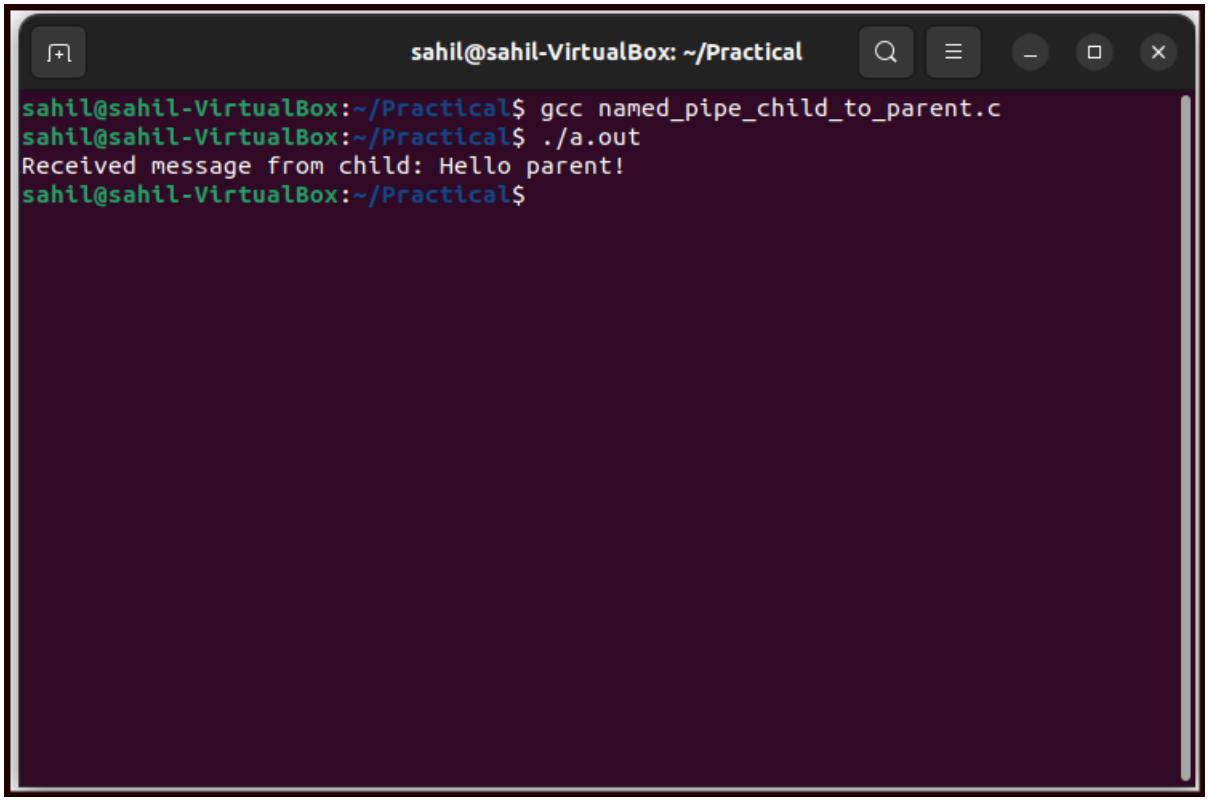
1. Write a program to create a named pipe where child sends message to parent

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
int main() {
   int fd,pid;
   char * myfifo = "namedpipe";
   char buf[256];
   /* create the named pipe */
   mkfifo(myfifo, 0666);
   pid = fork();
   if (pid > 0)
   {
      fd = open(myfifo, O_RDONLY);
      read(fd, buf, sizeof(buf));
      printf("Received message from child: %s\n", buf);
      close(fd);
      unlink(myfifo);
      _exit(0);
   }
   else
   {
      fd = open(myfifo, O_WRONLY);
      write(fd, "Hello parent!", sizeof("Hello parent!"));
      close(fd);
      _exit(0);
   }
}
```

Output :



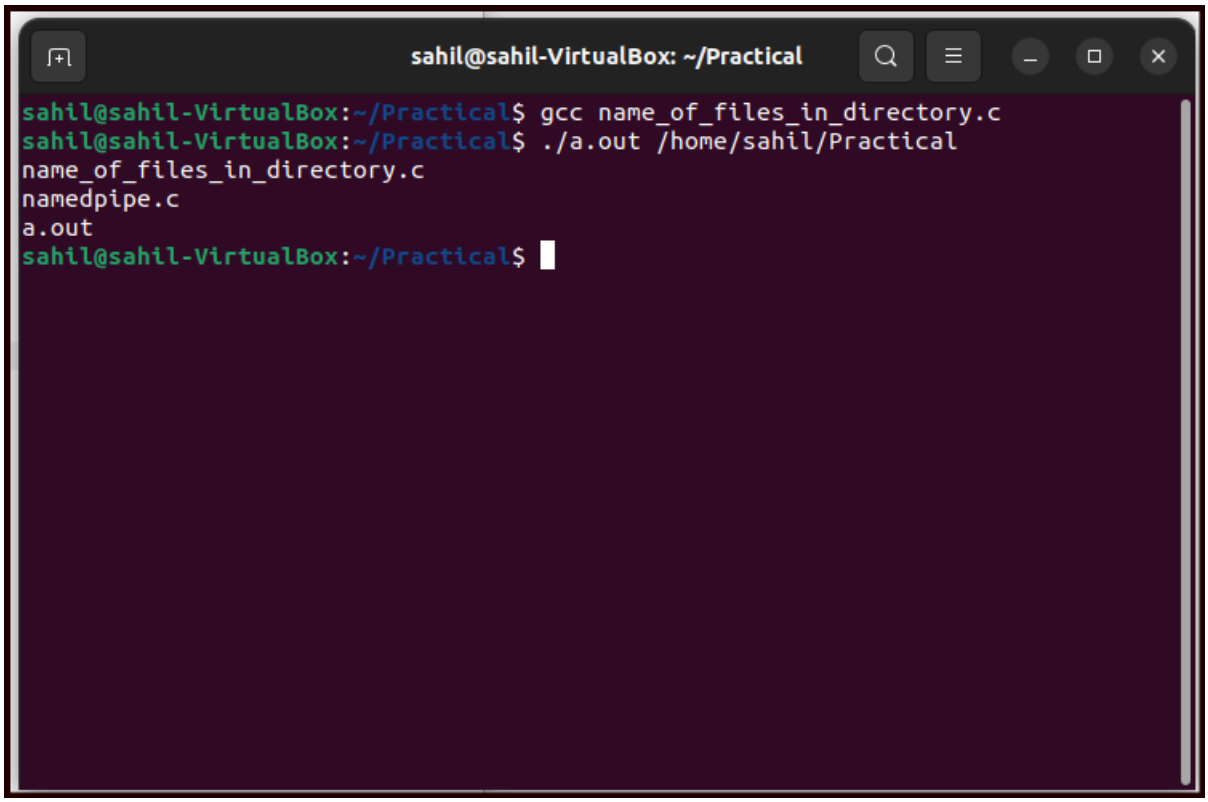
2. Write a program in C to print the file names of a specified directory.

Code :

```
#include<stdio.h>
#include<dirent.h>
#include<unistd.h>
void main(int argc ,char *argv[])

{
        DIR *dp;
        struct dirent *entry;
        dp = opendir(argv[1]);
        if (dp  == NULL) {
                printf("Directory does not exists.\n");
                _exit(0);
        }
        while (entry = readdir(dp))
                if (entry->d_type == DT_REG)
                        printf("%s\n",entry->d_name);
        closedir(dp);
```

```
}
```

Output :



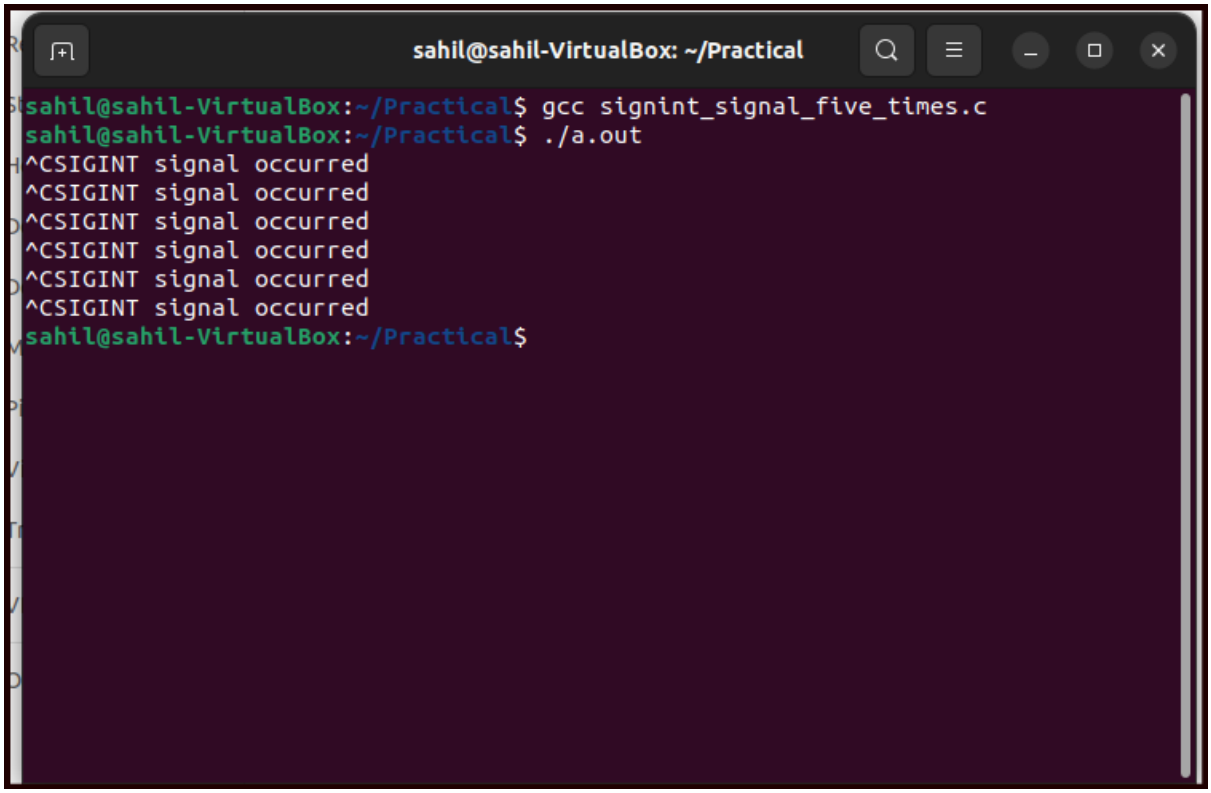
3. Write a program to catch SIGINT signal five times and print message 'SIGINT signal occurred ' every timed and exit at sixth occurrence. Also ignore every occurrence of SIGTSTP signal

Code :

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int count = 0;
void sigint_handler(int sig) {
   count++;
   printf("SIGINT signal occurred\n");
   if (count == 6)
      exit(0);
}
int main() {
   signal(SIGINT, sigint_handler);
```

```
    while (1) {
        // Wait for signals
    }
    return 0;
}
```

Output :



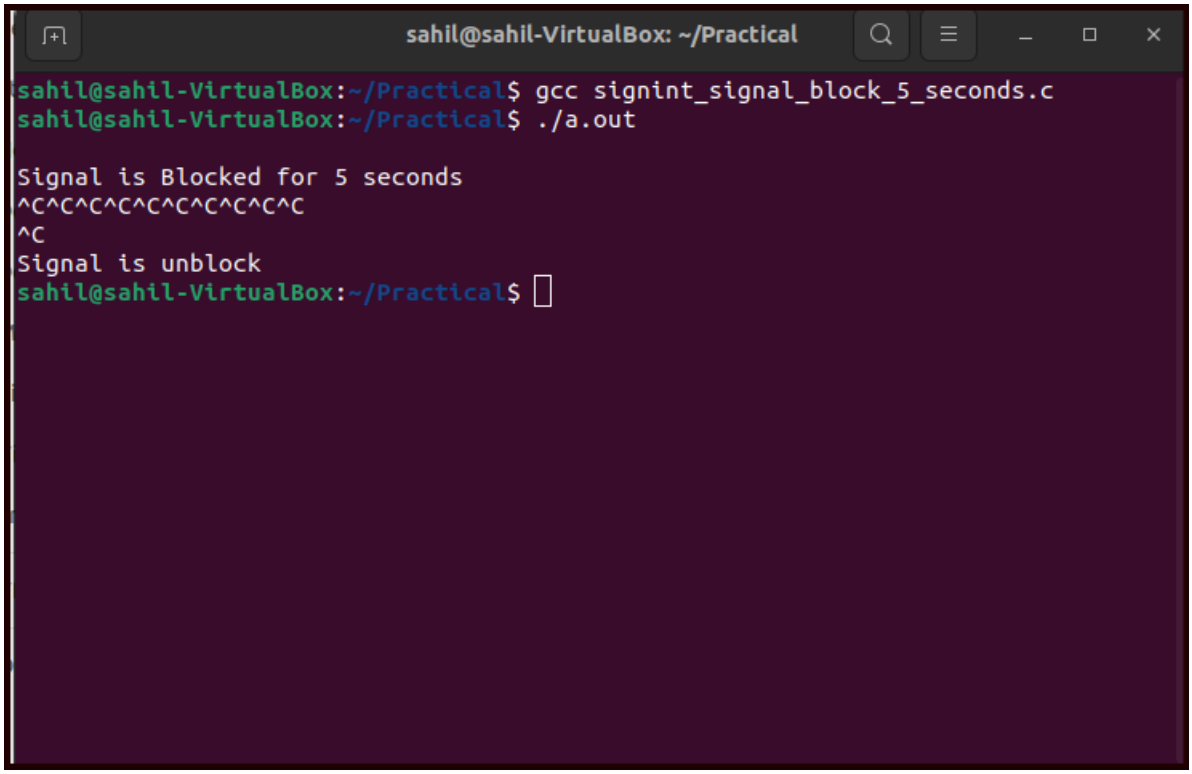
4. Write a program to block SIGINT signal for 5 seconds

Code :

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

void sigint_handler(int sig) {
    sleep(5);
    printf("\nSignal is unblock\n");
    _exit(0);
}
int main() {
```

```
    signal(SIGINT, sigint_handler);
    printf("\nSignal is Blocked for 5 seconds\n");
    while (1) {
        // Wait for signals
    }
    return 0;
}
```

Output :

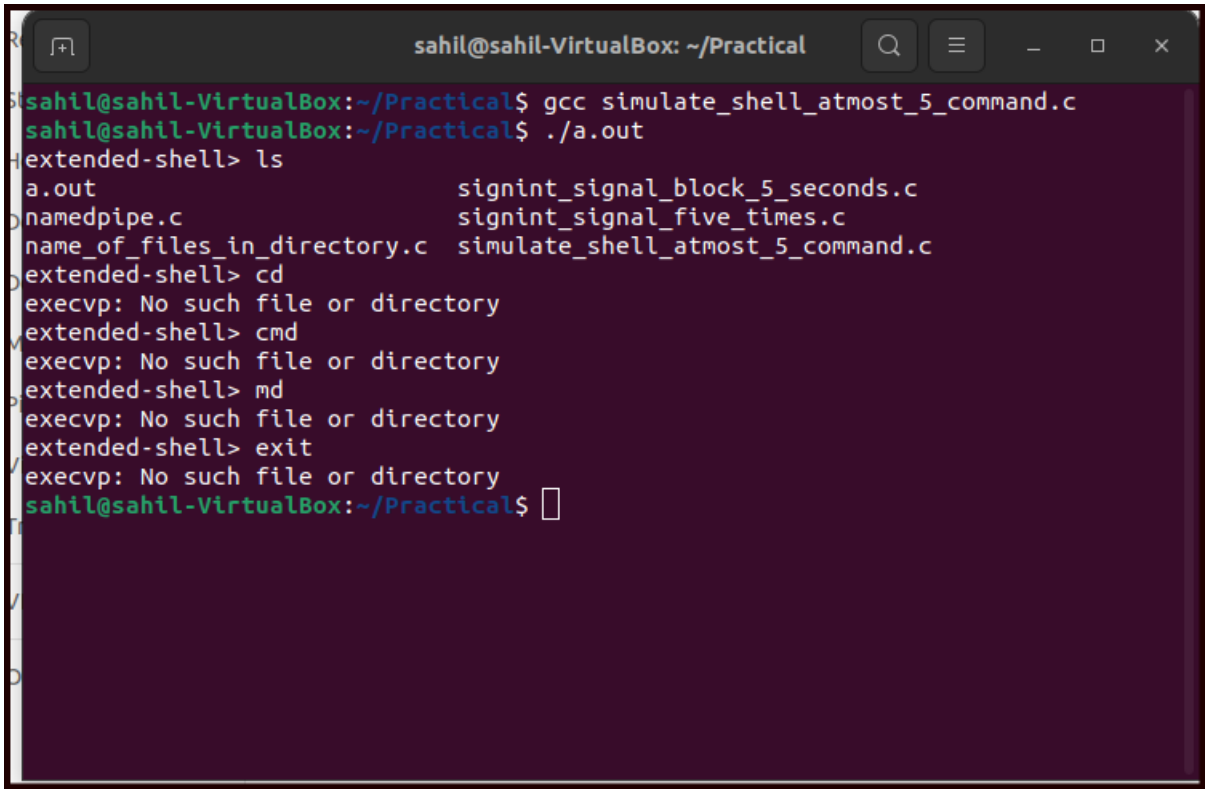


5. **Write a program in LINUX to simulate extended shell. Show the prompt and accept standard shell command which will be executed by child process using one of the exec family system calls. Parent process waits until child finished execution the command may consist of at the most 5 parameters. The process should be repeated till user types "exit".**

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
   char line[80];
   char* args[5];
   int status;
   while (1) {
      printf("extended-shell> ");
      if (fgets(line,80, stdin) == NULL)
         break;
      char* token = strtok(line, " \t\n");
      int i = 0;
      while (token != NULL && i < 5) {
         args[i] = token;
         token = strtok(NULL, " \t\n");
         i++;
      }
     args[i] = NULL;
      int pid = fork();
      if (pid == -1) {
         perror("fork");
         exit(1);
      } else if (pid == 0) {
         if (execvp(args[0], args) == -1) {
            perror("execvp");
            exit(1);
         }
      } else
         wait(&status);

      if (strcmp(args[0], "exit") == 0)
         break;
   }
   return 0;
}
```

Output :

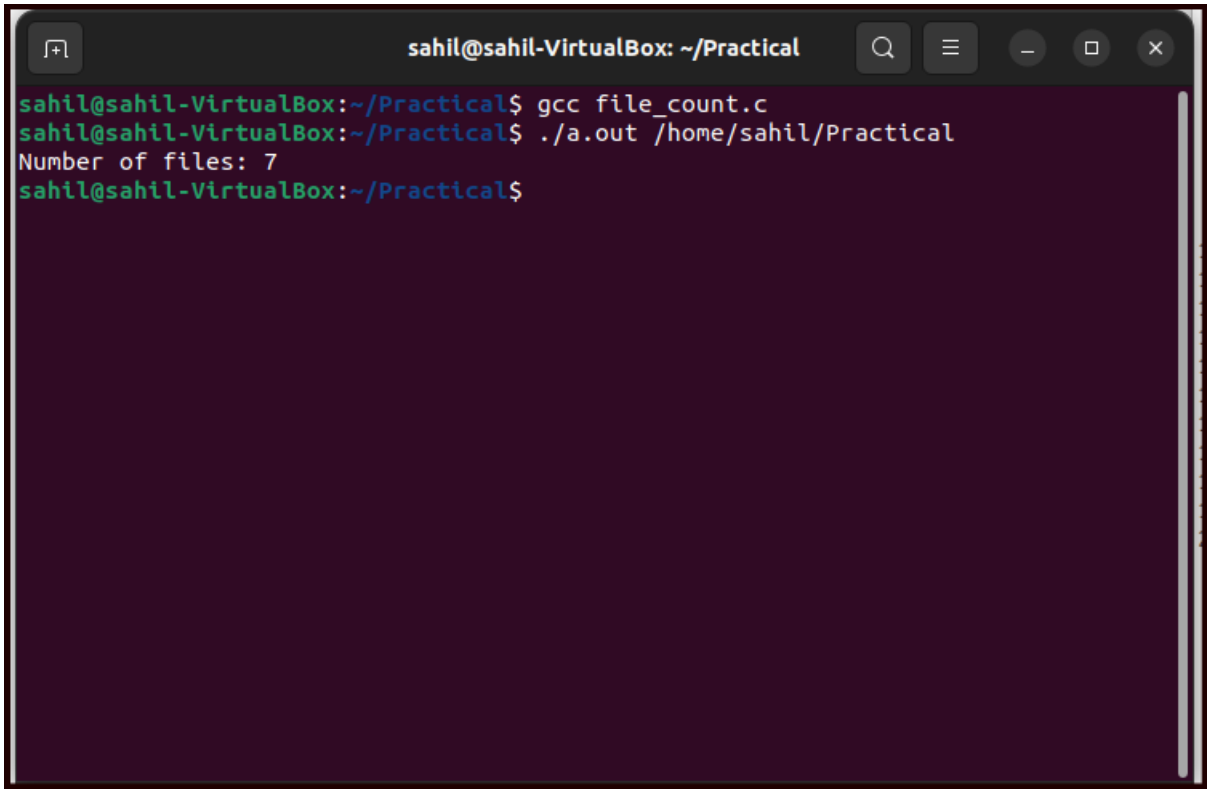


6. Write a program to count the no of files of a specified directory.

Code :

```
#include<stdio.h>
#include<dirent.h>
#include<unistd.h>
void main(int argc , char *argv[])
{
        int cnt = 0;
        DIR *dp;
        struct dirent *entry;
        dp = opendir(argv[1]);
        if (dp  == NULL) {
                printf("Directory does not exists.\n");
                _exit(0);
        }
        while (entry = readdir(dp))
                if (entry->d_type == DT_REG)
                        cnt++;
        printf("Number of files: %d\n", cnt);
        closedir(dp);

}
```

Output :



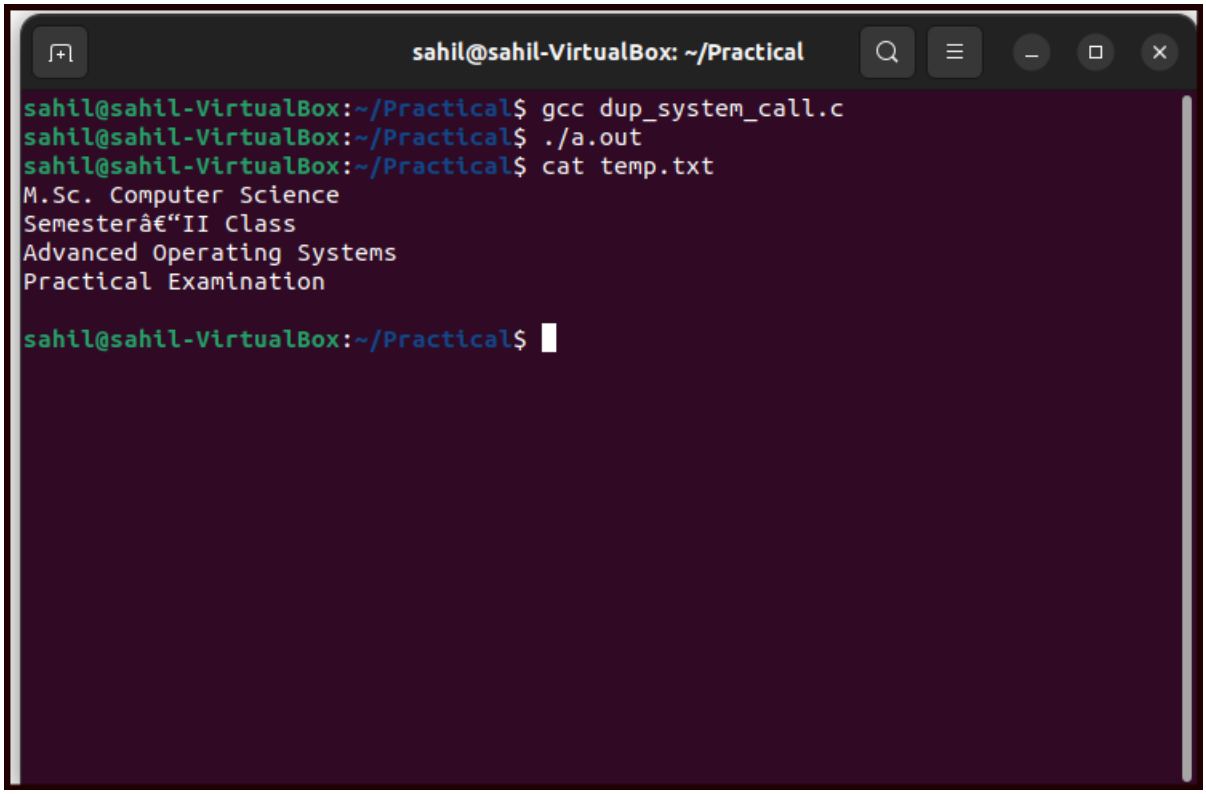
7. Use of dup and dup2 system call. Create one file named temp.txt. Write some contents in the file. Make use of dup and dup2 system calls to write following content in the file. Makes use of four descriptors and channel zero.

Code :

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>

void main() {
        char text1[] = "M.Sc. Computer Science\n";
        char text2[] = "Semesterâ€“II Class\n";
        char text3[] = "Advanced Operating Systems\n";
        char text4[] = "Practical Examination\n";
```

```
        int fd1 = open("temp.txt", O_WRONLY);
        int fd2 = dup(fd1);
        int fd3 = dup2(fd2, 0);
        int fd4 = dup2(fd3, 1);

        write(fd1, text1, strlen(text1));
        write(fd2, text2, strlen(text2));
        write(fd3, text3, strlen(text3));
        printf("%s", text4);

}
```

Output :



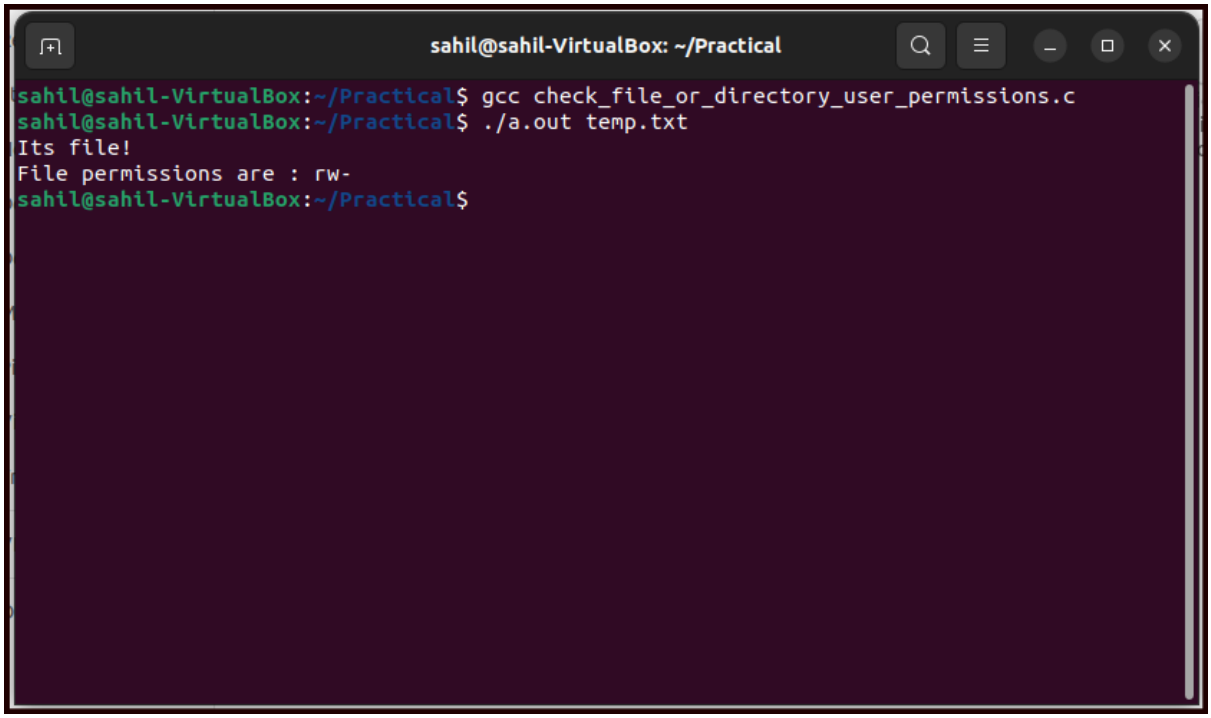
8. Check whether the specified file is regular file or directory and print user permissions of the file.

Code :

```
#include <stdio.h>
#include <dirent.h>
```

```
#include <unistd.h>
#include <sys/stat.h>
void main(int argc , char *argv[])
{
	char *fileName = argv[1];
	struct stat pathStat;
	stat(fileName, &pathStat);

	if (S_ISREG(pathStat.st_mode))
		printf("Its file!\n");
	else if (S_ISDIR(pathStat.st_mode))
		printf("its Directory!\n");
	printf("File permissions are : ");
	printf((pathStat.st_mode & S_IRUSR) ? "r" : "-");
	printf((pathStat.st_mode & S_IWUSR) ? "w" : "-");
	printf((pathStat.st_mode & S_IXUSR) ? "x\n" : "-\n");
}
```

Output :



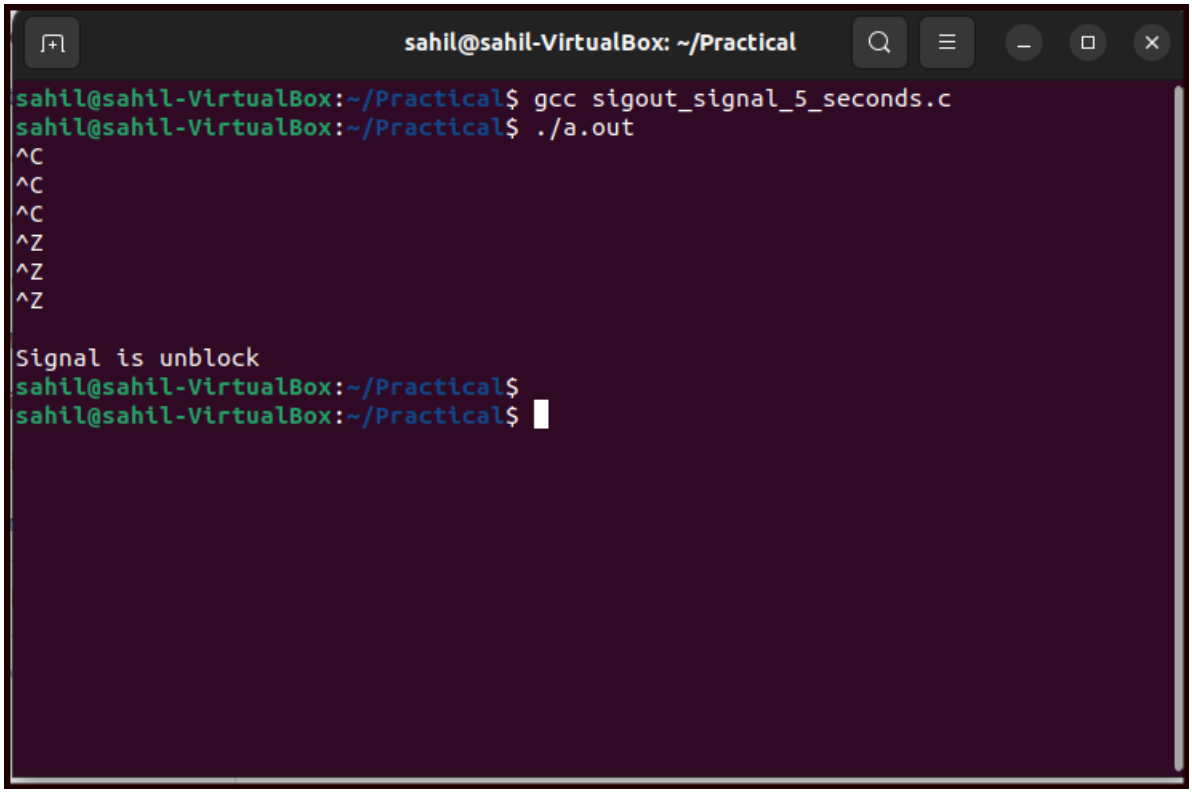
9. Write a program in Linux to block SIGOUIT signal for 5 seconds

Code :

```
#include <stdio.h>
#include <signal.h>
```

```
#include <unistd.h>

void sigint_handler(int sig) {
}

void handle_sigtstp(int sig) {
   sleep(5);
   printf("\nSignal is unblock\n");
   _exit(0);
}

int main() {
  signal(SIGINT, sigint_handler);
  signal(SIGTSTP, handle_sigtstp);

  while (1) {
    // Loop indefinitely
  }
  return 0;
}
```

Output :

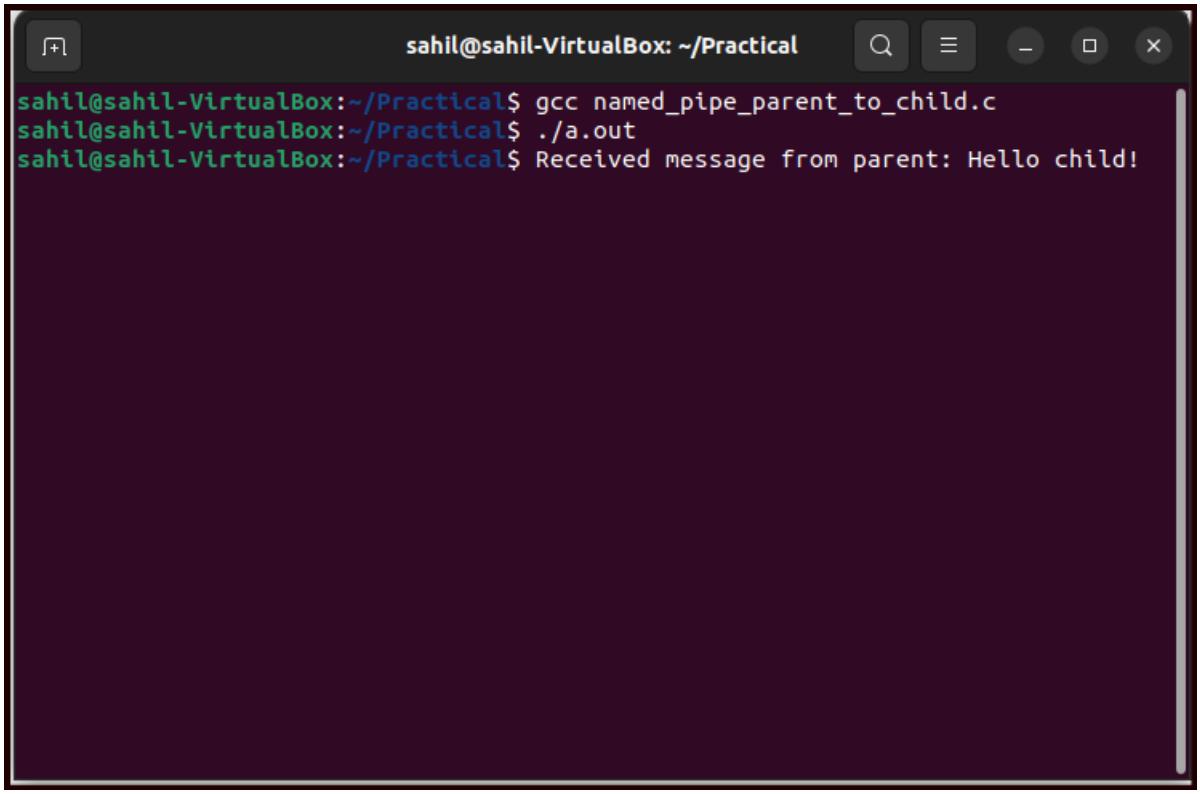


10. Write a program to create a named pipe where parent sends message to child.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
int main() {
   int fd,pid;
   char * myfifo = "namedpipe";
   char buf[256];
   /* create the named pipe */
   mkfifo(myfifo, 0666);
   pid = fork();
   if (pid > 0)
   {
      fd = open(myfifo, O_WRONLY);
      write(fd, "Hello child!", sizeof("Hello child!"));
      close(fd);
      _exit(0);
   }
   else
   {
      fd = open(myfifo, O_RDONLY);
      read(fd, buf, sizeof(buf));
      printf("Received message from parent: %s\n", buf);
      close(fd);
      unlink(myfifo);
      _exit(0);
   }
}
```

Output :



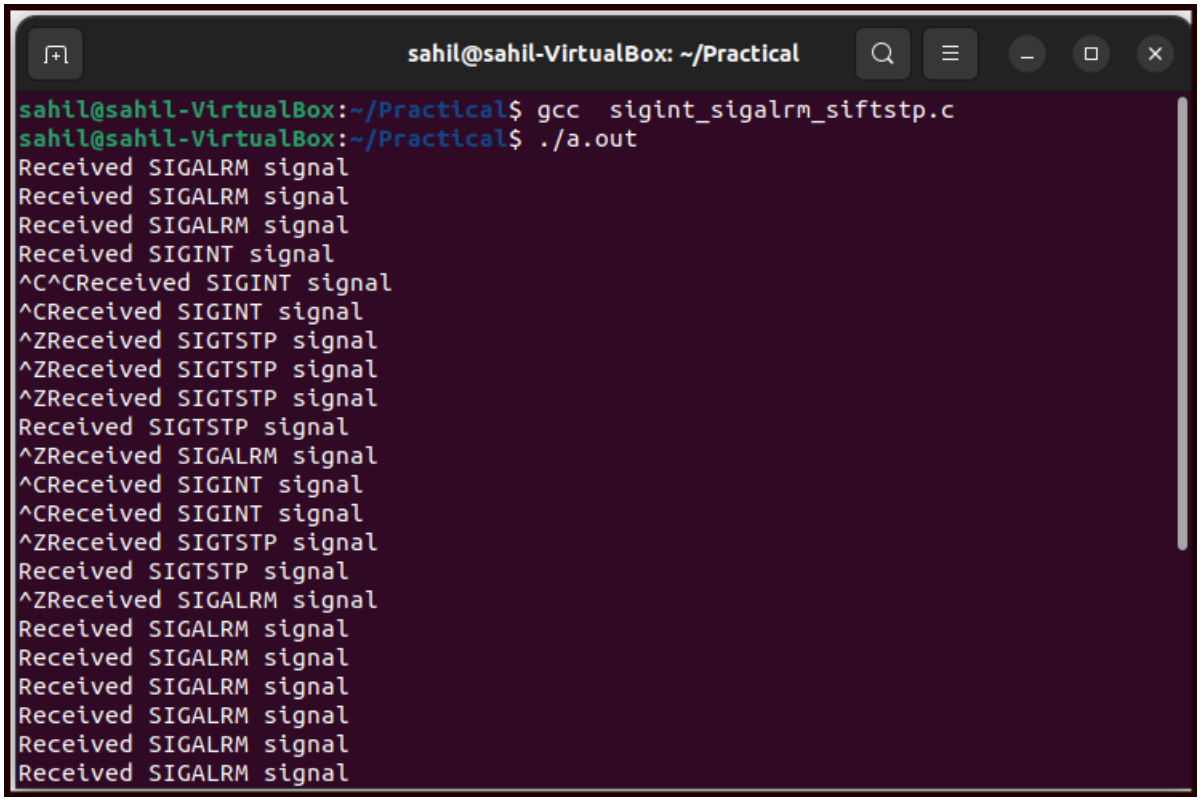
11. Write program to handle SIGINT, SIGALRM and SIFTSTP signals.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void handle_sigint(int sig) {
   printf("Received SIGINT signal\n");
}

void handle_sigalrm(int sig) {
   printf("Received SIGALRM signal\n");
}

void handle_sigtstp(int sig) {
   printf("Received SIGTSTP signal\n");
}
int main() {
   signal(SIGINT, handle_sigint);
   signal(SIGALRM, handle_sigalrm);
   signal(SIGTSTP, handle_sigtstp);
```

```
    while (1) {
        alarm(1);
        sleep(1);
    }
    return 0;
}
```

Output :



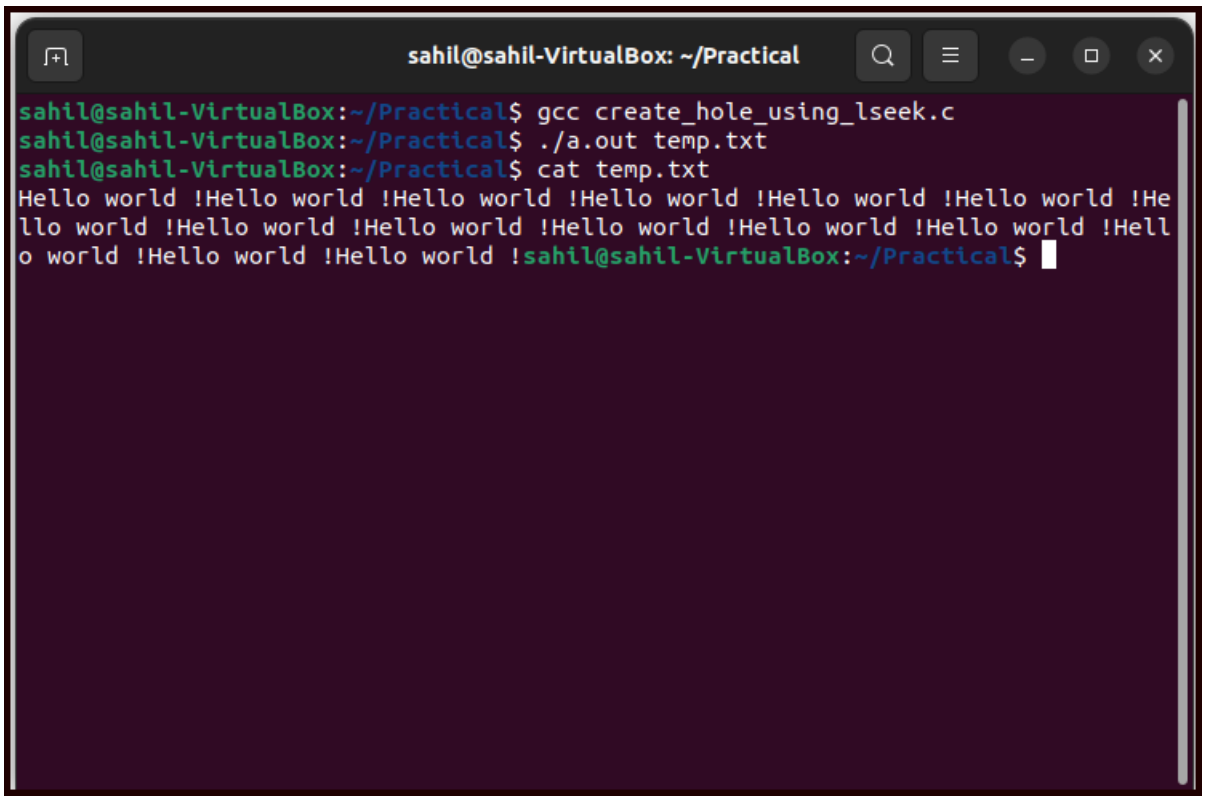
12. Write program to create hole in it.(Use Lseek system call). Write “Hello world !” five times after every 100 blank character.

Code :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include<string.h>
int main(argc,argv)
int argc;
char *argv[];
 {
```

```
  int fd,skval;
  char c;
  if(argc !=2)
   _exit(0);
  fd = open(argv[1],O_WRONLY|O_CREAT);
  if(fd == -1)
   _exit(0);
  for(int j=0; j<3; j++)
   {
     skval = lseek(fd,100L,1);
     for(int i=0; i<5;i++)
       write(fd,"Hello world !",strlen("Hello world !"));
   }
  close(fd);
  return 0;
}
```

Output :

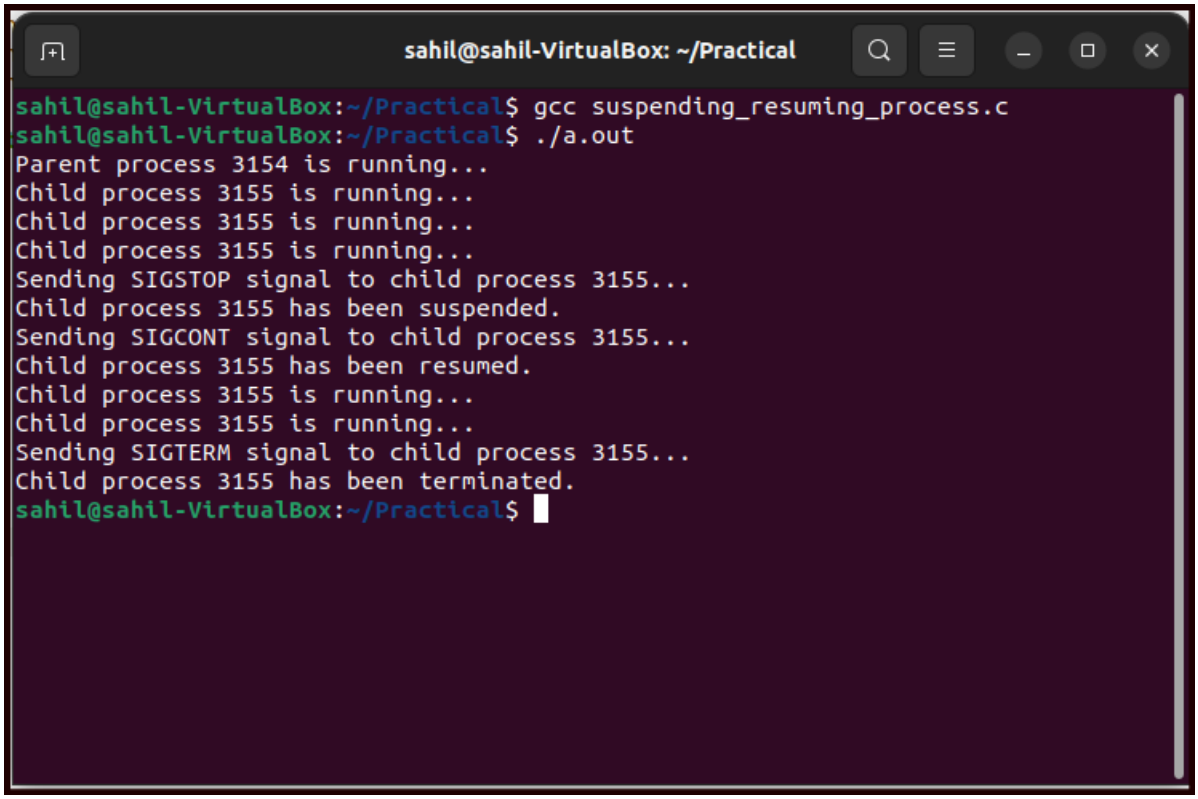


13. Write program that illustrate suspending and resuming processes using signals

Code :

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

int main() {
   int pid = fork();
   if (pid == 0)
   {
      printf("Child process %d is running...\n", getpid());
      while (1)
       {
         printf("Child process %d is running...\n", getpid());
         sleep(1);
      }
   }
   else
   {
      printf("Parent process %d is running...\n", getpid());

      sleep(2);
      printf("Sending SIGSTOP signal to child process %d...\n", pid);
      kill(pid, SIGSTOP);
      printf("Child process %d has been suspended.\n", pid);

      sleep(2);
      printf("Sending SIGCONT signal to child process %d...\n", pid);
      kill(pid, SIGCONT);
      printf("Child process %d has been resumed.\n", pid);

      sleep(2);
      printf("Sending SIGTERM signal to child process %d...\n", pid);
      kill(pid, SIGTERM);
      printf("Child process %d has been terminated.\n", pid);
   }
   return 0;
}
```

Output :



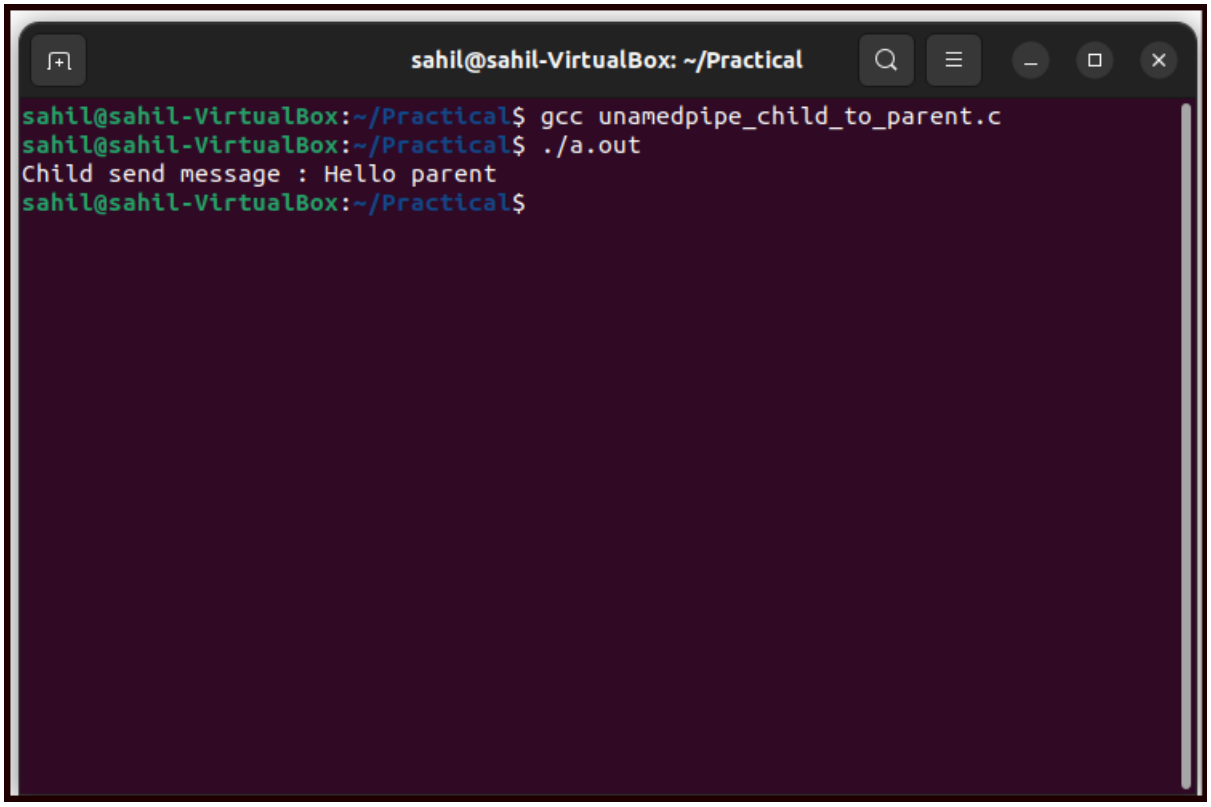
14. Write a program to create an unmanned pipe where child sends messages to parent.

Code :

```
#include<stdio.h>
#include<string.h>
#include <sys/types.h>
#include <unistd.h>
#include<fcntl.h>
int main()
{
  int fd[2], pid;
  char buff[50];
  char msg[]="Hello parent";
  pipe(fd);
  if((pid=fork()) != 0)
   {
    close(fd[1]);
    read(fd[0],buff,sizeof(msg));
    printf("Child send message : %s\n",buff);
    close(fd[0]);
   }
  else
   {
```

```
        close(fd[0]);
        write(fd[1],msg,strlen(msg));
        close(fd[1]);
      }
      return 0;
}
```

Output :



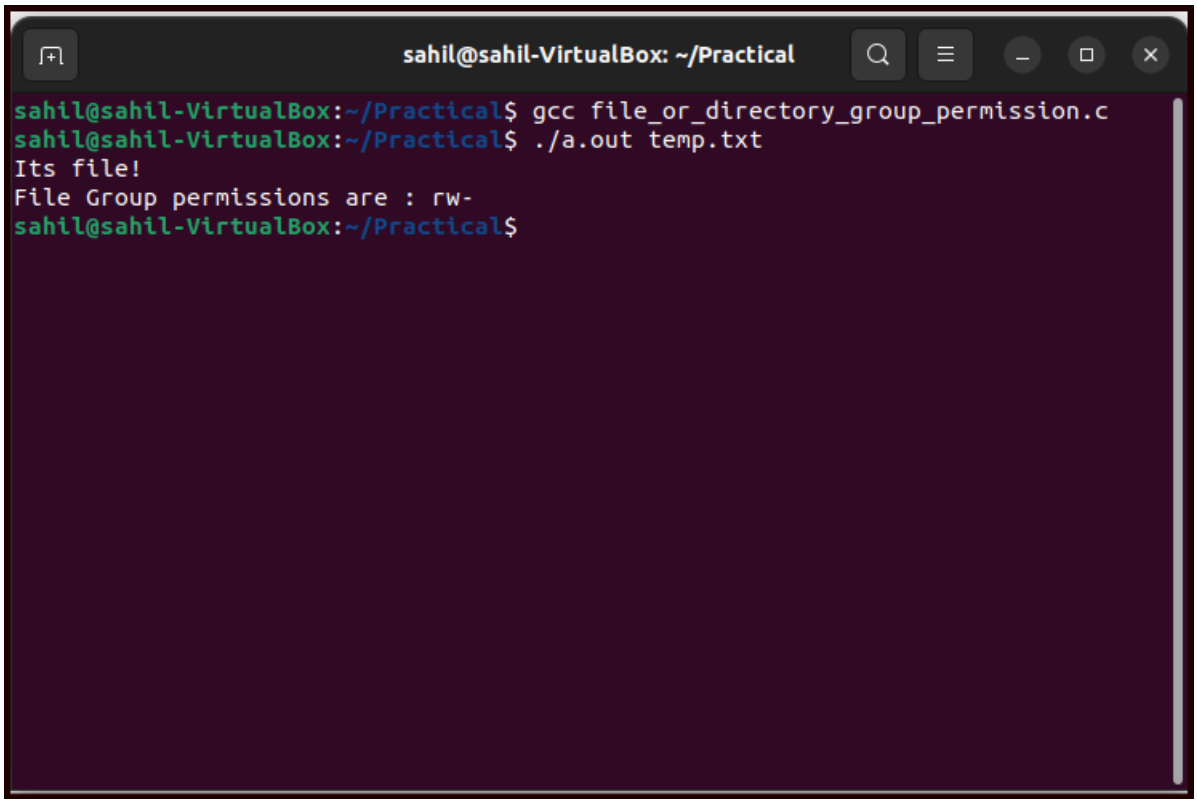
15. Print the type of file and group permission of it where file name is accepted thru command line arguments

Code :

```
#include <stdio.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>

void main(int argc , char *argv[])
{
      char *fileName = argv[1];
          struct stat pathStat;
```

```
        stat(fileName, &pathStat);

        if (S_ISREG(pathStat.st_mode))
                printf("Its file!\n");
        else if (S_ISDIR(pathStat.st_mode))
                printf("its Directory!\n");
        printf("File Group permissions are : ");
        printf((pathStat.st_mode & S_IRGRP) ? "r" : "-");
        printf((pathStat.st_mode & S_IWGRP) ? "w" : "-");
        printf((pathStat.st_mode & S_IXGRP) ? "x\n" : "-\n");
}
```

Output :



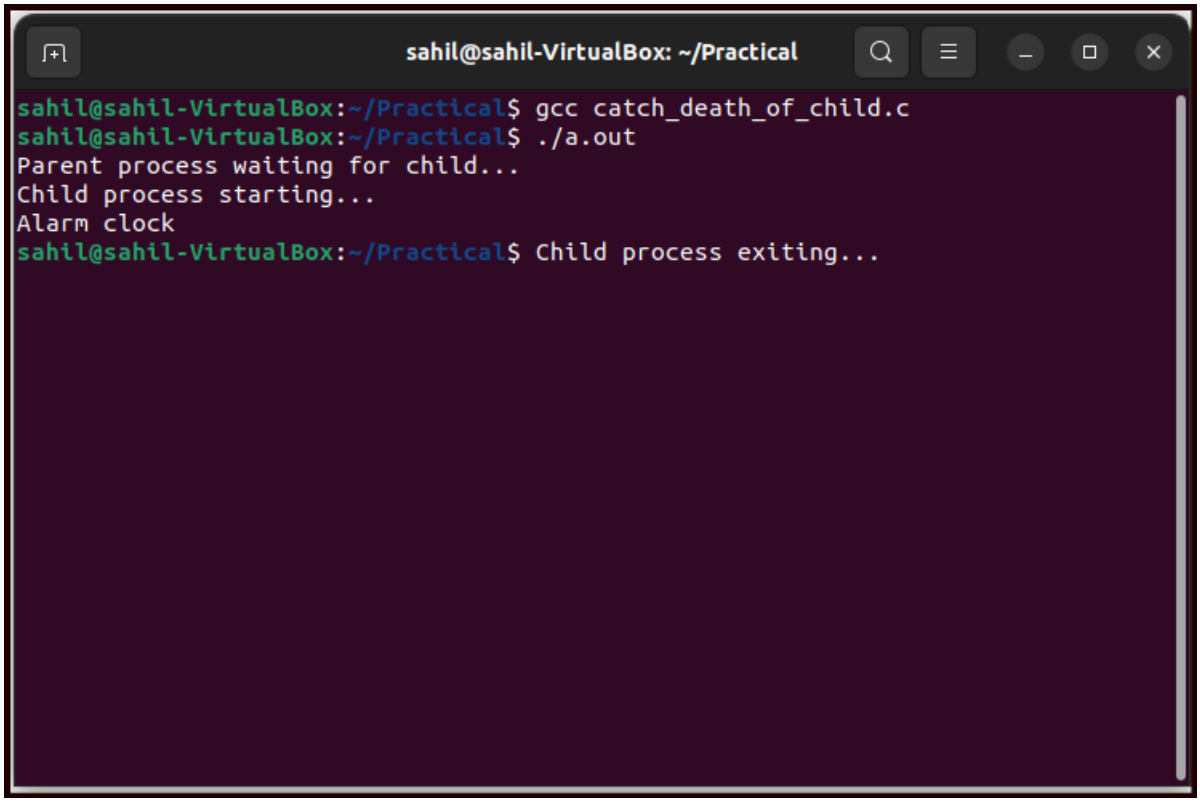
16. Write a program to catch death of child signal by parent process after 5 seconds. Use alarm system call

Code :

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
```

```
#include <sys/types.h>
#include <sys/wait.h>

void sigchld_handler(int sig) {
    printf("Child process terminated.\n");
}

int main() {
    pid_t pid;
    int status;

    signal(SIGCHLD, sigchld_handler);
    pid = fork();
    if (pid == 0) {
        printf("Child process starting...\n");
        sleep(10);
        printf("Child process exiting...\n");
        exit(0);
    }
    else if (pid > 0)
    {
        printf("Parent process waiting for child...\n");
        alarm(5);
        wait(&status);
    }
    else
    {
        printf("Fork failed.\n");
        exit(0);
    }
    printf("Parent process exiting.\n");
    return 0;
}
```

Output :



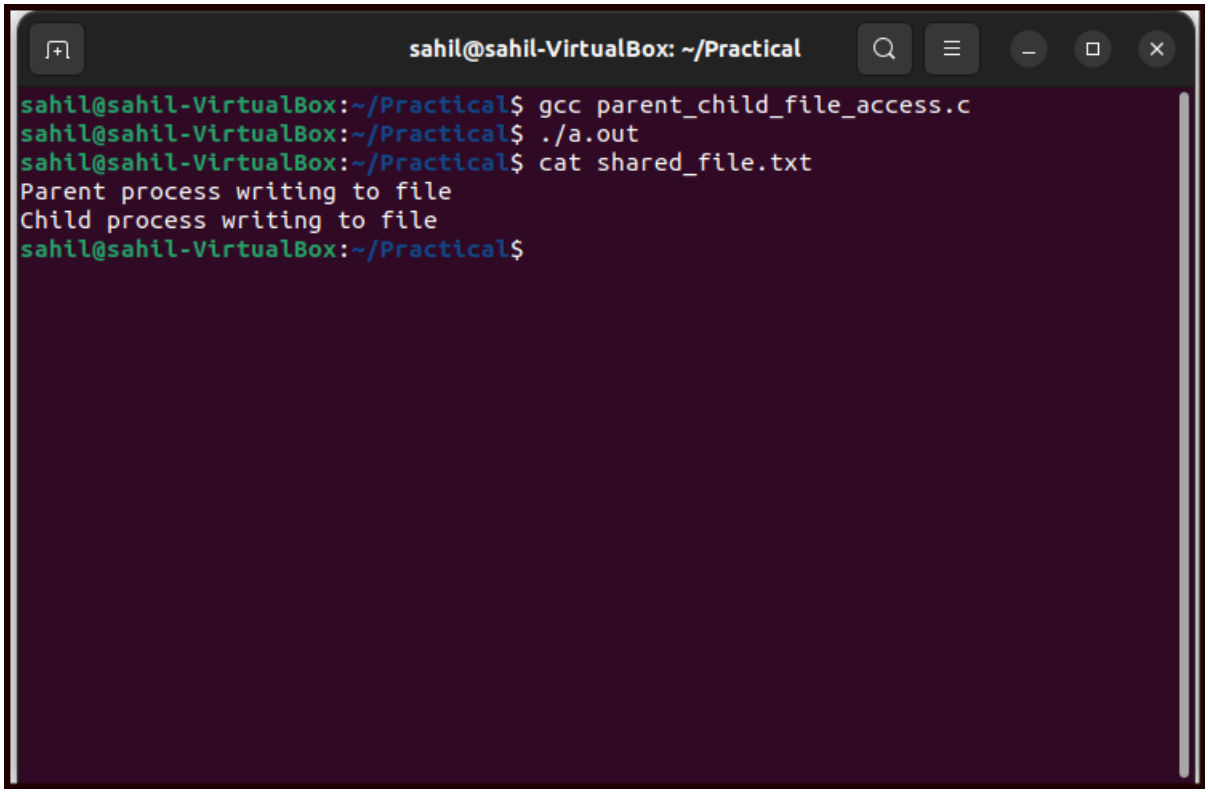
17. Write a program where parent and child share file access

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>

int main() {
   int fd;
   pid_t pid;
   char buf[256];

   fd = open("shared_file.txt", O_RDWR | O_CREAT,0644);
   pid = fork();
   if (pid < 0) {
      perror("fork");
      exit(0);
```

```
    }
    else if (pid == 0)
    {
        sprintf(buf, "Child process writing to file\n");
        write(fd, buf, strlen(buf));
        exit(0);
    }
    else
    {
        sprintf(buf, "Parent process writing to file\n");
        write(fd, buf, strlen(buf));
        exit(0);
    }
    return 0;
}
```

Output :



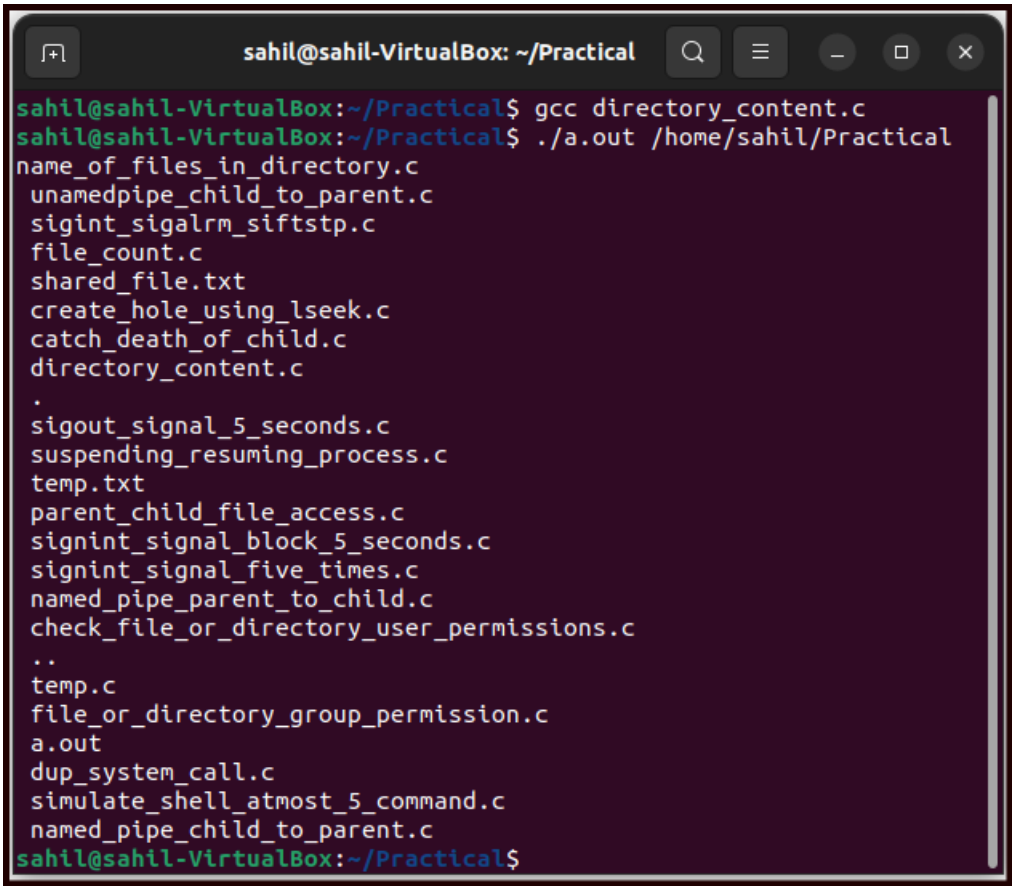
18. Write the program that accepts directory name as input and print its content

Code :

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

```
#include<dirent.h>
#include<unistd.h>

void main(int argc , char *argv[])
{
        DIR *dp;
        struct dirent *entry;
        dp = opendir(argv[1]);
        if (dp  == NULL) {
                printf("\n Directory does not exists .\n");
                _exit(0);
        }
        while( entry = readdir(dp))
                printf("%s\n ",entry-> d_name);
        closedir(dp);
}
```

Output :

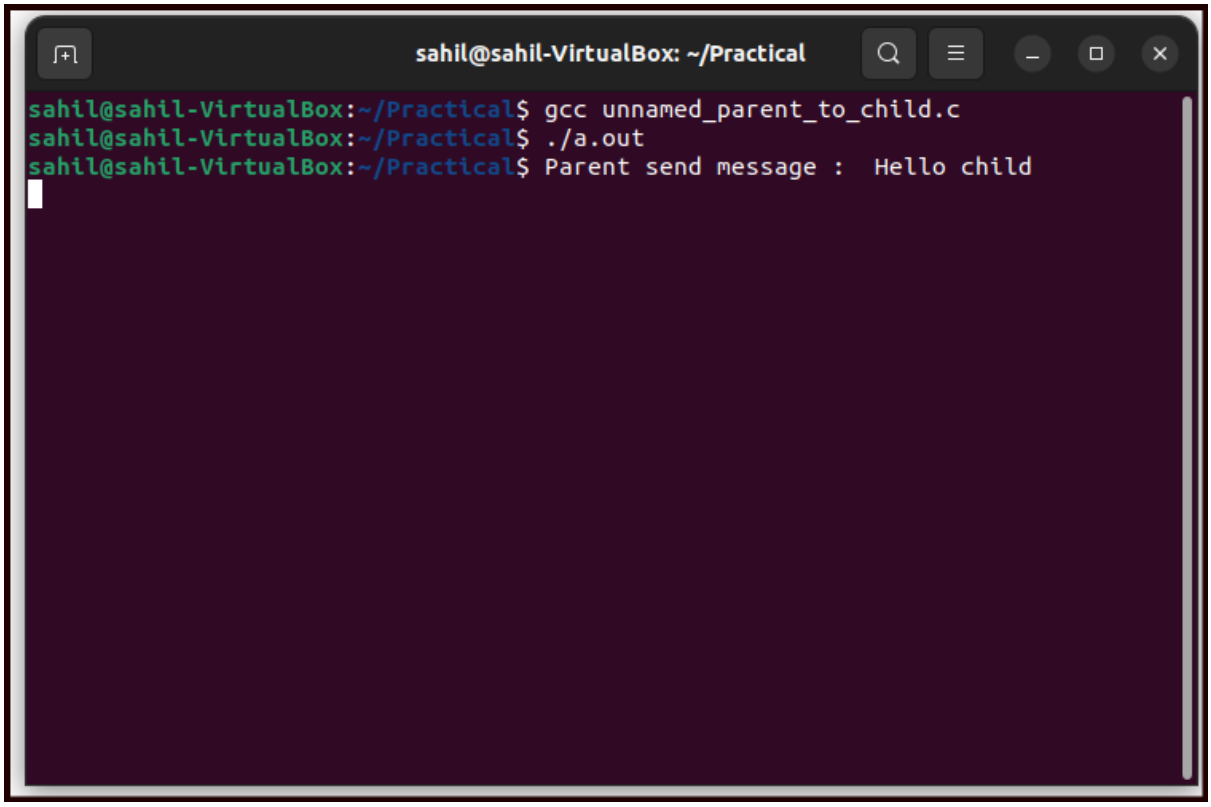


19. Write a program to create an unnamed pipe where parent sends message to child

Code :

```
#include<stdio.h>
#include<string.h>
#include <sys/types.h>
#include <unistd.h>
#include<fcntl.h>
int main()
{
  int fd[2], pid;
  char buff[50];
  char msg[]="Hello child";
  pipe(fd);
  if((pid=fork()) != 0)
   {
     close(fd[0]);
     write(fd[1],msg,strlen(msg));
     close(fd[1]);
   }
  else
   {
     close(fd[1]);
     read(fd[0],buff,sizeof(msg));
     printf("Parent send message :  %s\n",buff);
     close(fd[0]);
   }
   return 0;
}
```

Output :



20. Write a program to implement following commands as linux.

a) Typelines +5 <filename> print first 5 lines of a file

b) Typelines -8 <filename> print last 20 lines of a file

Code :

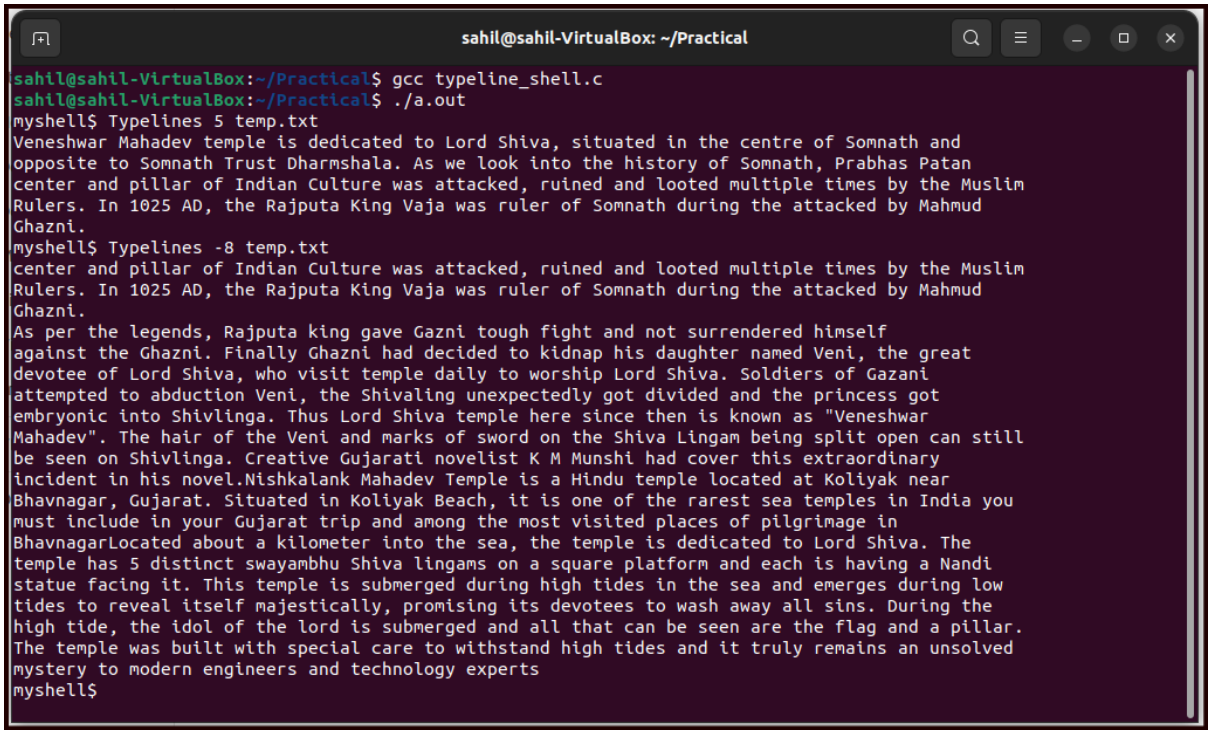
```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<fcntl.h>

#include<string.h>

void typeline(char *op, char *fn)

 {
```

```
int fh,i,j,n;
char c;
fh = open(fn,O_RDONLY);
if(fh == -1)
  {
    printf("File %s not found.\n", fn);
    return;
  }
 if(atoi(op)==5)
  {
    i = 0;
    while(read(fh, &c, 1) > 0)
      {
        printf("%c", c);
        if(c == '\n')
          i++;
        if(i == 5)
          break;
      }
  }
 if(strcmp(op,"-8")==0)
   {
     i = 0;
     while(read(fh, &c, 1) > 0)
       {
         if(c == '\n')
           i++;
       }
     lseek(fh, 0, SEEK_SET);
     j = 0;
     while(read(fh, &c, 1) > 0)
```

```
        {
          if(c == '\n')
            j++;
          if(j == i-20)
            break;
        }
      while(read(fh, &c, 1) > 0)
        printf("%c", c);
    }
  close(fh);
 }

int main()
 {
  char command[60],t1[20],t2[20],t3[20];
  int n;
  while(1)
  {
    printf("myshell$ ");
    fflush(stdin);
    fgets(command,60,stdin);
    n = sscanf(command,"%s %s %s",t1,t2,t3);

    switch(n)
     {
      case 1:
      char a=t1[0];
      if(a=='q')
        return 0;
      if(!fork())
       {
```

```
      execlp(t1,t1,NULL);
      perror(t1);
     }
    break;
    case 2:
    if(!fork())
     {
      execlp(t1,t1,t2,NULL);
      perror(t1);
     }
    break;
   case 3:
   if(strcmp(t1,"Typelines")==0)
    typeline(t2,t3);
   else
    {
     if(!fork())
      {
       execlp(t1,t1,t2,t3,NULL);
       perror(t1);
      }
    }
   break;
  }
 }
}
```

Output :

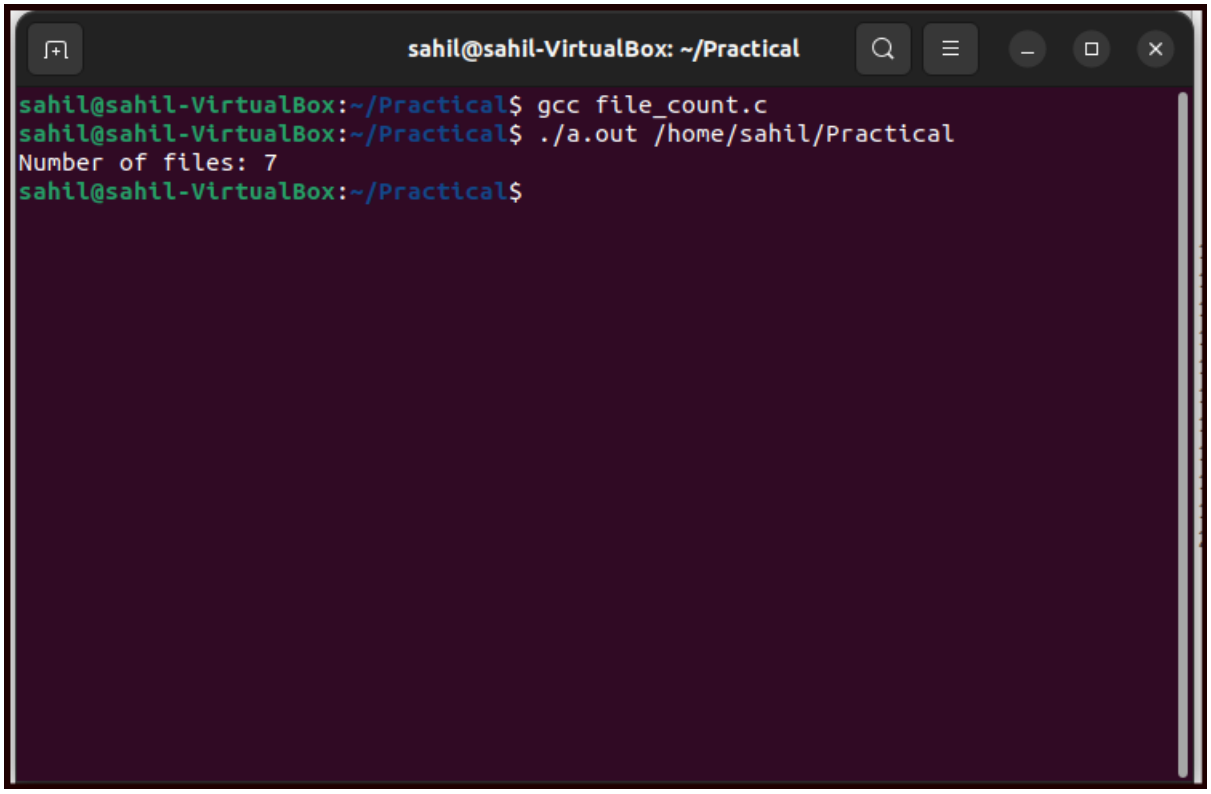


21. Write a program to count the no of files of a specified directory.

Code :

```
#include<stdio.h>
#include<dirent.h>
#include<unistd.h>
void main(int argc , char *argv[])
{
        int cnt = 0;
        DIR *dp;
        struct dirent *entry;
        dp = opendir(argv[1]);
        if (dp  == NULL) {
                printf("Directory does not exists.\n");
                _exit(0);
        }
        while (entry = readdir(dp))
                if (entry->d_type == DT_REG)
                        cnt++;
        printf("Number of files: %d\n", cnt);
        closedir(dp);

}
```

Output :



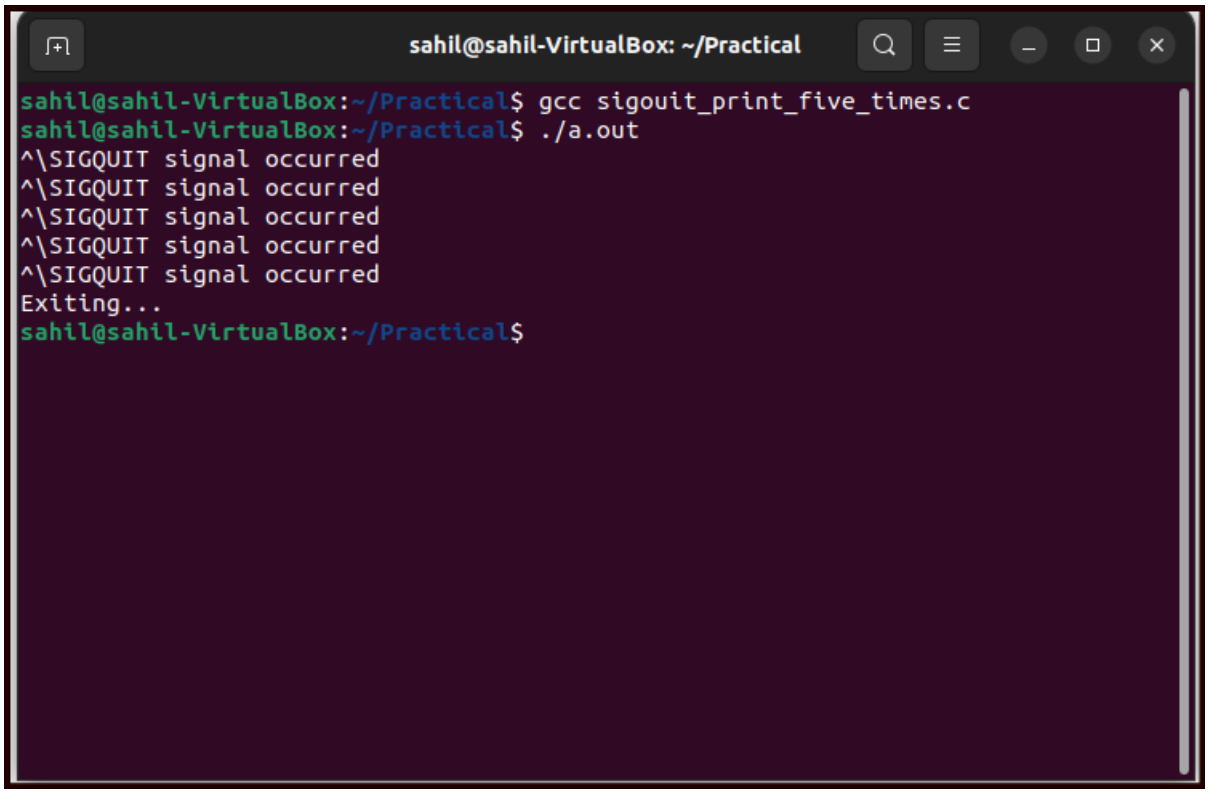
22. Write a program to catch SIGOUIT signal five times and print message 'SIGOUIT signal occurred' every time and exit at sixth occurrence. Also ignore every occurrence of SIGTSTP signal.

Code :

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

int count = 0;
void sig_handler(int sig) {
   printf("SIGQUIT signal occurred\n");
   count++;
   if (count == 5)
   {
```

```
        printf("Exiting...\n");
        _exit(0);
    }
}
int main() {
    signal(SIGQUIT, sig_handler);
    while (1) {
        // Wait for signals
    }
    return 0;
}
```

Output :



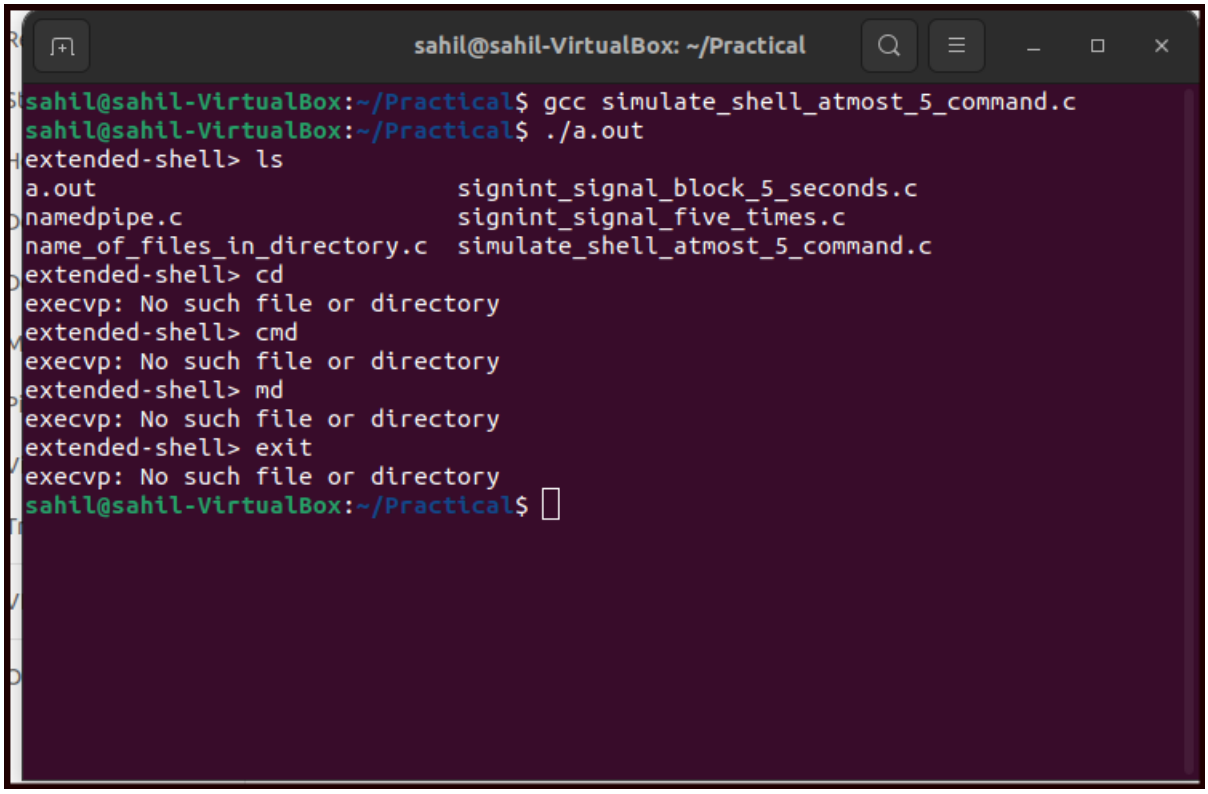
23. Write a program in LINUX to simulate extended shell. Show the prompt and accept standard shell command which will be executed by child process using one of the exec family system calls. Parent process waits until child finished execution the command may consist of

at the most 5 parameters. The process should be repeated till user types “exit”.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
   char line[80];
   char* args[5];
   int status;
   while (1) {
      printf("extended-shell> ");
      if (fgets(line,80, stdin) == NULL)
         break;
      char* token = strtok(line, " \t\n");
      int i = 0;
      while (token != NULL && i < 5) {
         args[i] = token;
         token = strtok(NULL, " \t\n");
         i++;
      }
     args[i] = NULL;
      int pid = fork();
      if (pid == -1) {
         perror("fork");
         exit(1);
      } else if (pid == 0) {
         if (execvp(args[0], args) == -1) {
            perror("execvp");
            exit(1);
         }
      } else
         wait(&status);

      if (strcmp(args[0], "exit") == 0)
         break;
   }
   return 0;
}
```

Output :

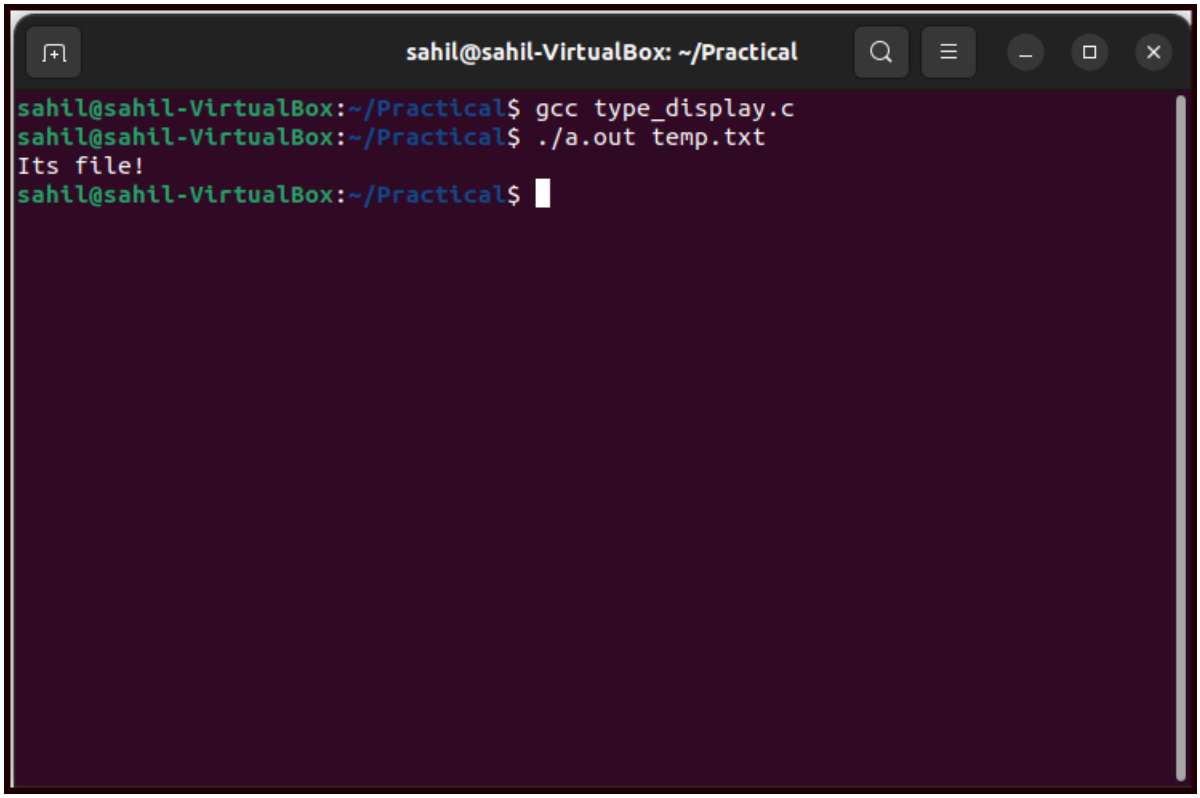


24. Print the type of file where file name is accepted thru command line argument

Code :

```
#include <stdio.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>
void main(int argc , char *argv[])
{
    char *fileName = argv[1];
        struct stat pathStat;
        stat(fileName, &pathStat);

        if (S_ISREG(pathStat.st_mode))
                printf("Its file!\n");
        else if (S_ISDIR(pathStat.st_mode))
                printf("its Directory!\n");
}
```

Output :


sahil@sahil-VirtualBox: ~/Practical
sahil@sahil-VirtualBox:~/Practical$ gcc type_display.c
sahil@sahil-VirtualBox:~/Practical$ ./a.out temp.txt
Its file!
sahil@sahil-VirtualBox:~/Practical$