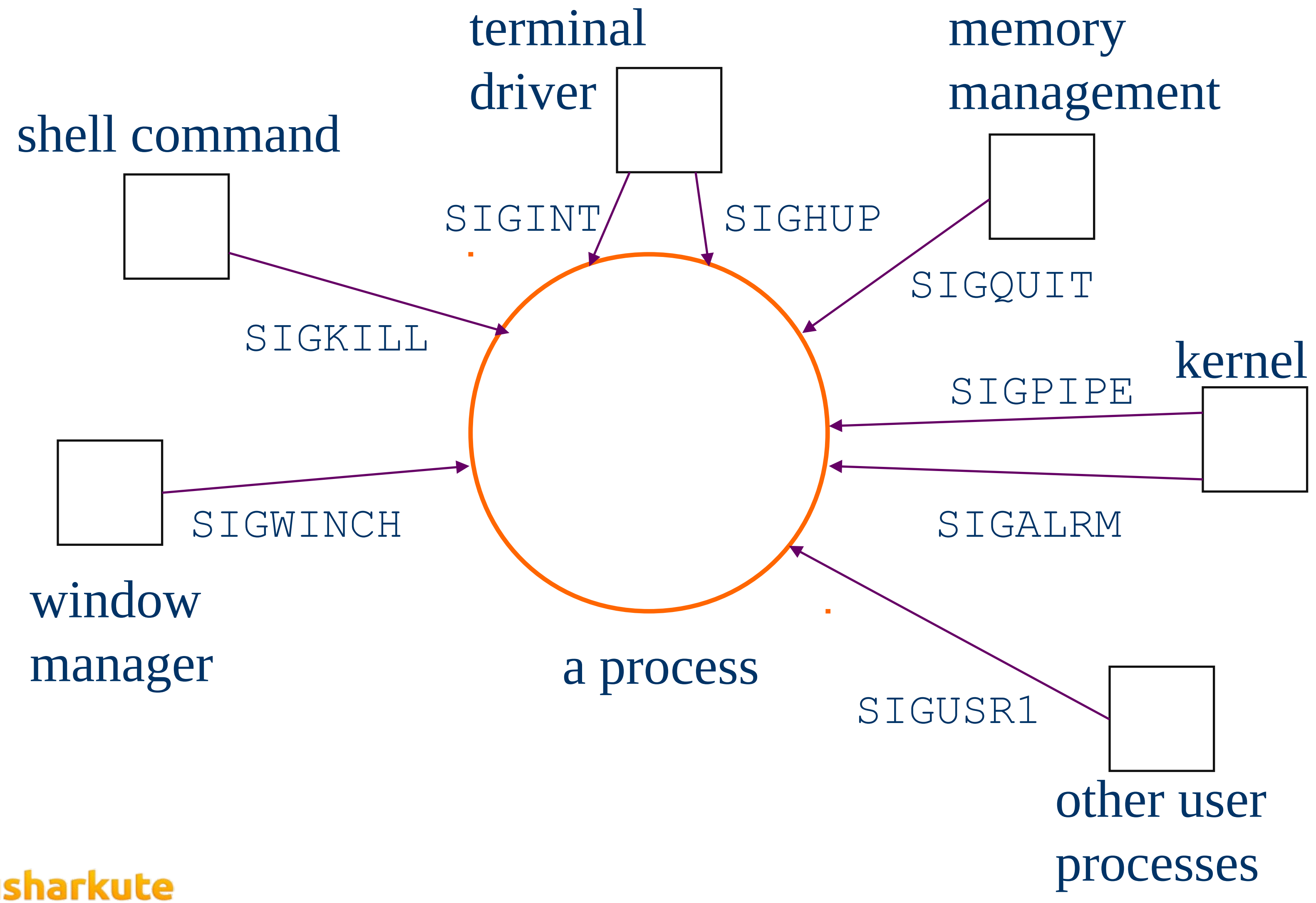
Signal Handling in Linux

Tushar B. Kute,
http://tusharkute.com

What is a Signal?

- A signal is an asynchronous event which is delivered to a process.
- Asynchronous means that the event can occur at any time may be unrelated to the execution of the process.
- Signals are raised by some error conditions, such as memory segment violations, floating point processor errors, or illegal instructions.
  - e.g. user types ctrl-C, or the modem hangs

Signal Sources



POSIX predefined signals

- SIGALRM: Alarm timer time-out. Generated by alarm( ) API.
- SIGABRT: Abort process execution. Generated by abort( ) API.
- SIGFPE: Illegal mathematical operation.
- SIGHUP: Controlling terminal hang-up.
- SIGILL: Execution of an illegal machine instruction.
- SIGINT: Process interruption. Can be generated by <Delete> or <ctrl_C> keys.
- SIGKILL: Sure kill a process. Can be generated by
  - “kill -9 <process_id>“ command.
- SIGPIPE: Illegal write to a pipe.
- SIGQUIT: Process quit. Generated by <crtl_\> keys.
- SIGSEGV: Segmentation fault. generated by de-referencing a NULL pointer.

POSIX predefined signals

- SIGTERM: process termination. Can be generated by
  - “kill <process_id>” command.
- SIGUSR1: Reserved to be defined by user.
- SIGUSR2: Reserved to be defined by user.
- SIGCHLD: Sent to a parent process when its child process has terminated.
- SIGCONT: Resume execution of a stopped process.
- SIGSTOP: Stop a process execution.
- SIGTTIN: Stop a background process when it tries to read from from its controlling terminal.
- SIGTSTP: Stop a process execution by the control_Z keys.
- SIGTTOUT: Stop a background process when it tries to write to its controlling terminal.

Actions on signals

- Process that receives a signal can take one of three action:
- Perform the system-specified default for the signal
  - notify the parent process that it is terminating;
  - generate a core file; (a file containing the current memory image of the process)
  - terminate.
- Ignore the signal
  - A process can do ignoring with all signal but two special signals: SIGSTOP and SIGKILL.
- Catch the Signal
  - When a process catches a signal, except SIGSTOP and SIGKILL, it invokes a special signal handing routine.

Example of signals

- **User types Ctrl-c**
  - Event gains attention of OS
  - OS stops the application process immediately, sending it a 2/SIGINT signal
  - Signal handler for 2/SIGINT signal executes to completion
  - Default signal handler for 2/SIGINT signal exits process
- **Process makes illegal memory reference**
  - Event gains attention of OS
  - OS stops application process immediately, sending it a 11/SIGSEGV signal
  - Signal handler for 11/SIGSEGV signal executes to completion
  - Default signal handler for 11/SIGSEGV signal prints “segmentation fault” and exits process

Signal Number

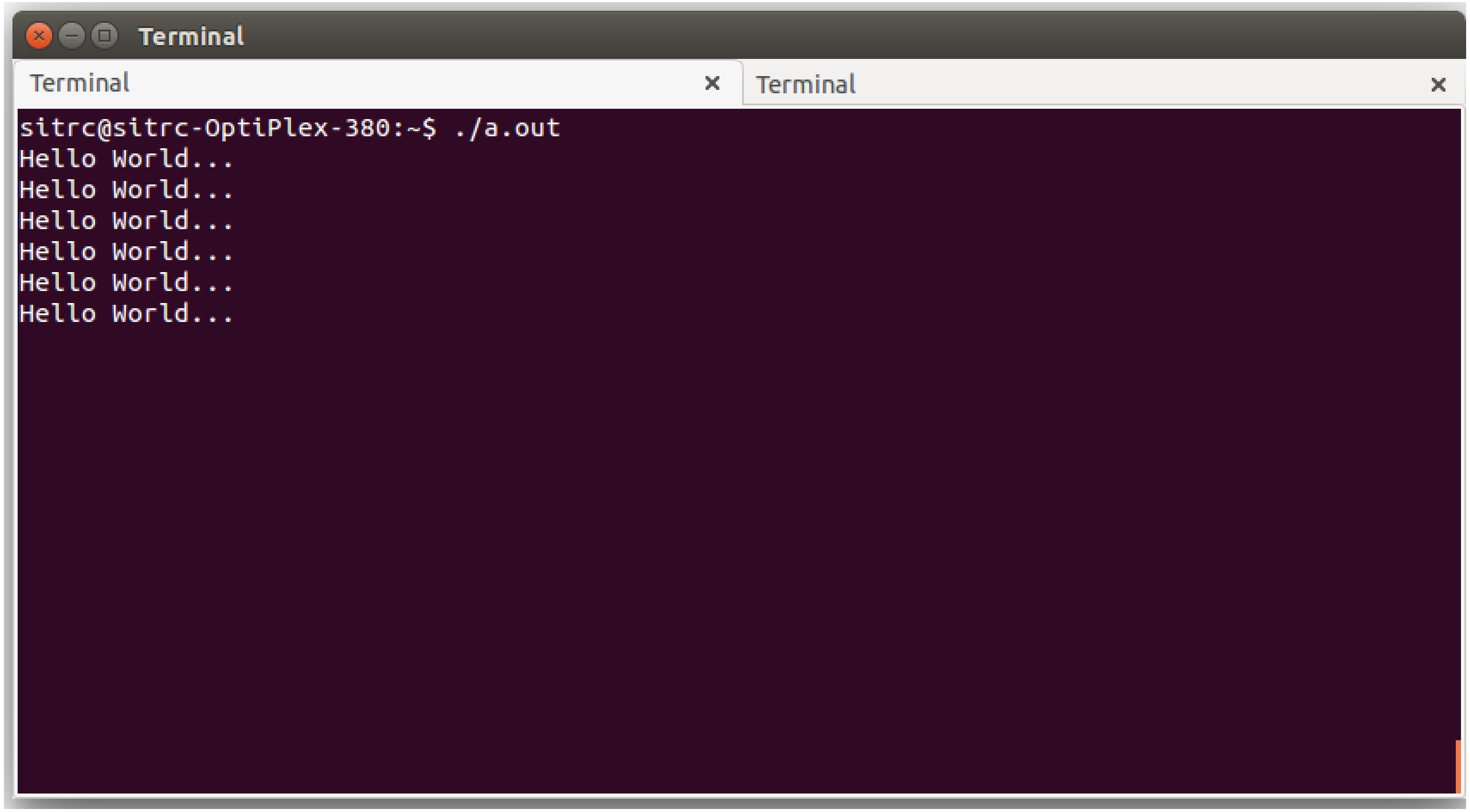
Send signals via commands

- **kill Command**
  - **`kill -signal pid`**
    - Send a signal of type signal to the process with id pid
    - Can specify either signal type name (-SIGINT) or number (-2)
  - *No signal type name or number specified => sends 15/SIGTERM signal*
- Default 15/SIGTERM handler exits process
  - Better command name would be sendsig
- Examples
  - **`kill –2 1234`**
  - **`kill -SIGINT 1234`**
    - Same as pressing Ctrl-c if process 1234 is running in foreground

Demonstration

```
#include<stdio.h>
int main()
{
  while(1)
    printf("Hello World...\n");
  return 0;
}
```

Check the output

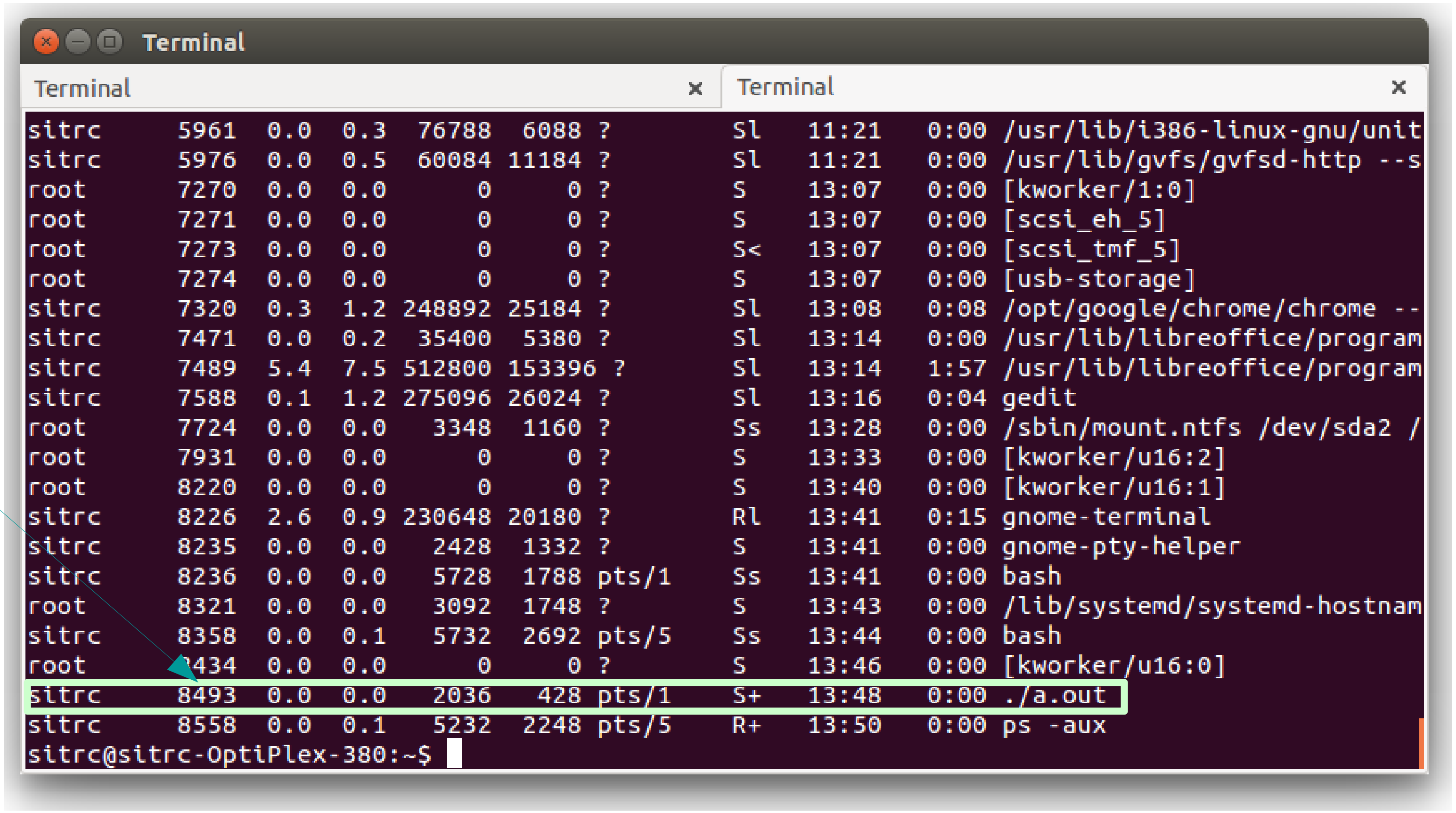


```
sitrc@sitrc-OptiPlex-380:~$ ./a.out
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
```

Check the output

- Go to new terminal and check the process list (`ps -aux`)

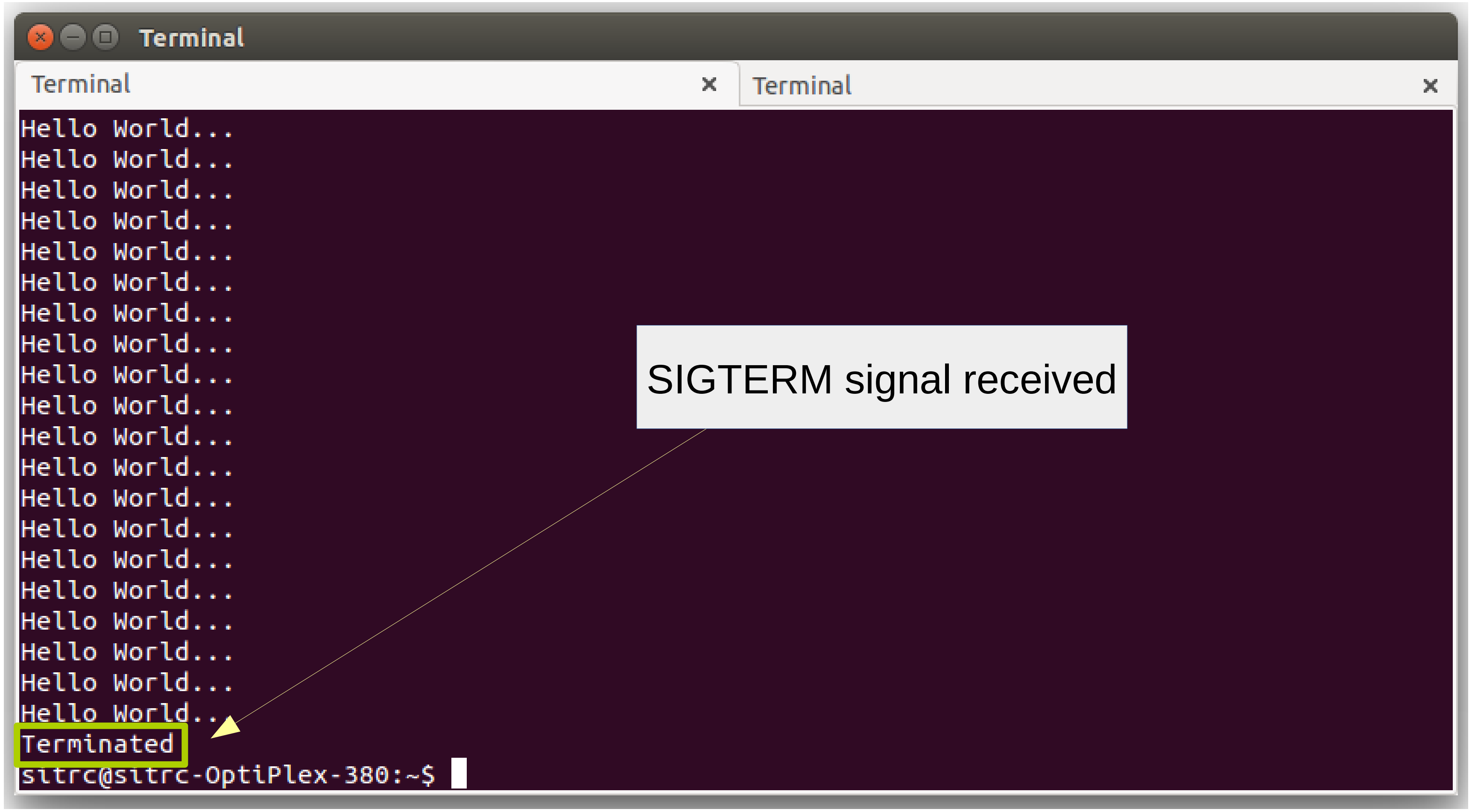
pid



```
sitrc     5961  0.0  0.3  76788  6088 ?        Sl   11:21   0:00 /usr/lib/i386-linux-gnu/unit
sitrc     5976  0.0  0.5  60084 11184 ?        Sl   11:21   0:00 /usr/lib/gvfs/gvfsd-http --s
root      7270  0.0  0.0      0     0 ?        S    13:07   0:00 [kworker/1:0]
root      7271  0.0  0.0      0     0 ?        S    13:07   0:00 [scsi_eh_5]
root      7273  0.0  0.0      0     0 ?        S<   13:07   0:00 [scsi_tmf_5]
root      7274  0.0  0.0      0     0 ?        S    13:07   0:00 [usb-storage]
sitrc     7320  0.3  1.2 248892 25184 ?        Sl   13:08   0:08 /opt/google/chrome/chrome --
sitrc     7471  0.0  0.2  35400  5380 ?        Sl   13:14   0:00 /usr/lib/libreoffice/program
sitrc     7489  5.4  7.5 512800 153396 ?       Sl   13:14   1:57 /usr/lib/libreoffice/program
sitrc     7588  0.1  1.2 275096 26024 ?        Sl   13:16   0:04 gedit
root      7724  0.0  0.0   3348  1160 ?        Ss   13:28   0:00 /sbin/mount.ntfs /dev/sda2 /
root      7931  0.0  0.0      0     0 ?        S    13:33   0:00 [kworker/u16:2]
root      8220  0.0  0.0      0     0 ?        S    13:40   0:00 [kworker/u16:1]
sitrc     8226  2.6  0.9 230648 20180 ?        Rl   13:41   0:15 gnome-terminal
sitrc     8235  0.0  0.0   2428  1332 ?        S    13:41   0:00 gnome-pty-helper
sitrc     8236  0.0  0.0   5728  1788 pts/1    Ss   13:41   0:00 bash
root      8321  0.0  0.0   3092  1748 ?        S    13:43   0:00 /lib/systemd/systemd-hostnam
sitrc     8358  0.0  0.1   5732  2692 pts/5    Ss   13:44   0:00 bash
root      8434  0.0  0.0      0     0 ?        S    13:46   0:00 [kworker/u16:0]
sitrc     8493  0.0  0.0   2036   428 pts/1    S+   13:48   0:00 ./a.out
sitrc     8558  0.0  0.1   5232  2248 pts/5    R+   13:50   0:00 ps -aux
sitrc@sitrc-OptiPlex-380:~$
```

Kill the process

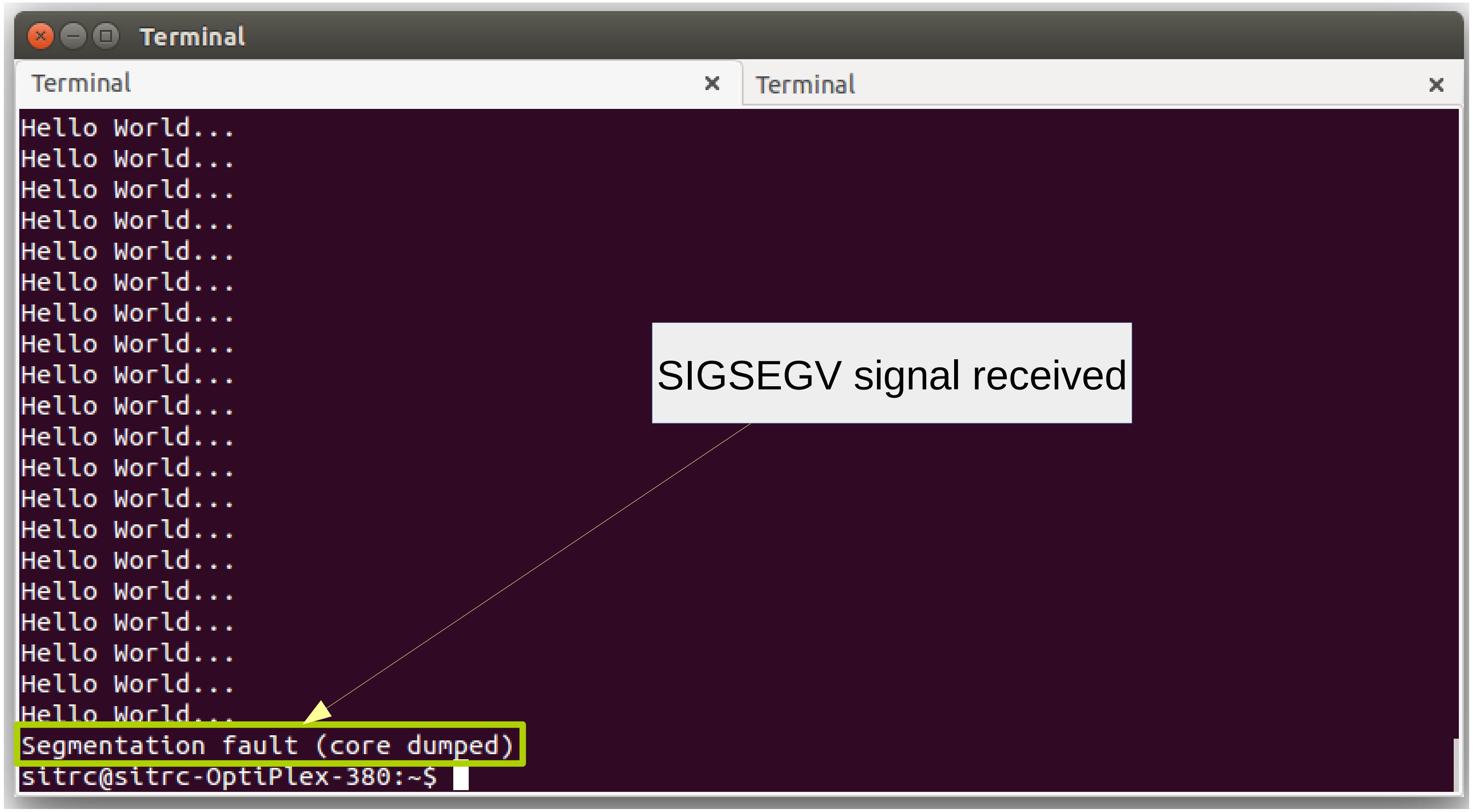
- `kill 8493`



```
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Terminated
sitrc@sitrc-OptiPlex-380:~$
```

Killing process by different signals

- `kill -SIGSEGV 8493`



```
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Hello World...
Segmentation fault (core dumped)
sitrc@sitrc-OptiPlex-380:~$
```

Signal Concepts

- Signals are defined in <signal.h>
- **man 7 signal** for complete list of signals and their numeric values.
- **kill –l** for full list of signals on a system.
- 64 signals. The first 32 are traditional signals, the rest are for real time applications

Signal Function

- Programs can handle signals using the signal library function.

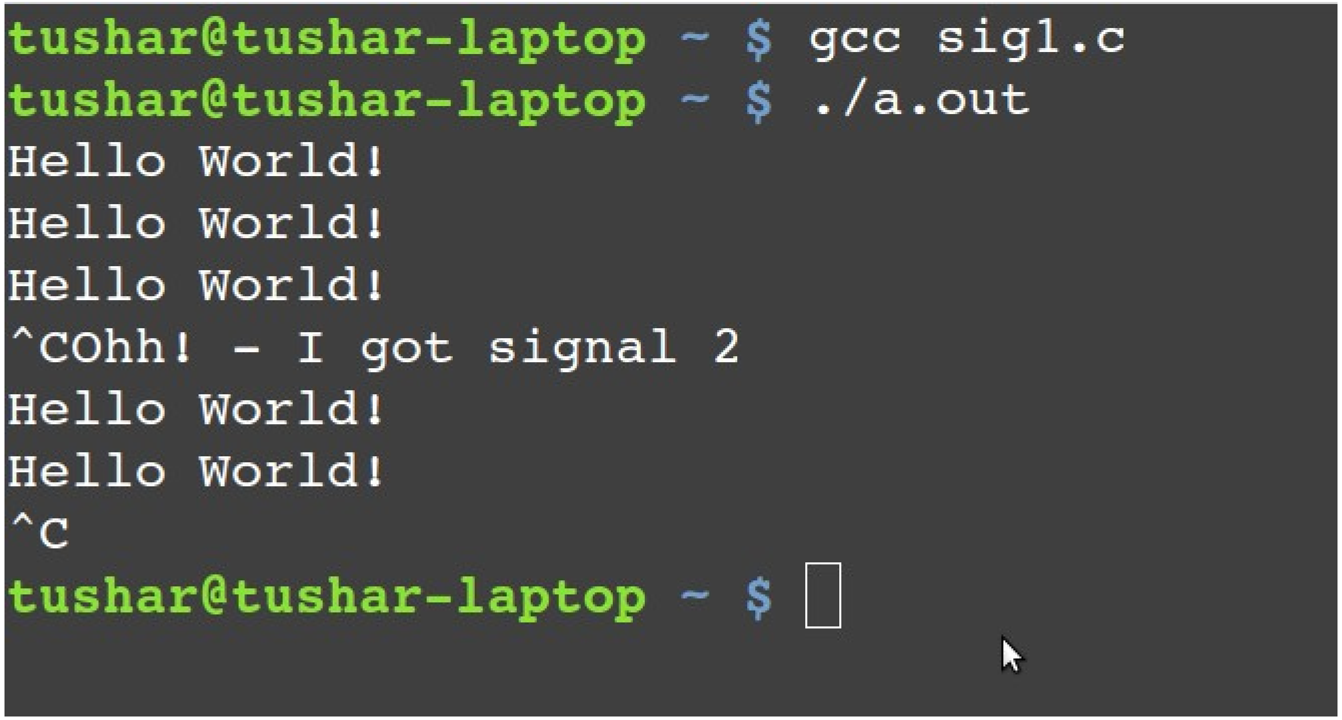
  void (*signal(int signo, void (*func)(int)))(int);

- signo is the signal number to handle
- func defines how to handle the signal
  - SIG_IGN
  - SIG_DFL
  - Function pointer of a custom handler
- Returns previous disposition if ok, or SIG_ERR on error

Example:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void ohh(int sig)
{
    printf("Ohh! - I got signal %d\n", sig);
    (void) signal(SIGINT, SIG_DFL);
}
int main()
{
    (void) signal(SIGINT, ohh);
    while(1)
    {
         printf("Hello World!\n");
         sleep(1);
    }
  return 0;
}
```

Output

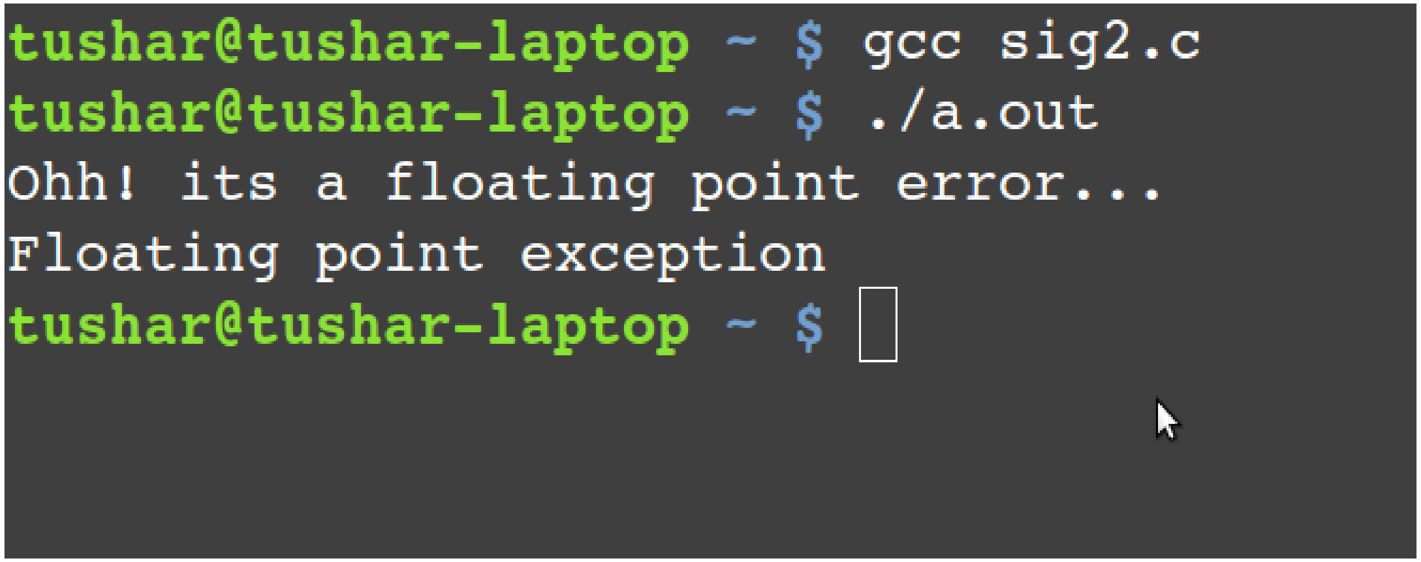


```
tushar@tushar-laptop ~ $ gcc sig1.c
tushar@tushar-laptop ~ $ ./a.out
Hello World!
Hello World!
Hello World!
^COhh! - I got signal 2
Hello World!
Hello World!
^C
tushar@tushar-laptop ~ $
```

Example:2

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void error(int sig)
{
     printf("Ohh! its a floating point error...\n");
     (void) signal(SIGFPE, SIG_DFL);
}
int main()
{
     (void) signal(SIGFPE, error);
     int a = 12, b = 0, result;
     result = a / b;
     printf("Result is : %d\n",result);
     return 0;
}
```

Output



```
tushar@tushar-laptop ~ $ gcc sig2.c
tushar@tushar-laptop ~ $ ./a.out
Ohh! its a floating point error...
Floating point exception
tushar@tushar-laptop ~ $ 
```

sigaction

- `int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);`
- The sigaction structure, used to define the actions to be taken on receipt of the signal specified by sig, is defined in signal.h and has at least the following members:

void (*) (int) sa_handler	**function, SIG_DFL or SIG_IGN**
sigset_t sa_mask	**signals to block in sa_handler**
int sa_flags	**signal action modifiers**

- The sigaction function sets the action associated with the signal sig . If oact is not null, sigaction writes the previous signal action to the location it refers to. If act is null, this is all sigaction does. If act isn’t null, the action for the specified signal is set.

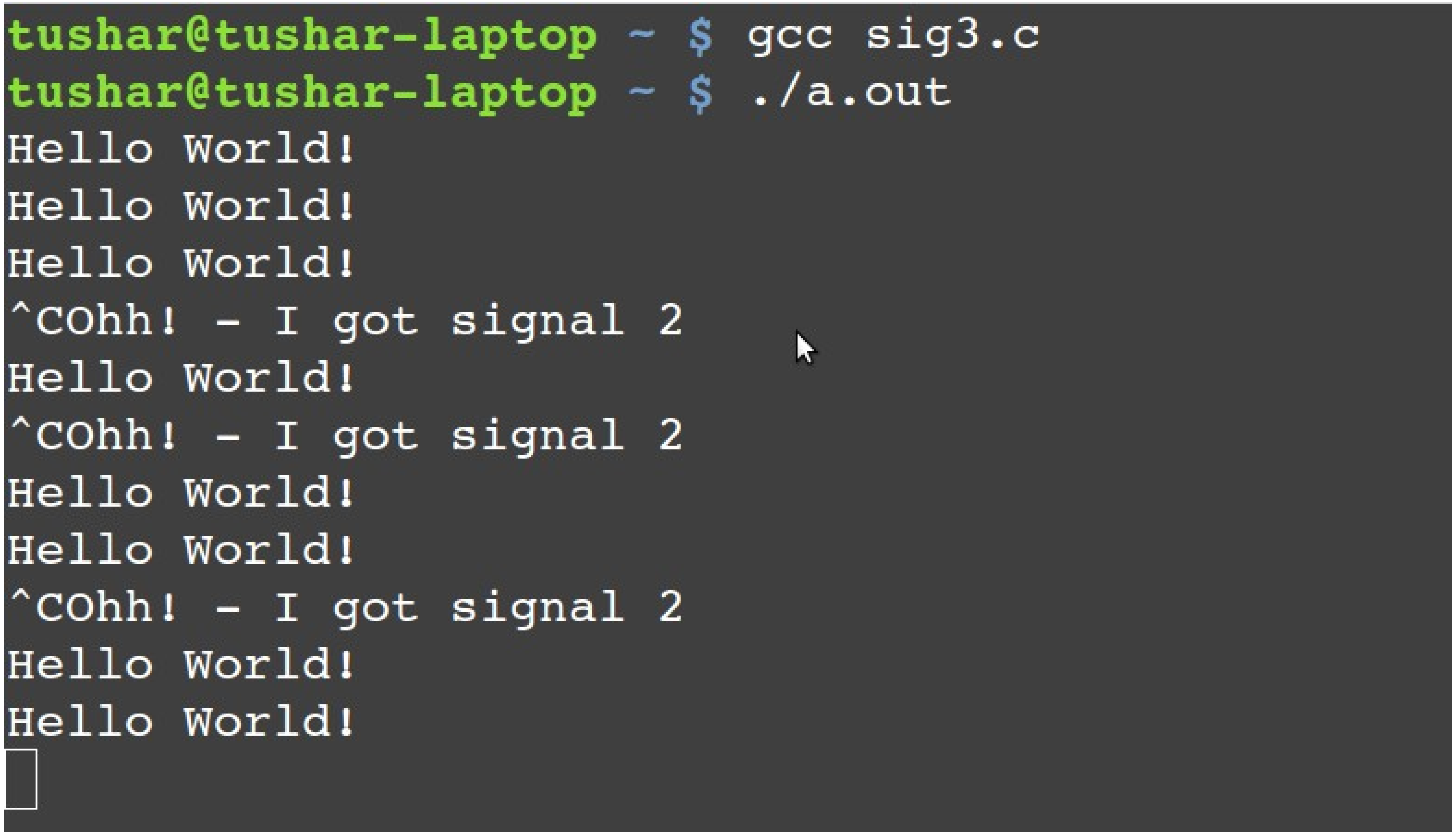
sigaction

- As with signal , sigaction returns 0 if successful and -1 if not. The error variable errno will be set to EINVAL if the specified signal is invalid or if an attempt is made to catch or ignore a signal that can't be caught or ignored.
- Within the sigaction structure pointed to by the argument act , sa_handler is a pointer to a function called when signal sig is received. This is much like the function func you saw earlier passed to signal .
- You can use the special values SIG_IGN and SIG_DFL in the sa_handler field to indicate that the signal is to be ignored or the action is to be restored to its default, respectively.

Example

```
void ohh(int sig)
{
    printf("Ohh! - I got signal %d\n", sig);
}
int main()
{
    struct sigaction act;
    act.sa_handler = ohh;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGINT, &act, 0);
    while(1)
    {
        printf("Hello World!\n");
        sleep(1);
    }
}
```

Output



```
tushar@tushar-laptop ~ $ gcc sig3.c
tushar@tushar-laptop ~ $ ./a.out
Hello World!
Hello World!
Hello World!
^COhh! - I got signal 2
Hello World!
^COhh! - I got signal 2
Hello World!
Hello World!
^COhh! - I got signal 2
Hello World!
Hello World!
```

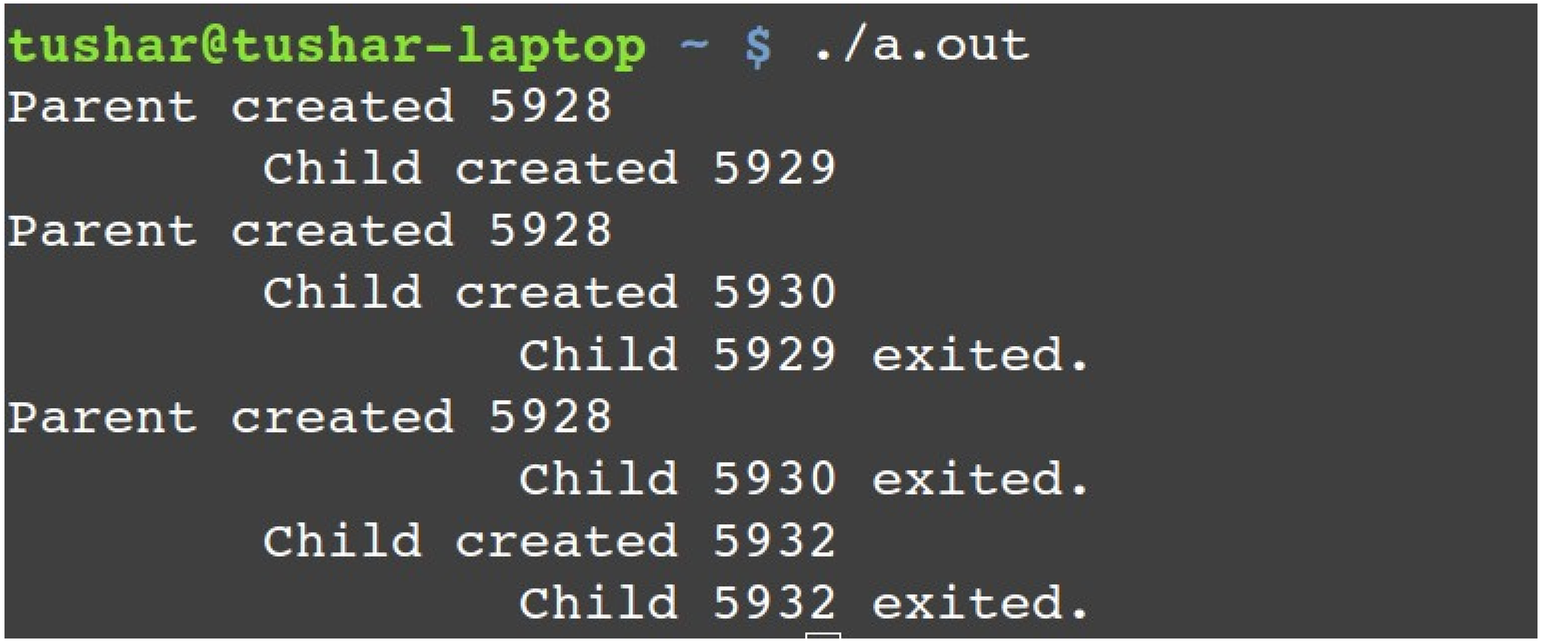
Problem Statement

- Implement the C program to demonstrate the use of SIGCHLD signal. A parent process Creates multiple child process (minimum three child processes). Parent process should be Sleeping until it creates the number of child processes. Child processes send SIGCHLD signal to parent process to interrupt from the sleep and force the parent to call wait for the Collection of status of terminated child processes.

Program

```c
void handler(int sig)
{
  pid_t pid;
  pid = wait(NULL);
  printf("\t\tChild %d exited.\n", pid);
  signal(SIGCHLD, handler);
}
int main()
{
  int i;
  signal(SIGCHLD, handler);
  for(i=0;i<3;i++)
  switch(fork())
  {
    case 0:
        printf("\tChild created %d\n", getpid());
        exit(0);
  }
  sleep(2);
  return 0;
}
```

Output



```
tushar@tushar-laptop ~ $ ./a.out
Parent created 5928
        Child created 5929
Parent created 5928
        Child created 5930
                Child 5929 exited.
Parent created 5928
                Child 5930 exited.
        Child created 5932
                Child 5932 exited.
```

Thank you

Web Resources
http://tusharkute.com

Blogs
http://digitallocha.blogspot.in
http://kyamputar.blogspot.in

tushar@tusharkute.com