**Commands**

* apt -get update && apt -get upgrade
* uname
* uname -r
* uname -a
* history **for listing recently used commands**
* clear **for clear the terminal**
* ls **for list Directory**
* sana@sana/$  **to access user directory**
* sana@sana~$  **to access root directory**
* cd(catch Directory)  **to get back from user/root dirctory**
* touch filename **to create file**
* rm filename **to remove file**
* mkdir folder name  **to make folder**
* rmdir folder name **to del folder**
* Cat file name  **to read the content of file**

**Permission**

**Admin, group, public**

**7 6 5**

**excute (1) x**

**write (2) w**

**Read (4) r**

ls -l filename.ext **to check permission**

**Result:** -rw-rw-r-- 1

-rw rw- r--

chmod 765 filename.ext **to change your file permission**

**Result**

-rwxrw-r-xcd

1. **To write c Program for system call in terminal**

nano first.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

write(1, "hello",5);

}

Ctrl+s **for save code**

ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc first.c **to execute c code**

./a.out **to execute object file of c code**

1. **Write c program for get Process id**

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

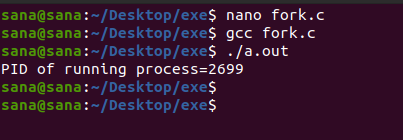
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for fork system call**

**Fork():**

Fork() system call is used to create separate/duplicate/ child process with same content of file but assign different PIDs

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Fork();

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

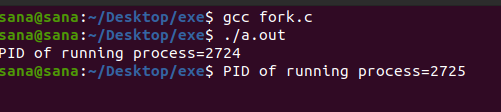
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for multiple fork system call**

nano fork.c  **to write c code**

**we reached GNU platform**

#include<unistd.h>

int main()

{

Fork();

Fork();

Fork();

Printf(PID of running process=%d\n”, getpid());

Return 0;

}

Ctrl+s **for save code**

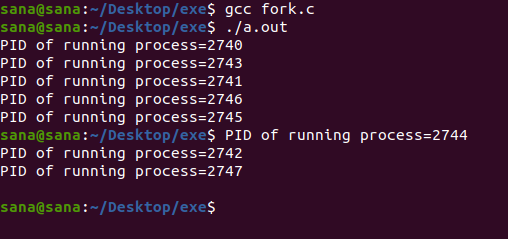
ctrl+x **to exit from GNU platform**

**we reached back to terminal**

gcc fork.c **to execute c code**

./exe1.c **to execute object file of c code**

**Output:**



1. **Write c program for exec system call**

**Exec() system call:**

Exec() system call is used to create separate/duplicate/ child process with different content of file but assign same PIDs

nano exe2.c

**we reached GNU platform**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[])

{

Printf(“we are in exe2.c\n”);

Printf(PID of exe2.c=%d\n”, getpid());

Return 0;

}

nano exe1.c

**we reached GNU platform**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[])

{

Printf(PID of exe1.c=%d\n”, getpid());

Char \* args={“hello”, “programmer”,null};

Execv(“./exe2.c”, args);

Printf(“back to exe1.c”);

Return 0;

}

Ctrl+s **for save code**

ctrl+x **to exit from GNU platform**

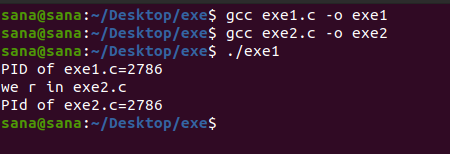
**we reached back to terminal**

gcc exe1.c -o exe1 **to execute c code**

gcc exe2.c -o exe2 **to execute c code**

./exe1 **to execute object file of c code**

**Output:**



1. **Write c program for fork and wait system call**

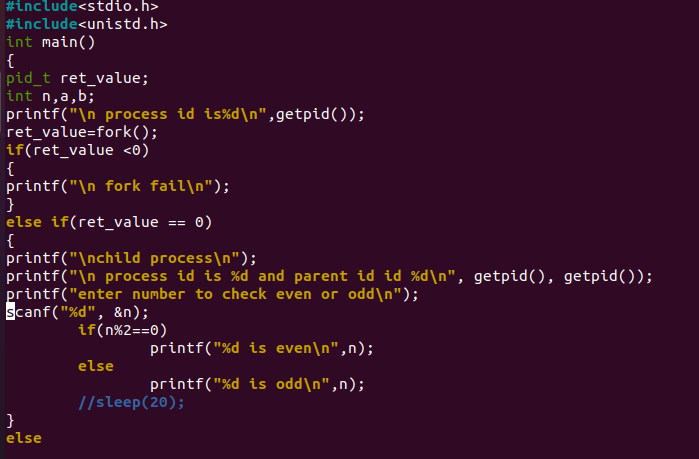
**Fork():**Fork() system call is used to create separate/duplicate/ child process with same content of file but assign different PIDs

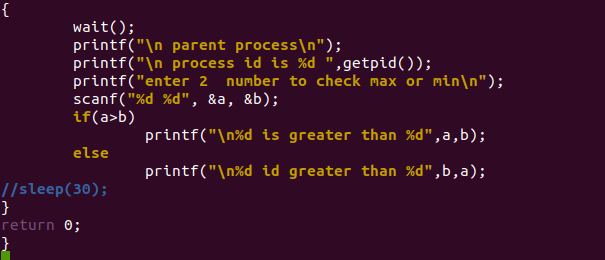
**Wait(): Parent process should execute only when the child process has been terminated**

**Without Wait() system call Child**

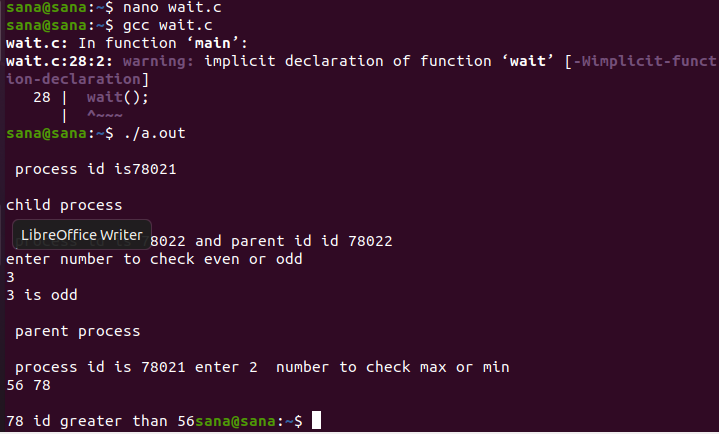
nano wait.c

**we reached GNU platform**





**Output:**



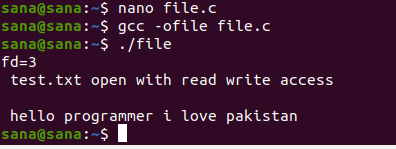
1. **Open, read write close lseek system call**

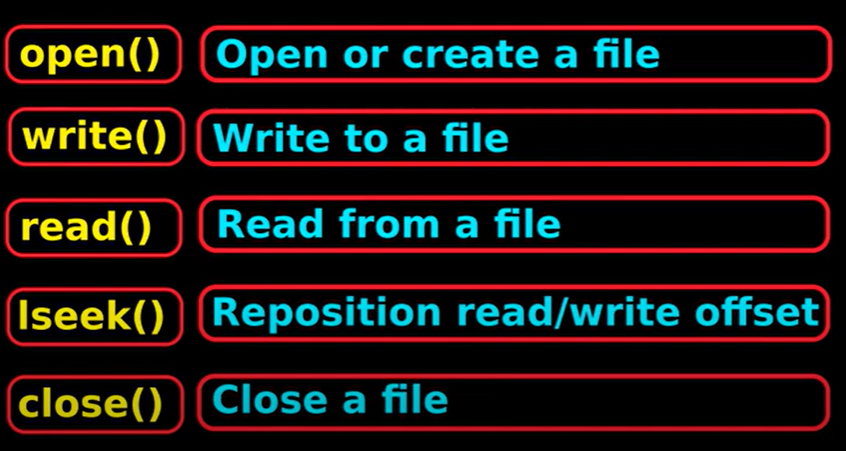
nano file.c

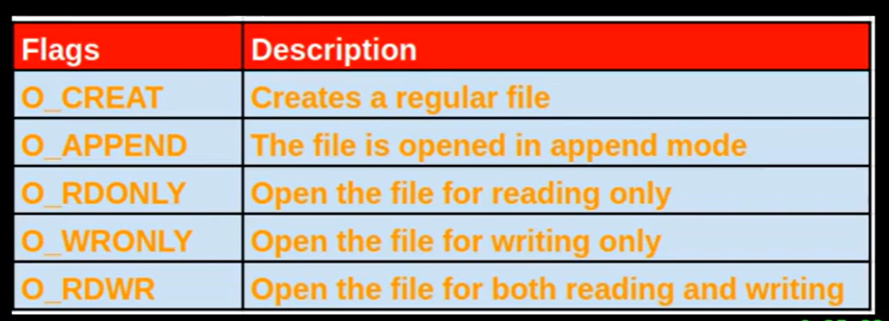
**we reached GNU platform**

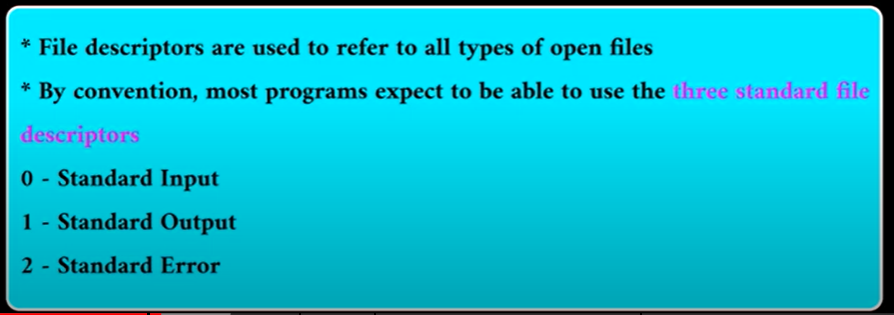


**Output:**

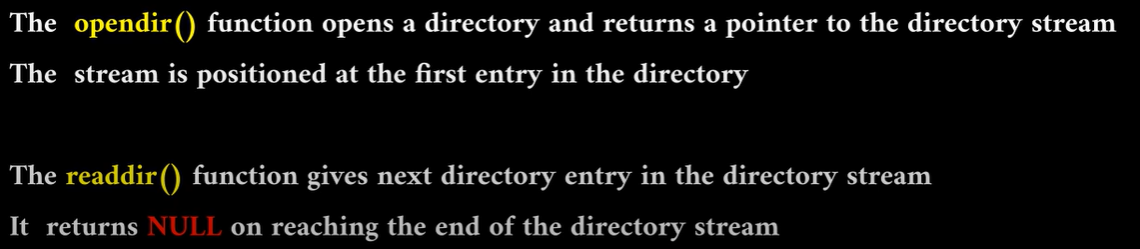






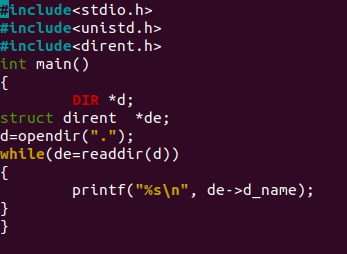


1. **Opendir() readdir() systemcall**

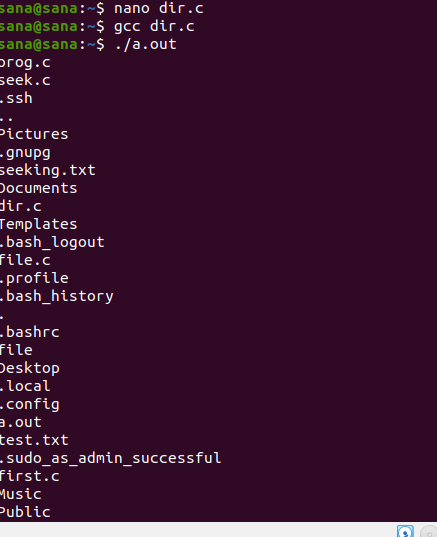


nano dir.c

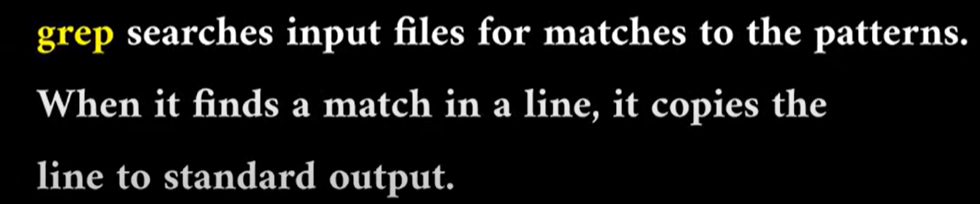
**we reached GNU platform**



**Output:**



1. **Grep Command**





**PRODUCER CONSUMER PROBLEM USING SEMAPHORE**

**AIM:** To write a C program to implement the Producer & consumer Problem (Semaphore)

**ALGORITHM:**

Step 1: The Semaphore mutex, full & empty are initialized.

Step 2: In the case of producer process

i) Produce an item in to temporary variable.

ii) If there is empty space in the buffer check the mutex value for enter into the critical section.

iii) If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.

Step 3: In the case of consumer process

i) It should wait if the buffer is empty

ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer

iii) Signal the mutex value and reduce the empty value by 1.

iv) Consume the item.

Step 4: Print the result

**PROGRAM :**

nano prco.c

#define BUFFERSIZE 10

int mutex,n,empty,full=0,item,item1;

int buffer[20];

int in=0,out=0,mutex=1;

void wait(int s)

{

while(s<0)

{

printf(“\nCannot add an item\n”);

exit(0);

}

s--;

}

void signal(int s)

{

s++;

}

void producer()

{

do

{

wait (empty);

wait(mutex);

printf(“\nEnter an item:”);

scanf(“%d”,&item);

buffer[in]=item;

in=in+1;

signal(mutex);

signal(full);

}

while(in<n);

}

void consumer()

{

do

{

wait(full);

wait(mutex);

item1=buffer[out];

printf(“\nConsumed item =%d”,item1);

out=out+1;

signal(mutex);

signal(empty);

}

while(out<n);

}

void main()

{

printf(“Enter the value of n:”);

scanf(“%d “,&n);

empty=n;

while(in<n)

producer();

while(in!=out)

consumer();

}

**OUTPUT**:

$ gcc prco.c

$ a.out

Enter the value of n :3

Enter the item:2

Enter the item:5

Enter the item:9

consumed item=2

consumed item=5

consumed item=9

$

**RESULT:**

Thus the program for solving producer and consumer problem using semaphore was executed

successfully