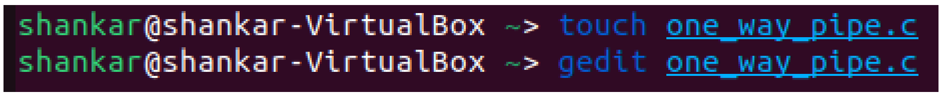
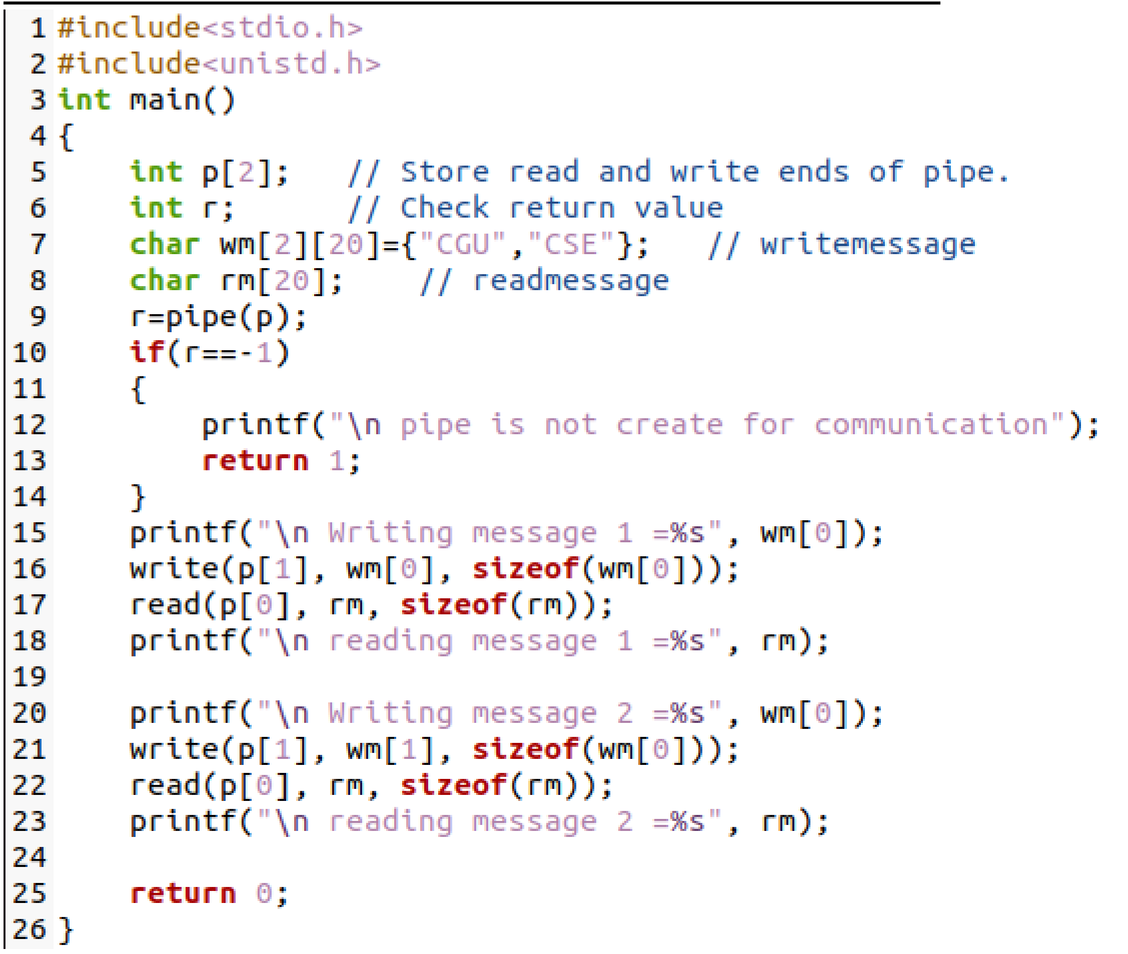
EXPERIMENT:-06

AIM:- IPC (Inter Process Communication) , Thread Manipulation, System Calls and Synchronization.

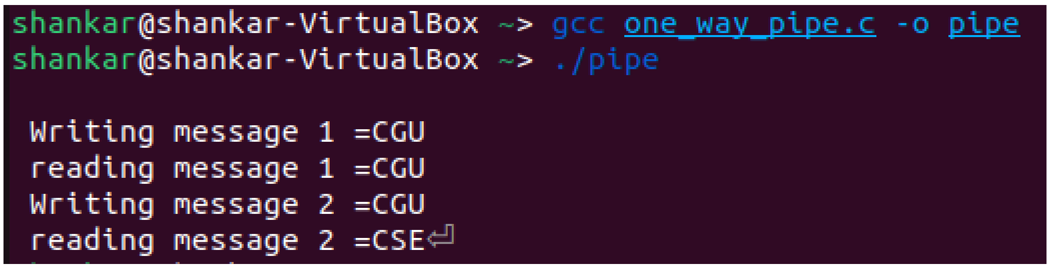
1. Write C program to read and write any 2 messages in one-way pipe between two processes.



```
shankar@shankar-VirtualBox ~> touch one_way_pipe.c
shankar@shankar-VirtualBox ~> gedit one_way_pipe.c
```

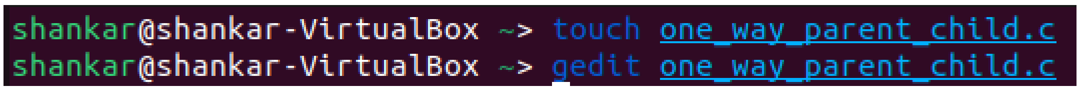


```c
#include<stdio.h>
#include<unistd.h>
int main()
{
    int p[2];    // Store read and write ends of pipe.
    int r;       // Check return value
    char wm[2][20]={"CGU","CSE"};    // writemessage
    char rm[20];     // readmessage
    r=pipe(p);
    if(r==-1)
    {
        printf("\n pipe is not create for communication");
        return 1;
    }
    printf("\n Writing message 1 =%s", wm[0]);
    write(p[1], wm[0], sizeof(wm[0]));
    read(p[0], rm, sizeof(rm));
    printf("\n reading message 1 =%s", rm);

    printf("\n Writing message 2 =%s", wm[0]);
    write(p[1], wm[1], sizeof(wm[0]));
    read(p[0], rm, sizeof(rm));
    printf("\n reading message 2 =%s", rm);

    return 0;
}
```

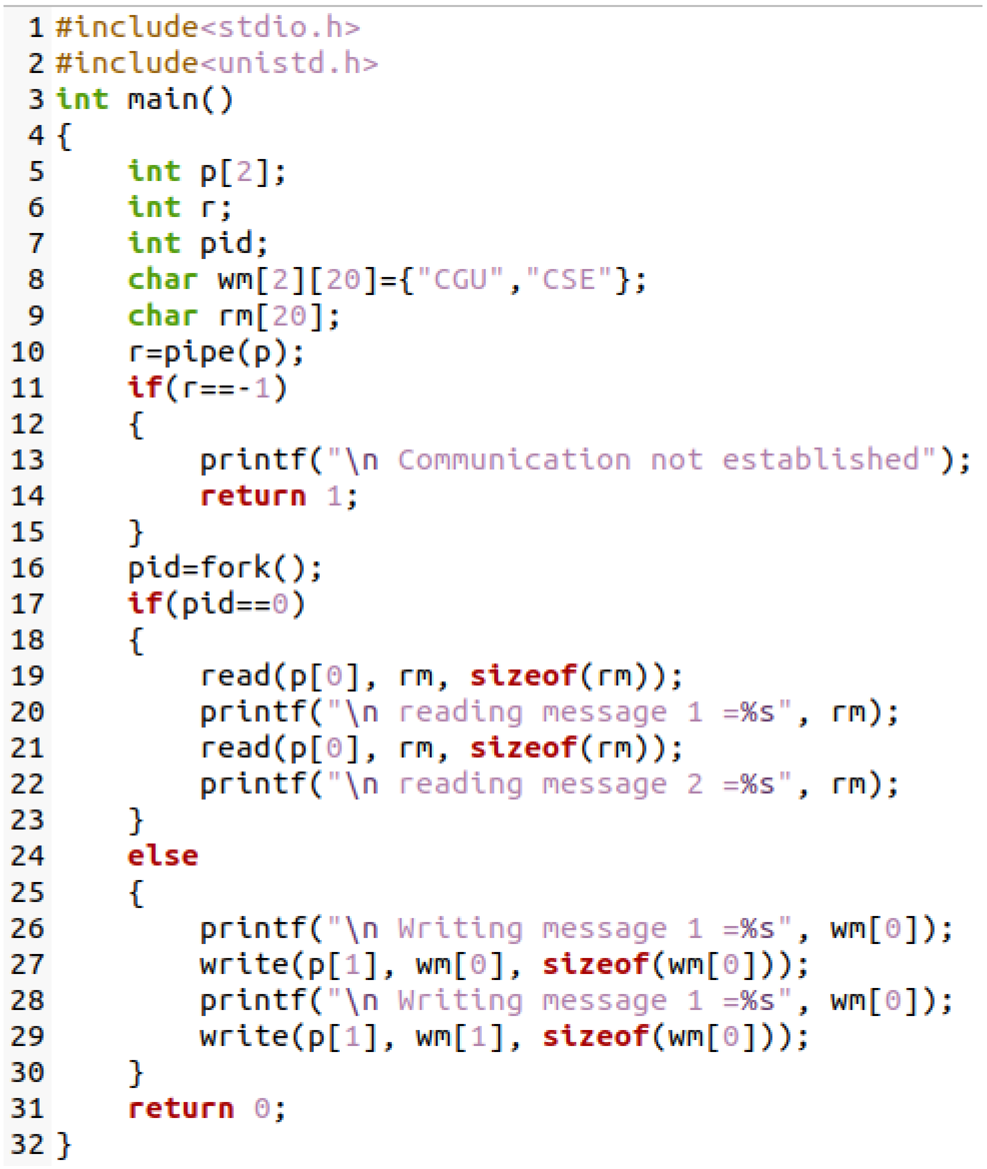


```
shankar@shankar-VirtualBox ~> gcc one_way_pipe.c -o pipe
shankar@shankar-VirtualBox ~> ./pipe

 Writing message 1 =CGU
 reading message 1 =CGU
 Writing message 2 =CGU
 reading message 2 =CSE⏎
```

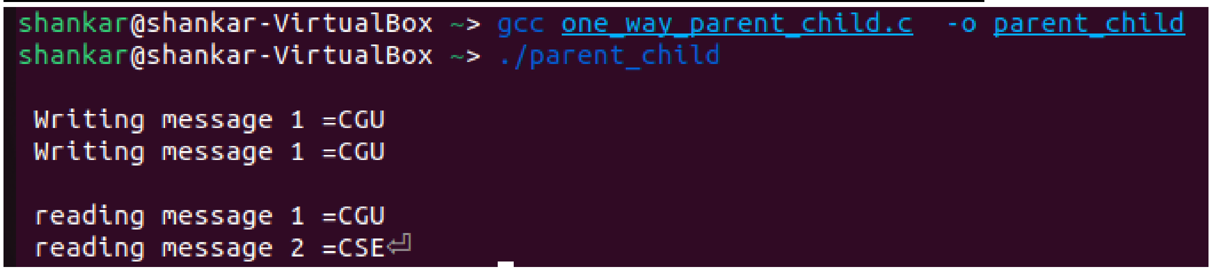
2. Write a C program to write and read any 2 messages though single pipe between parent and child process.



```
shankar@shankar-VirtualBox ~> touch one_way_parent_child.c
shankar@shankar-VirtualBox ~> gedit one_way_parent_child.c
```

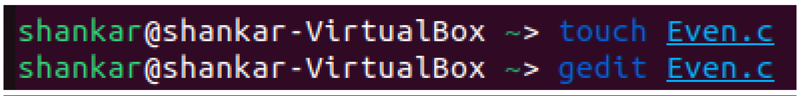


```c
#include<stdio.h>
#include<unistd.h>
int main()
{
    int p[2];
    int r;
    int pid;
    char wm[2][20]={"CGU","CSE"};
    char rm[20];
    r=pipe(p);
    if(r==-1)
    {
        printf("\n Communication not established");
        return 1;
    }
    pid=fork();
    if(pid==0)
    {
        read(p[0], rm, sizeof(rm));
        printf("\n reading message 1 =%s", rm);
        read(p[0], rm, sizeof(rm));
        printf("\n reading message 2 =%s", rm);
    }
    else
    {
        printf("\n Writing message 1 =%s", wm[0]);
        write(p[1], wm[0], sizeof(wm[0]));
        printf("\n Writing message 1 =%s", wm[0]);
        write(p[1], wm[1], sizeof(wm[0]));
    }
    return 0;
}
```

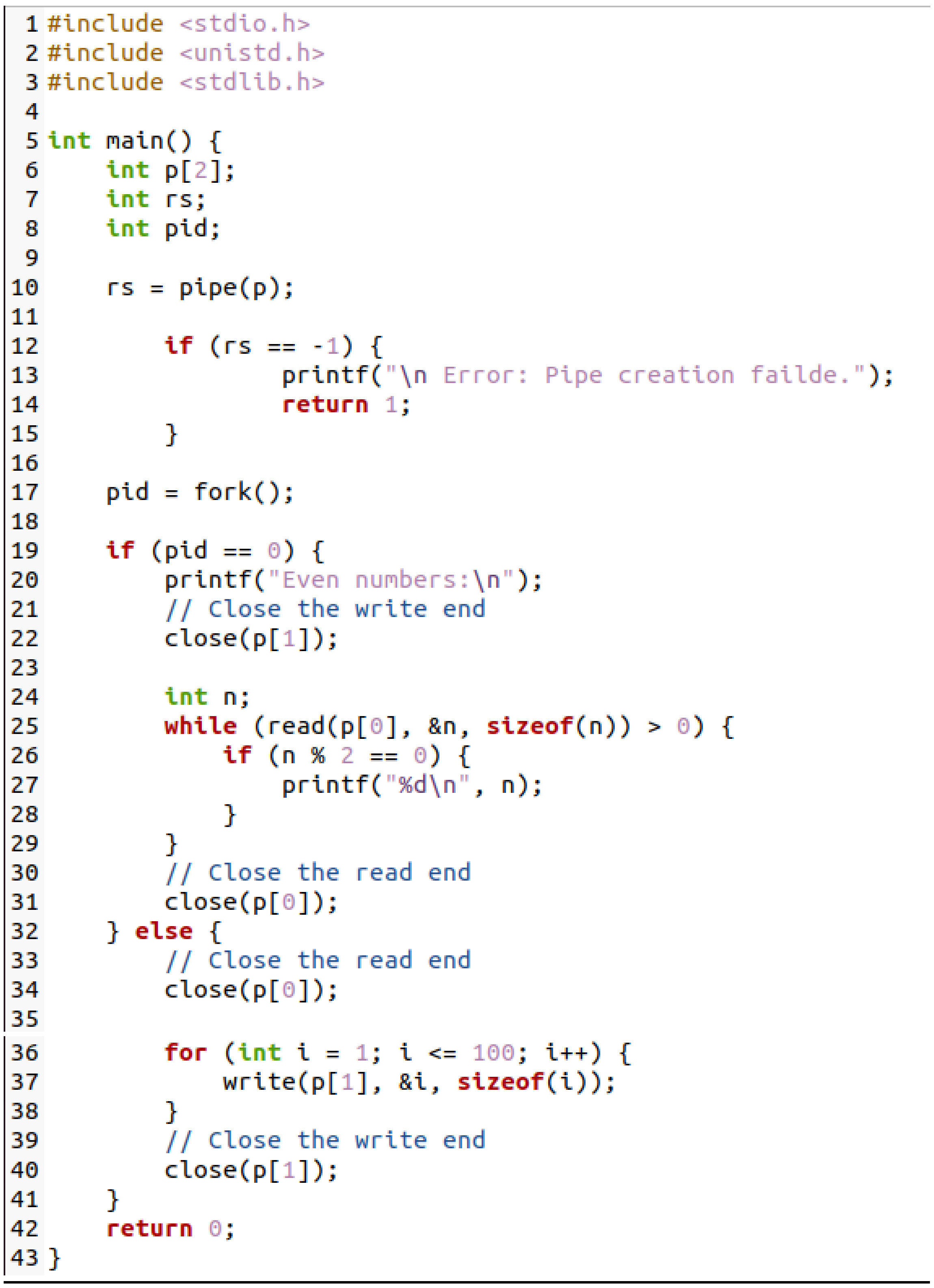


```
shankar@shankar-VirtualBox ~> gcc one_way_parent_child.c  -o parent_child
shankar@shankar-VirtualBox ~> ./parent_child

 Writing message 1 =CGU
 Writing message 1 =CGU

 reading message 1 =CGU
 reading message 2 =CSE⏎
```

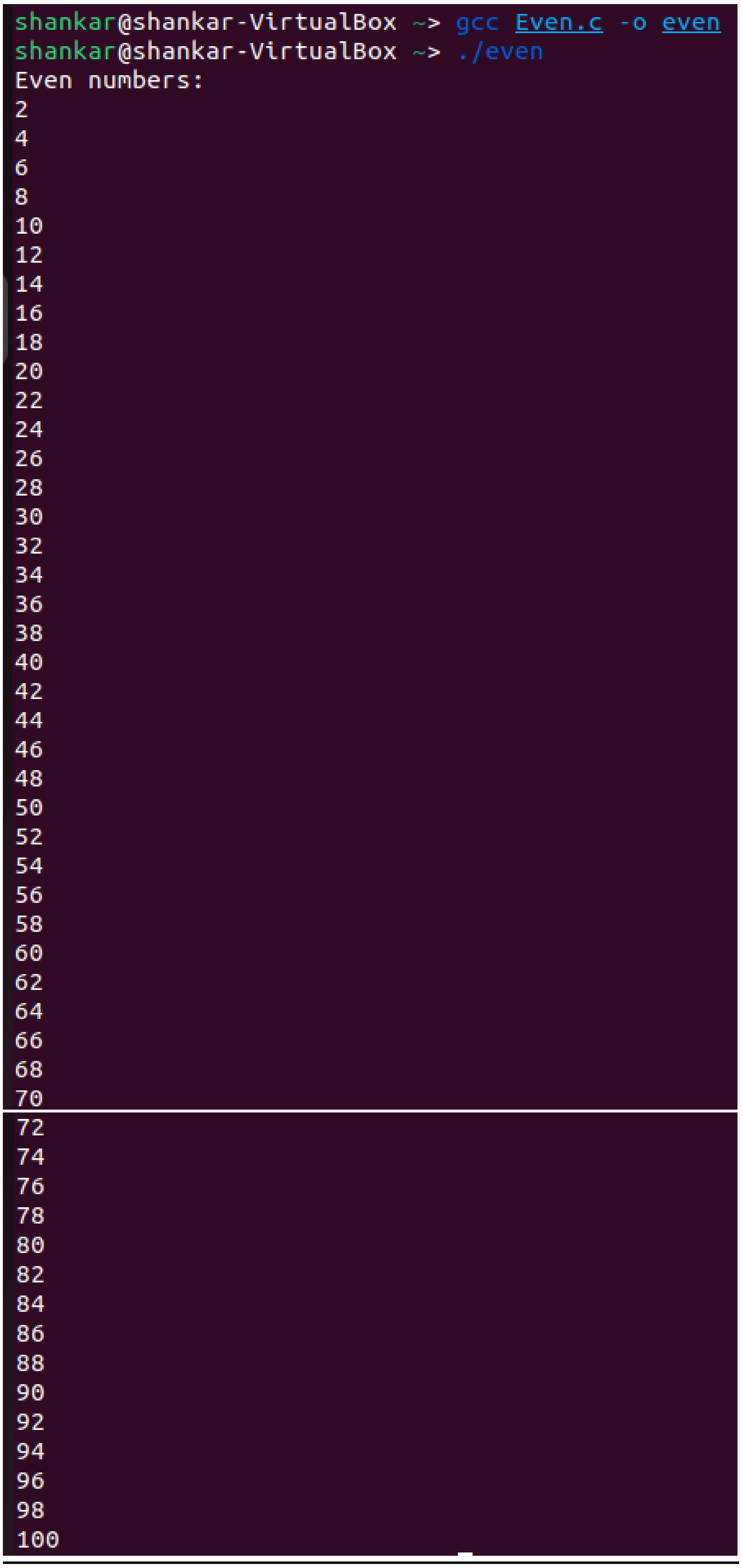
3. C program to print all even no. between 1 to 100 using IPC through pipe.



```
shankar@shankar-VirtualBox ~> touch Even.c
shankar@shankar-VirtualBox ~> gedit Even.c
```

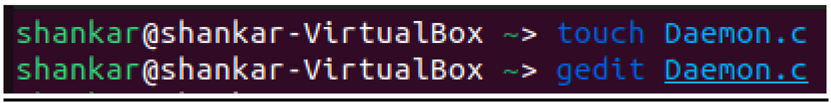


```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    int p[2];
    int rs;
    int pid;

    rs = pipe(p);

        if (rs == -1) {
                printf("\n Error: Pipe creation failde.");
                return 1;
        }

    pid = fork();

    if (pid == 0) {
        printf("Even numbers:\n");
        // Close the write end
        close(p[1]);

        int n;
        while (read(p[0], &n, sizeof(n)) > 0) {
            if (n % 2 == 0) {
                printf("%d\n", n);
            }
        }
        // Close the read end
        close(p[0]);
    } else {
        // Close the read end
        close(p[0]);

        for (int i = 1; i <= 100; i++) {
            write(p[1], &i, sizeof(i));
        }
        // Close the write end
        close(p[1]);
    }
    return 0;
}
```

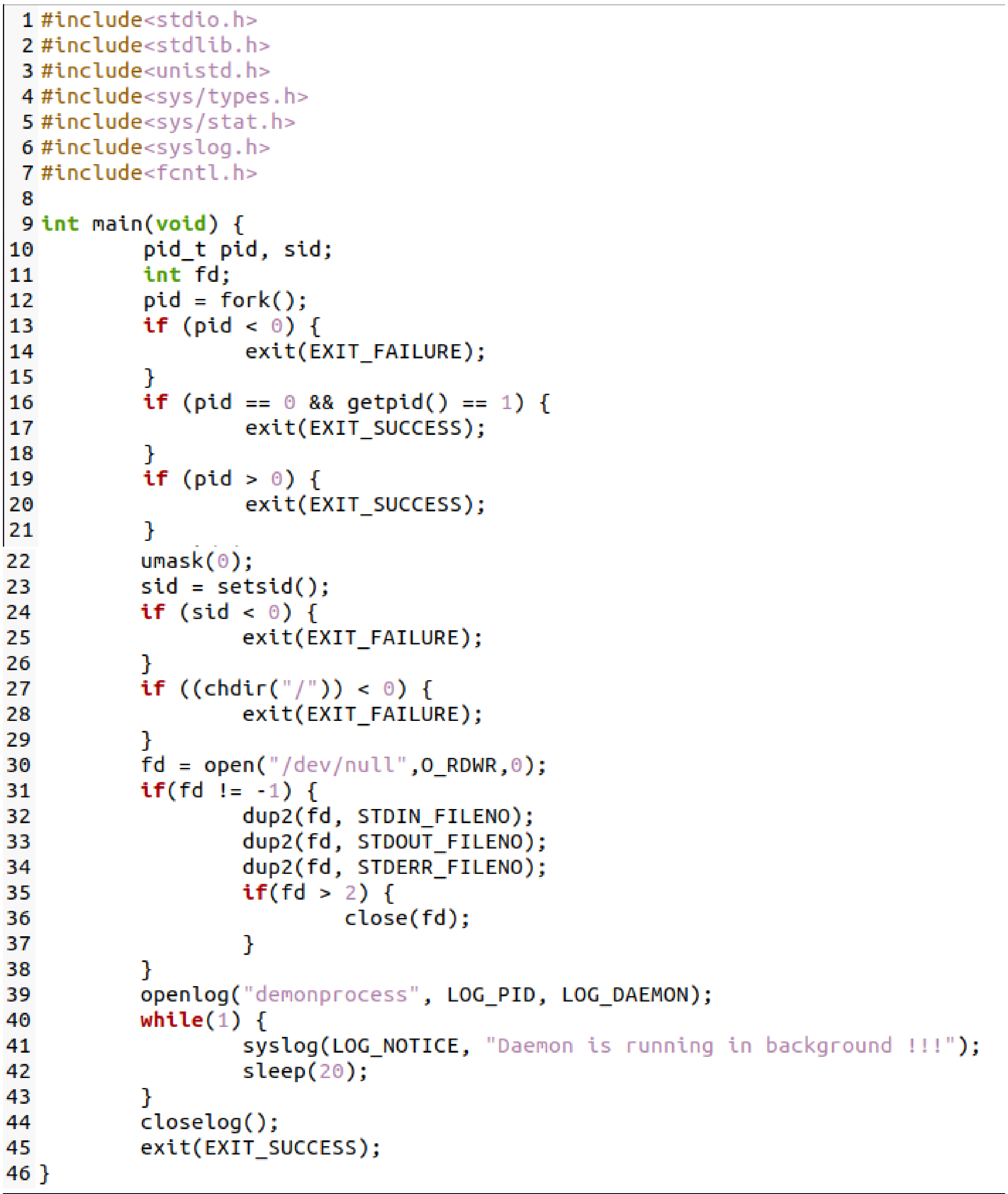


```
shankar@shankar-VirtualBox ~> gcc Even.c -o even
shankar@shankar-VirtualBox ~> ./even
Even numbers:
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
96
98
100
```

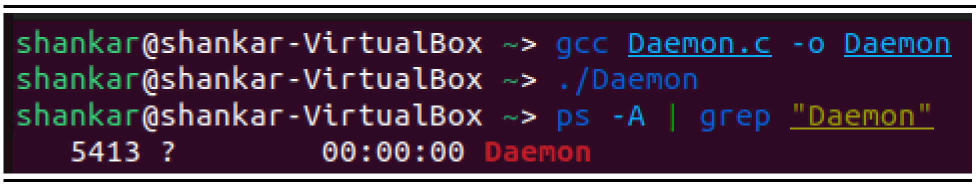
4. Daemon process creation using fork.



```
shankar@shankar-VirtualBox ~> touch Daemon.c
shankar@shankar-VirtualBox ~> gedit Daemon.c
```

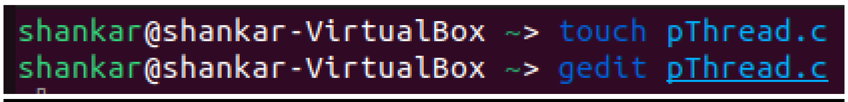


```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<syslog.h>
#include<fcntl.h>

int main(void) {
        pid_t pid, sid;
        int fd;
        pid = fork();
        if (pid < 0) {
                exit(EXIT_FAILURE);
        }
        if (pid == 0 && getpid() == 1) {
                exit(EXIT_SUCCESS);
        }
        if (pid > 0) {
                exit(EXIT_SUCCESS);
        }
        umask(0);
        sid = setsid();
        if (sid < 0) {
                exit(EXIT_FAILURE);
        }
        if ((chdir("/")) < 0) {
                exit(EXIT_FAILURE);
        }
        fd = open("/dev/null",O_RDWR,0);
        if(fd != -1) {
                dup2(fd, STDIN_FILENO);
                dup2(fd, STDOUT_FILENO);
                dup2(fd, STDERR_FILENO);
                if(fd > 2) {
                        close(fd);
                }
        }
        openlog("demonprocess", LOG_PID, LOG_DAEMON);
        while(1) {
                syslog(LOG_NOTICE, "Daemon is running in background !!!");
                sleep(20);
        }
        closelog();
        exit(EXIT_SUCCESS);
}
```

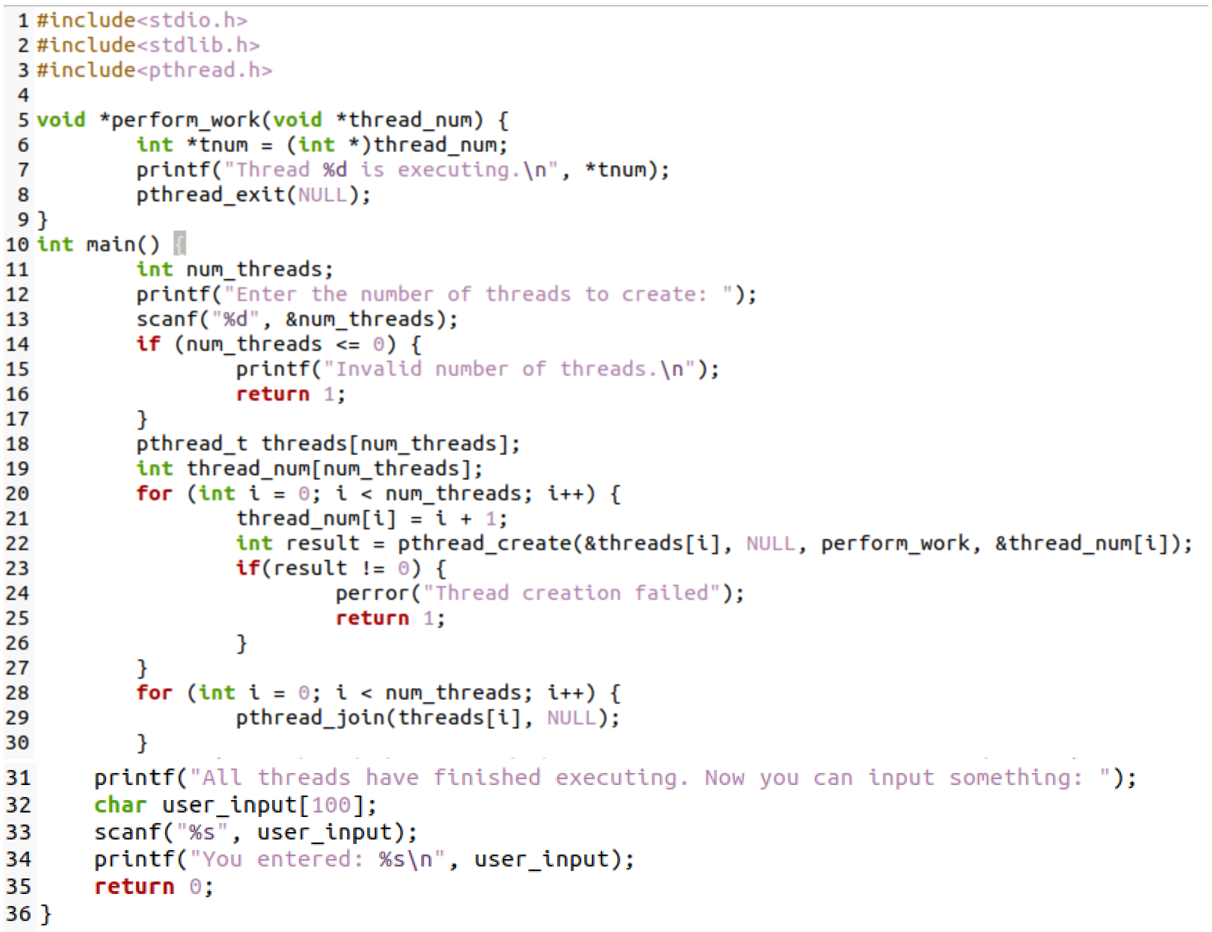


```
shankar@shankar-VirtualBox ~> gcc Daemon.c -o Daemon
shankar@shankar-VirtualBox ~> ./Daemon
shankar@shankar-VirtualBox ~> ps -A | grep "Daemon"
   5413 ?        00:00:00 Daemon
```

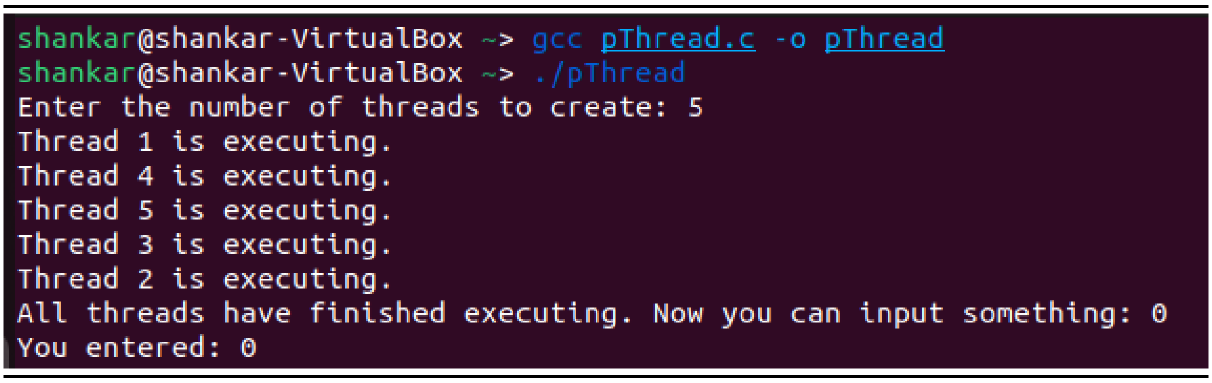
5. A C program to creates five threads. each executing the function perform Work that prints the unique number off this thread to standard Output. (PThread Library).



```
shankar@shankar-VirtualBox ~> touch pThread.c
shankar@shankar-VirtualBox ~> gedit pThread.c
```



```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

void *perform_work(void *thread_num) {
        int *tnum = (int *)thread_num;
        printf("Thread %d is executing.\n", *tnum);
        pthread_exit(NULL);
}
int main() {
        int num_threads;
        printf("Enter the number of threads to create: ");
        scanf("%d", &num_threads);
        if (num_threads <= 0) {
                printf("Invalid number of threads.\n");
                return 1;
        }
        pthread_t threads[num_threads];
        int thread_num[num_threads];
        for (int i = 0; i < num_threads; i++) {
                thread_num[i] = i + 1;
                int result = pthread_create(&threads[i], NULL, perform_work, &thread_num[i]);
                if(result != 0) {
                        perror("Thread creation failed");
                        return 1;
                }
        }
        for (int i = 0; i < num_threads; i++) {
                pthread_join(threads[i], NULL);
        }
    printf("All threads have finished executing. Now you can input something: ");
    char user_input[100];
    scanf("%s", user_input);
    printf("You entered: %s\n", user_input);
    return 0;
}
```



```
shankar@shankar-VirtualBox ~> gcc pThread.c -o pThread
shankar@shankar-VirtualBox ~> ./pThread
Enter the number of threads to create: 5
Thread 1 is executing.
Thread 4 is executing.
Thread 5 is executing.
Thread 3 is executing.
Thread 2 is executing.
All threads have finished executing. Now you can input something: 0
You entered: 0
```