Sending Signals

- **Sending Signals:**
- A process can explicitly send signals to itself or to another process. raise() and kill() function can be used for sending signals. declared in signal.h header file. int raise(int signum)
- raise() function used for sending signal *signum* to the calling process (itself). It returns zero if successful and a nonzero value if it fails.
- int kill(pid_t pid, int signum)
- The kill function used for send a signal *signum* to a process or process group specified by *pid*.

```
#include<stdio.h>
#include<signal.h>
void sig_handler(int signum){
  printf("Inside handler function\n");
}
int main(){
  signal(SIGUSR1,sig_handler); // Register signal handler
  printf("Inside main function\n");
  raise(SIGUSR1);
  printf("Inside main function\n");
  return 0;
}
```

```
#include<stdio.h>   #include <unistd.h> #include<signal.h>
void sig_handler(int signum){
  printf("Inside handler function\n");
}
int main(){
  pid_t pid;
  signal(SIGUSR1,sig_handler); // Register signal handler
  printf("Inside main function\n");
  pid=getpid();      //Process ID of itself
  kill(pid,SIGUSR1);        // Send SIGUSR1 to itself
  printf("Inside main function\n");
  return 0;
}
```

```
#include<stdio.h>  #include <unistd.h>  #include <stdlib.h>
#include<signal.h>
void sig_handler_parent(int signum) {
    printf("Parent : Received a response signal from child \n"); }
void sig_handler_child(int signum){
    printf("Child : Received a signal from parent \n");
    sleep(1);
    kill(getppid(),SIGUSR1);  }
  int main(){    pid_t pid;
    if((pid=fork())<0){
      printf("Fork Failed\n");
      exit(1);   }
```

```
/* Child Process */
else if(pid==0){
  signal(SIGUSR1,sig_handler_child); // Register signal handler
  printf("Child: waiting for signal\n");
  pause();
}
/* Parent Process */
else{
  signal(SIGUSR1,sig_handler_parent); // Register signal handler
  sleep(1);
  printf("Parent: sending signal to Child\n");
  kill(pid,SIGUSR1);
  printf("Parent: waiting for response\n");
  pause();
}
return 0;
}
```

- *fork()* function creates child process and return zero to child process and child process ID to parent process.
- Pid decides parent and child process.
- Parent sleeps for 1 second so that child process can register signal handler function and wait for the signal from parent.
- After 1 second parent process send **SIGUSR1** signal to child process and wait for the response signal from child. Child Process waits for signal from parent and when signal is received, handler function is invoked.
- handler function, the child process sends another **SIGUSR1** signal to parent. Here *getppid()* function is used for getting parent process ID.
- The **pause** function suspends program execution until a signal arrives whose action is either to execute a handler function, or to terminate the process. If the signal causes a handler function to be executed, then **pause** returns

- 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 control c keys.
- SIGKILL: Sure kill a process. Can be generated by – "kill -9 " command.
- SIGPIPE: Illegal write to a pipe.
- SIGQUIT: Process quit. Generated by keys.
- SIGSEGV: Segmentation fault. generated by de-referencing a NULL pointer

- SIGTERM: process termination. Can be generated by – "kill " 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.

Signal Handling Additional Features

int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);

- 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
- 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.

- 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.

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.

```
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);
} }
```

```
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; }
```