**PROGRAM 1**

**Write a program that demonstrates how two processes can share a variable using semaphore.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

int shared\_var = 0;

#define SEM\_KEY 1234

struct sembuf increment\_op = {0, 1, 0};

struct sembuf decrement\_op = {0, -1, 0};

int main() {

int sem\_id = semget(SEM\_KEY, 1, IPC\_CREAT | 0666);

if (sem\_id == -1) {

perror("semget");

exit(1);

}

if (semctl(sem\_id, 0, SETVAL, 1) == -1) {

perror("semctl");

exit(1);

}

pid\_t pid1 = fork();

if (pid1 == -1) {

perror("fork");

exit(1);

}

if (pid1 == 0) {

if (semop(sem\_id, &increment\_op, 1) == -1) {

perror("semop");

exit(1);

}

shared\_var++;

if (semop(sem\_id, &decrement\_op, 1) == -1) {

perror("semop");

exit(1);

}

exit(0);

}

pid\_t pid2 = fork();

if (pid2 == -1) {

perror("fork");

exit(1);

}

if (pid2 == 0) {

if (semop(sem\_id, &decrement\_op, 1) == -1) {

perror("semop");

exit(1);

}

shared\_var--;

if (semop(sem\_id, &increment\_op, 1) == -1) {

perror("semop");

exit(1);

}

exit(0);

}

waitpid(pid1, NULL, 0);

waitpid(pid2, NULL, 0);

printf("The final value of the shared variable is %d\n", shared\_var);

if (semctl(sem\_id, 0, IPC\_RMID) == -1) {

perror("semctl");

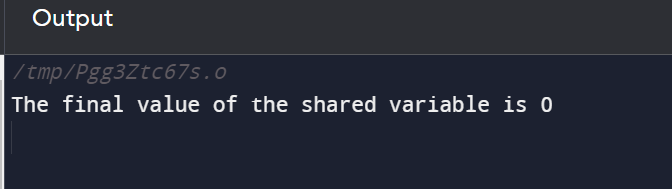
exit(1);

}

return 0;

}

**OUTPUT**

****

**PROGRAM 2**

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

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

--mutex;

++full;

--empty;

x++;

printf("\nProducer produces""item %d",x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

printf("\nConsumer consumes ""item %d",x);

x--;

++mutex;

}

int main()

{

int n, i;

printf("\n1. Press 1 for Producer""\n2. Press 2 for Consumer""\n3. Press 3 for Exit");

#pragma omp critical

for (i = 1; i > 0; i++)

{

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n)

{

case 1:

if ((mutex == 1)

&& (empty != 0))

{

producer();

}

else

{

printf("Buffer is full!");

}

break;

case 2:

if ((mutex == 1)

&& (full != 0)) {

consumer();

}

else

{

printf("Buffer is empty!");

}

break;

case 3:

exit(0);

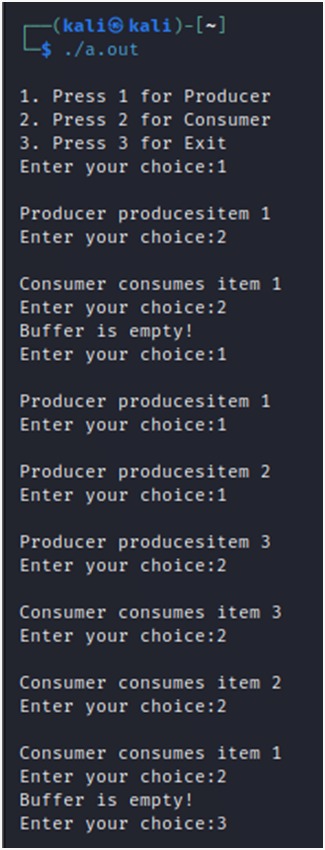
break;

}

}

}

**OUTPUT**

****