Write a program to implement interprocess communication using **pipe** (), **signal** ().

By Harshali Bhuwad

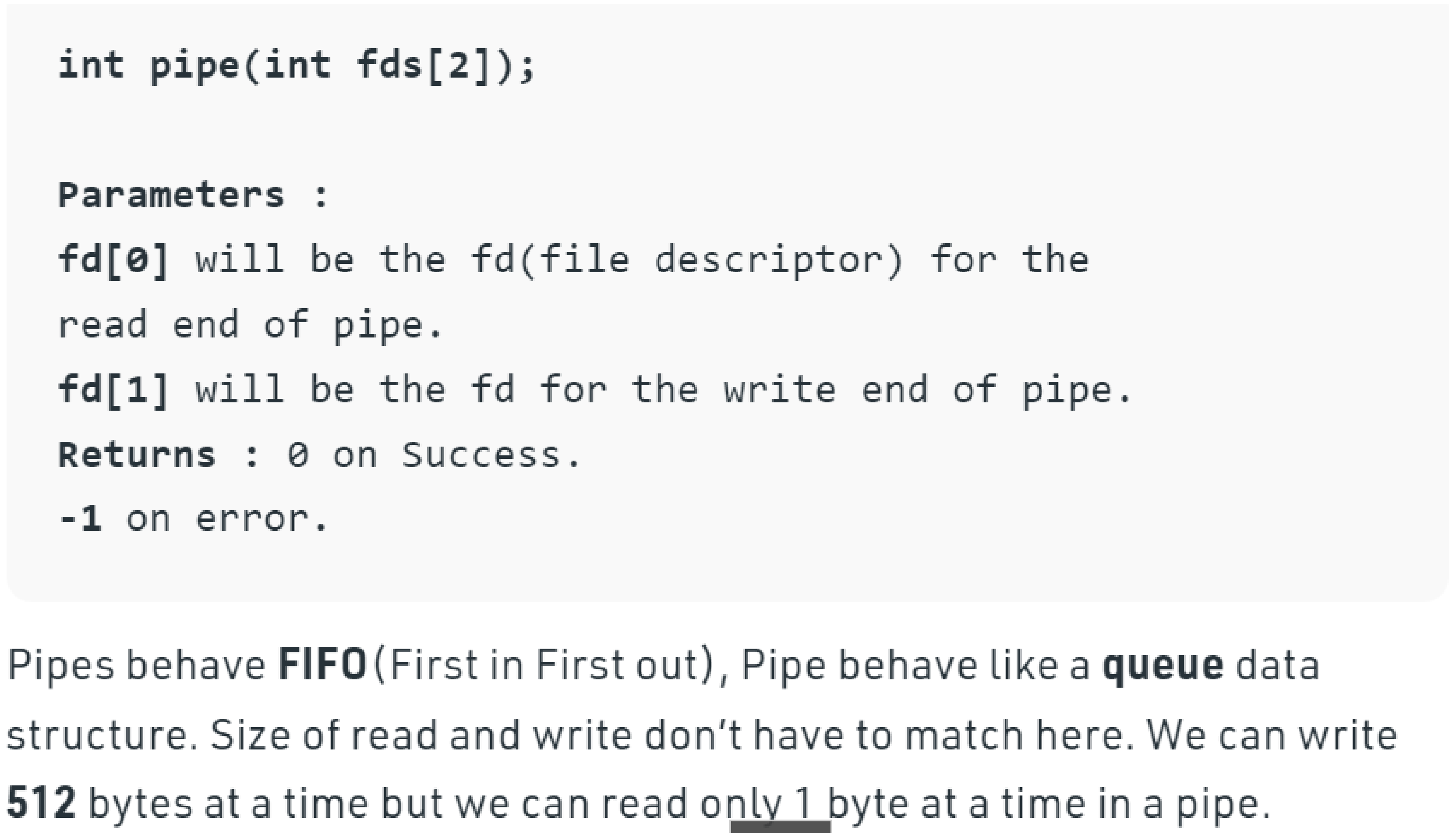
Inter Process Communication - Pipes

- Pipe is a communication medium between two or more related or interrelated processes.
- It can be either within one process or a communication between the child and the parent processes.

```
#include<unistd.h>
int pipe(int pipedes[2]);
```

- This system call would create a pipe for one-way communication i.e., it creates two descriptors.
- This call would return zero on success and -1 in case of failure

Syntax in C language



```
int pipe(int fds[2]);

Parameters :
fd[0] will be the fd(file descriptor) for the
read end of pipe.
fd[1] will be the fd for the write end of pipe.
Returns : 0 on Success.
-1 on error.
```

Pipes behave **FIFO**(First in First out), Pipe behave like a **queue** data structure. Size of read and write don't have to match here. We can write **512** bytes at a time but we can read only 1 byte at a time in a pipe.

Example program 1 − Program to write and read two messages using pipe.

- Algorithm
- **Step 1** − Create a pipe.
- **Step 2** − Send a message to the pipe.
- **Step 3** − Retrieve the message from the pipe and write it to the standard output.
- **Step 4** − Send another message to the pipe.
- **Step 5** − Retrieve the message from the pipe and write it to the standard output.
- **Note** − Retrieving messages can also be done after sending all messages.

```c
#include<stdio.h>
#include<unistd.h>

int main() {
   int pipefds[2];
   int returnstatus;
   char writemessages[2][20]={"Hi", "Hello"};
   char readmessage[20];
   returnstatus = pipe(pipefds);

   if (returnstatus == -1) {
      printf("Unable to create pipe\n");
      return 1;
   }

   printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
   write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
   read(pipefds[0], readmessage, sizeof(readmessage));
   printf("Reading from pipe – Message 1 is %s\n", readmessage);
   printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
   write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
   read(pipefds[0], readmessage, sizeof(readmessage));
   printf("Reading from pipe – Message 2 is %s\n", readmessage);
   return 0;
}
```

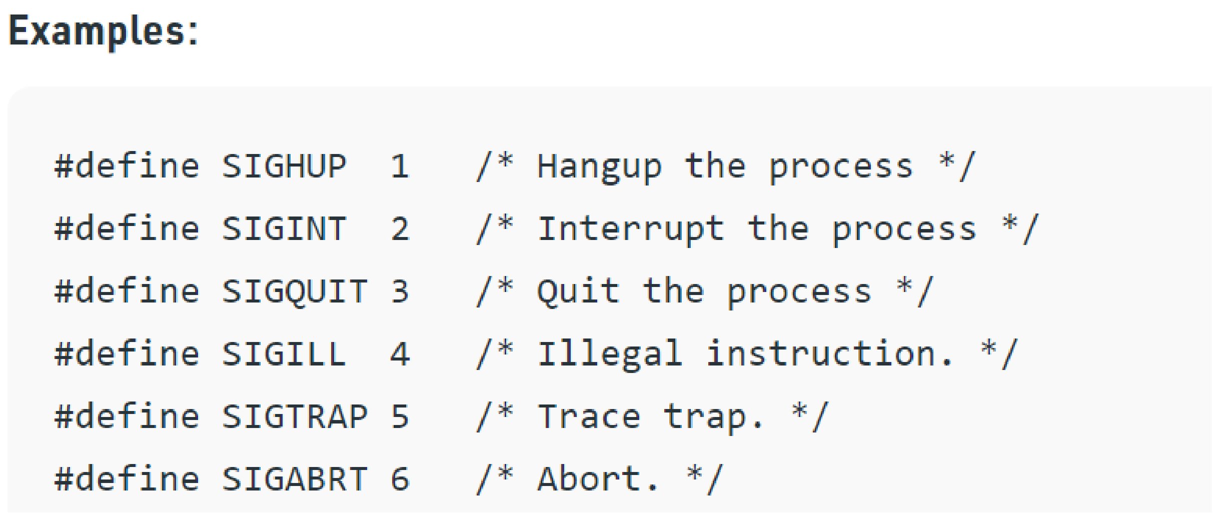
Inter Process Communication-**Signals**

- A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process.

- There are fix set of signals that can be sent to a process. signal are identified by integers.

- Signal number have symbolic names. For example **SIGCHLD** is number of the signal sent to the parent process when child terminates.

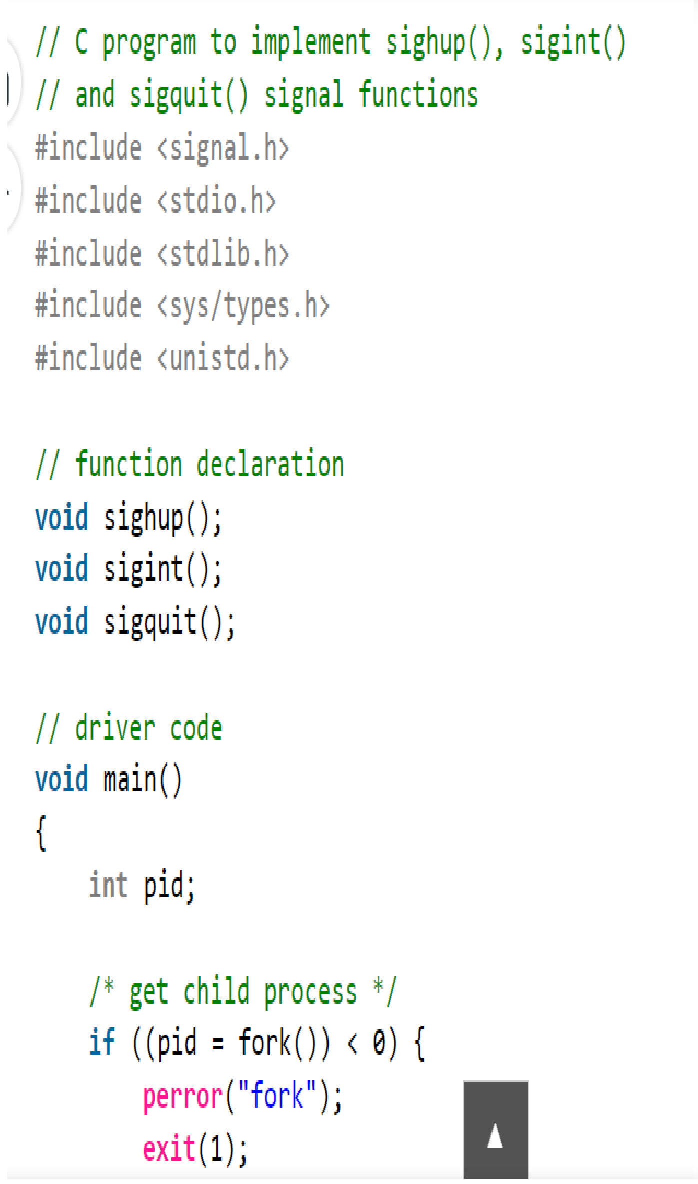
Default Signal Handlers



Examples:

```
#define SIGHUP  1   /* Hangup the process */
#define SIGINT  2   /* Interrupt the process */
#define SIGQUIT 3   /* Quit the process */
#define SIGILL  4   /* Illegal instruction. */
#define SIGTRAP 5   /* Trace trap. */
#define SIGABRT 6   /* Abort. */
```

- Ign: Ignore the signal; i.e., do nothing, just return
- Term: terminate the process
- Cont: unblock a stopped process
- Stop: block the process

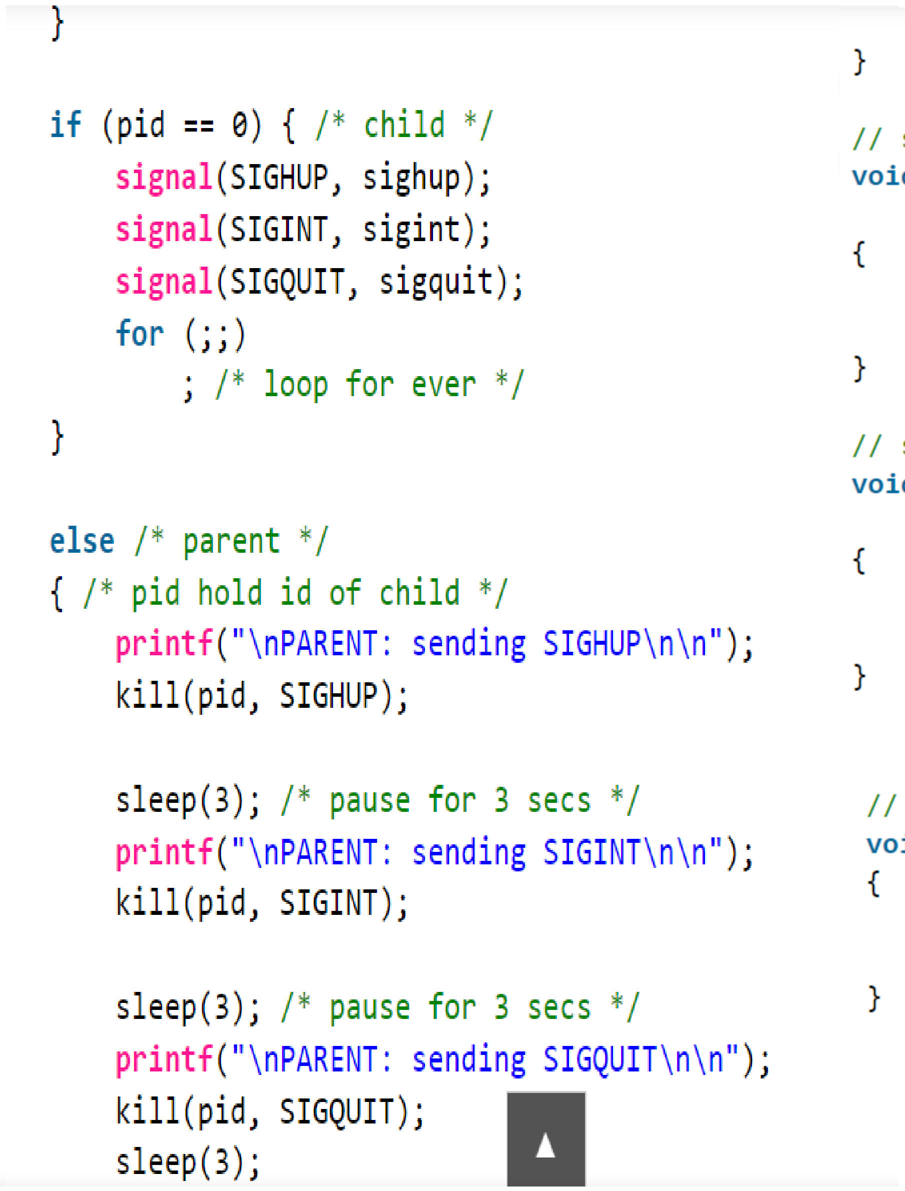


```c
// C program to implement sighup(), sigint()
// and sigquit() signal functions
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

// function declaration
void sighup();
void sigint();
void sigquit();

// driver code
void main()
{
    int pid;

    /* get child process */
    if ((pid = fork()) < 0) {
        perror("fork");
        exit(1);
```



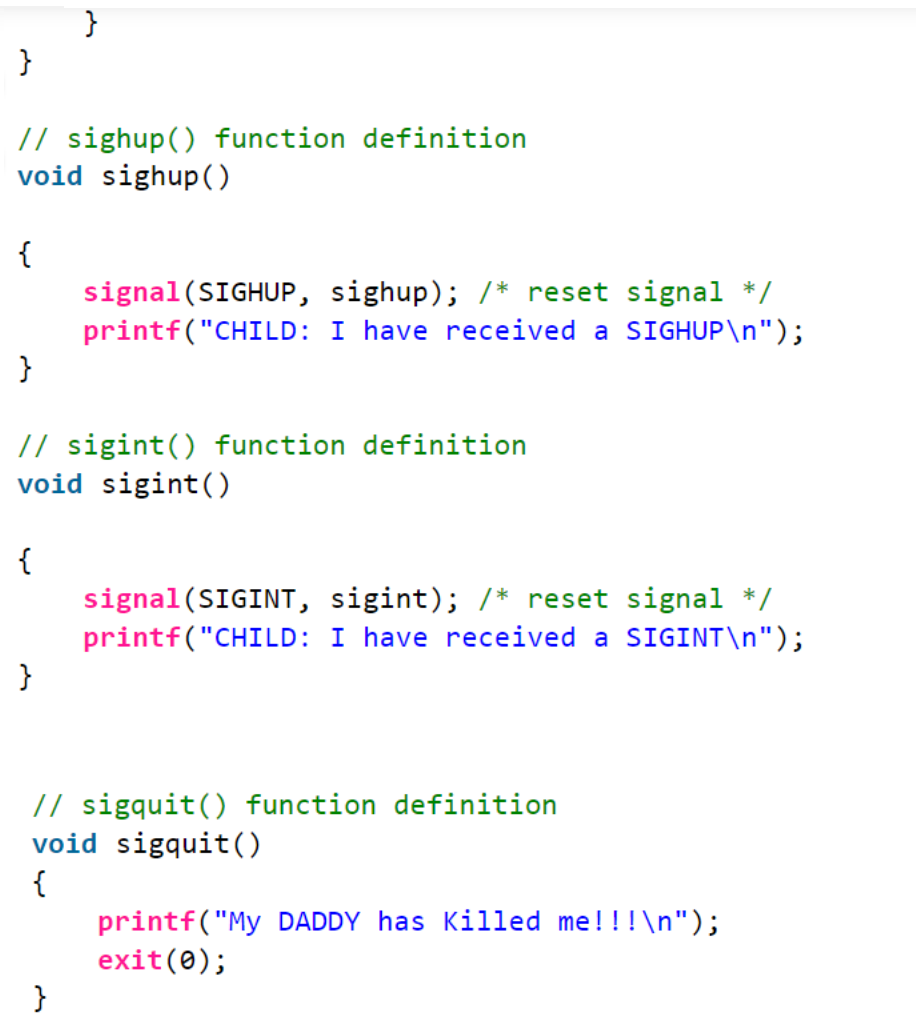
```c
}

if (pid == 0) { /* child */
    signal(SIGHUP, sighup);
    signal(SIGINT, sigint);
    signal(SIGQUIT, sigquit);
    for (;;)
        ; /* loop for ever */
}

else /* parent */
{ /* pid hold id of child */
    printf("\nPARENT: sending SIGHUP\n\n");
    kill(pid, SIGHUP);

    sleep(3); /* pause for 3 secs */
    printf("\nPARENT: sending SIGINT\n\n");
    kill(pid, SIGINT);

    sleep(3); /* pause for 3 secs */
    printf("\nPARENT: sending SIGQUIT\n\n");
    kill(pid, SIGQUIT);
    sleep(3);
```



```c
    }
}

// sighup() function definition
void sighup()

{
    signal(SIGHUP, sighup); /* reset signal */
    printf("CHILD: I have received a SIGHUP\n");
}

// sigint() function definition
void sigint()

{
    signal(SIGINT, sigint); /* reset signal */
    printf("CHILD: I have received a SIGINT\n");
}


 // sigquit() function definition
 void sigquit()
 {
     printf("My DADDY has Killed me!!!\n");
     exit(0);
 }
```