OPERATING SYSTEM

ASSIGNMENT simulation Based

Report

**NAME** : K Pavan kumar **EmailID**:kichagaripavan123 @gmail.com

**Reg.no/Roll.no** : 11705490/15 **Sec**:EE034

**GitHub Link** :

Question.no – 15

**18.** This problem demonstrates the use of semaphores to coordinate three types ofprocesses.6 Santa Claus sleeps in his shop at the North Pole and can only be wakened by either (1) all nine reindeer being back from their vacation in the South Pacific, or(2) some of the elves having difficulties making toys; to allow Santa to get some sleep, the elves can only wake him when three of them have problems .When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop’s door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready. (It is assumed that the reindeer do not want to leave the tropics, and therefore they stay there until the last possible moment.) The last reindeer to arrive must get Santa while the others wait in a warming hut before being harnessed to the sleigh. Using synchronization tools like locks, semaphores and monitors provide a solution to this problem.

CODE:

#include <pthread.h>

#include <stdlib.h>

#include <assert.h>

#include <unistd.h>

#include <stdio.h>

#include <stdbool.h>

#include <semaphore.h>

pthread\_t \*CreateThread(void \*(\*f)(void \*), void \*a)

{

pthread\_t \*t = malloc(sizeof(pthread\_t));

assert(t != NULL);

int ret = pthread\_create(t, NULL, f, a);

assert(ret == 0);

return t;

}

static const int N\_ELVES = 10;

static const int N\_REINDEER = 9;

static int elves;

static int reindeer;

static sem\_t santaSem;

static sem\_t reindeerSem;

static sem\_t elfTex;

static sem\_t mutex;

void \*SantaClaus(void \*arg)

{

printf("Santa Claus: Hoho, here I am\n");

while (true)

{

sem\_wait(&santaSem);

sem\_wait(&mutex);

if (reindeer == N\_REINDEER)

{

int r;

printf("Santa Claus: preparing sleigh\n");

for ( r = 0; r < N\_REINDEER; r++)

sem\_post(&reindeerSem);

printf("Santa Claus: make all kids in the world happy\n");

reindeer = 0;

}

else if (elves == 3)

{

printf("Santa Claus: helping elves\n");

}

sem\_post(&mutex);

}

return arg;

}

void \*Reindeer(void \*arg)

{

int id = (int)arg;

printf("This is reindeer %d\n", id);

while (true)

{

sem\_wait(&mutex);

reindeer++;

if (reindeer == N\_REINDEER)

sem\_post(&santaSem);

sem\_post(&mutex);

sem\_wait(&reindeerSem);

printf("Reindeer %d getting hitched\n", id);

sleep(20);

}

return arg;

}

void \*Elve(void \*arg)

{

int id = (int)arg;

printf("This is elve %d\n", id);

while (true)

{

bool need\_help = random() % 100 < 10;

if (need\_help)

{

sem\_wait(&elfTex);

sem\_wait(&mutex);

elves++;

if (elves == 3)

sem\_post(&santaSem);

else

sem\_post(&elfTex);

sem\_post(&mutex);

printf("Elve %d will get help from Santa Claus\n", id);

sleep(10);

sem\_wait(&mutex);

elves--;

if (elves == 0)

sem\_post(&elfTex);

sem\_post(&mutex);

}

// Do some work

printf("Elve %d at work\n", id);

sleep(2 + random() % 5);

}

return arg;

}

int main(int ac, char \*\*av)

{

elves = 0;

reindeer = 0;

sem\_init(&santaSem, 0, 0);

sem\_init(&reindeerSem, 0, 0);

sem\_init(&elfTex, 0, 1);

sem\_init(&mutex, 0, 1);

pthread\_t \*santa\_claus = CreateThread(SantaClaus, 0);

pthread\_t \*reindeers[N\_REINDEER];

for (int r = 0; r < N\_REINDEER; r++)

reindeers[r] = CreateThread(Reindeer, (void \*)r + 1);

pthread\_t \*elves[N\_ELVES];

for (int e = 0; e < N\_ELVES; e++)

elves[e] = CreateThread(Elve, (void \*)e + 1);

int ret = pthread\_join(\*santa\_claus, NULL);

assert(ret == 0);

}

}

**Explanation : (**In terms of os)

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

* **Wait**

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

SEMAPHORE S;

wait(S)

{

S.value = S.value -1;

if ( S.value < 0 )

{

add this process to S.L; block;

} }

* **Signal**

The signal operation increments the value of its argument S.

SEMAPHORE S;

signal(S)

{

s.value = s.value + 1;

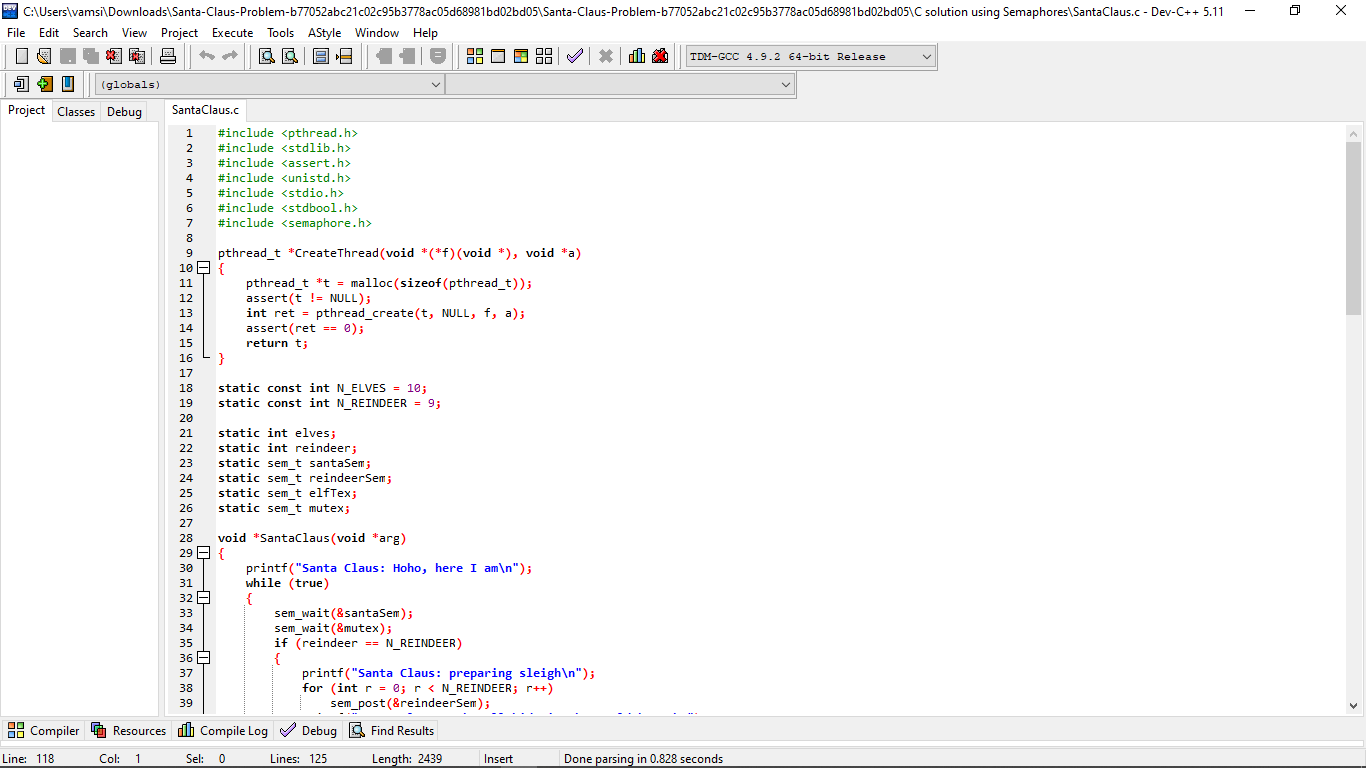
if ( s.value <= 0 )

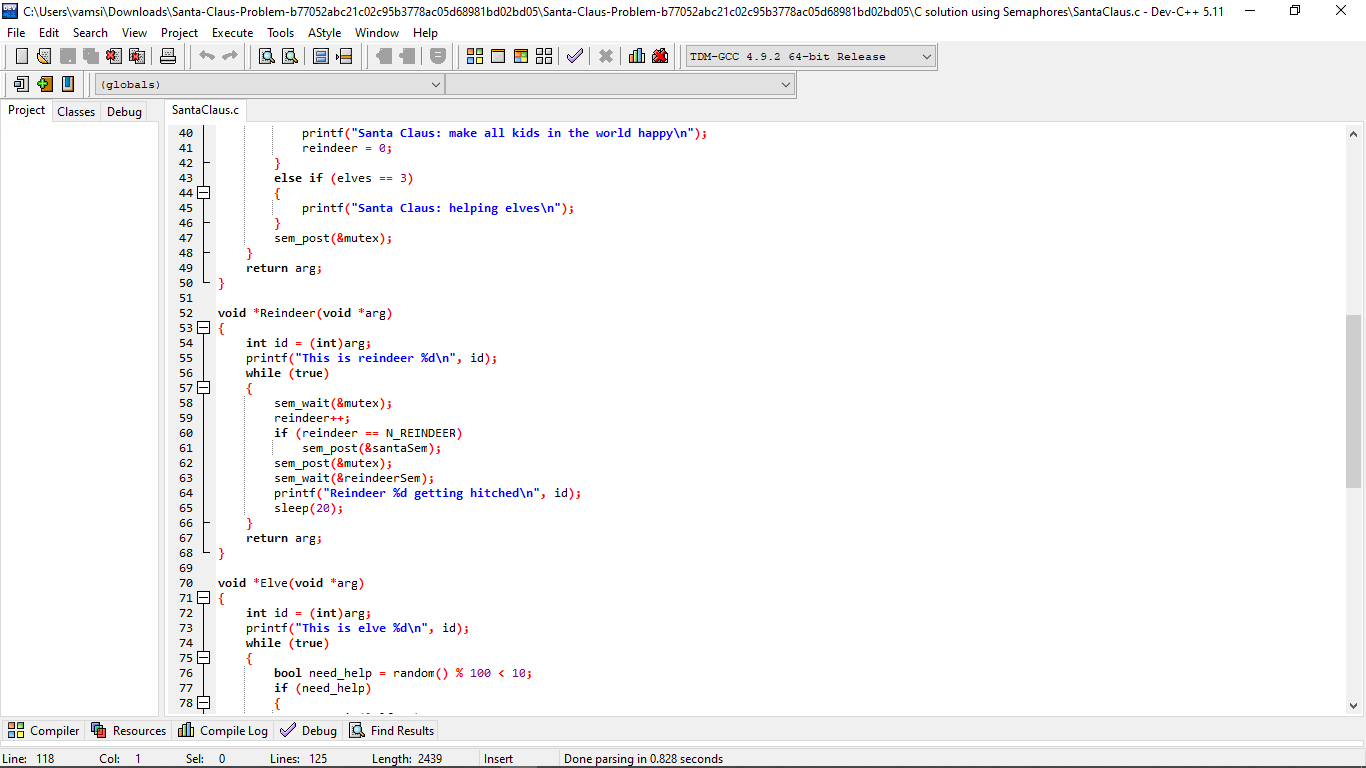
{

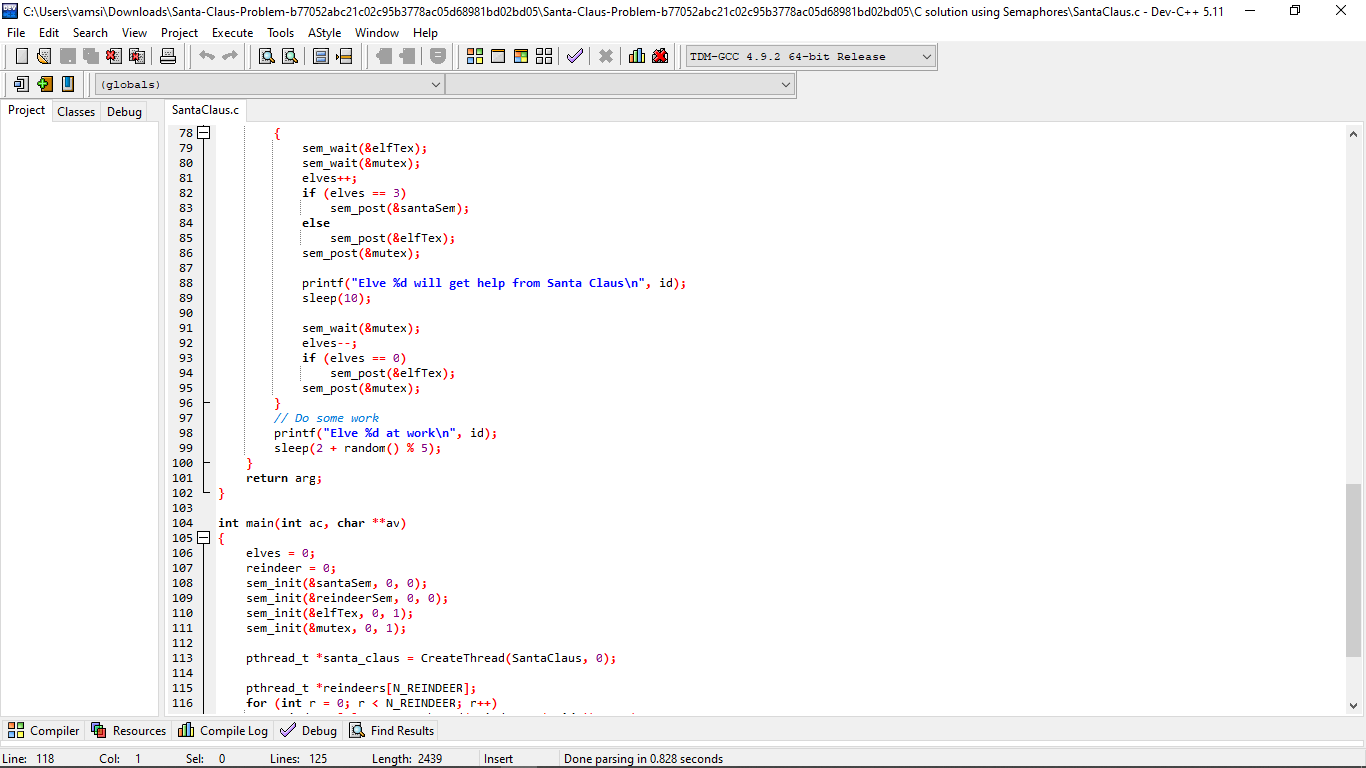
remove a process P from s.L; wakeup(P);

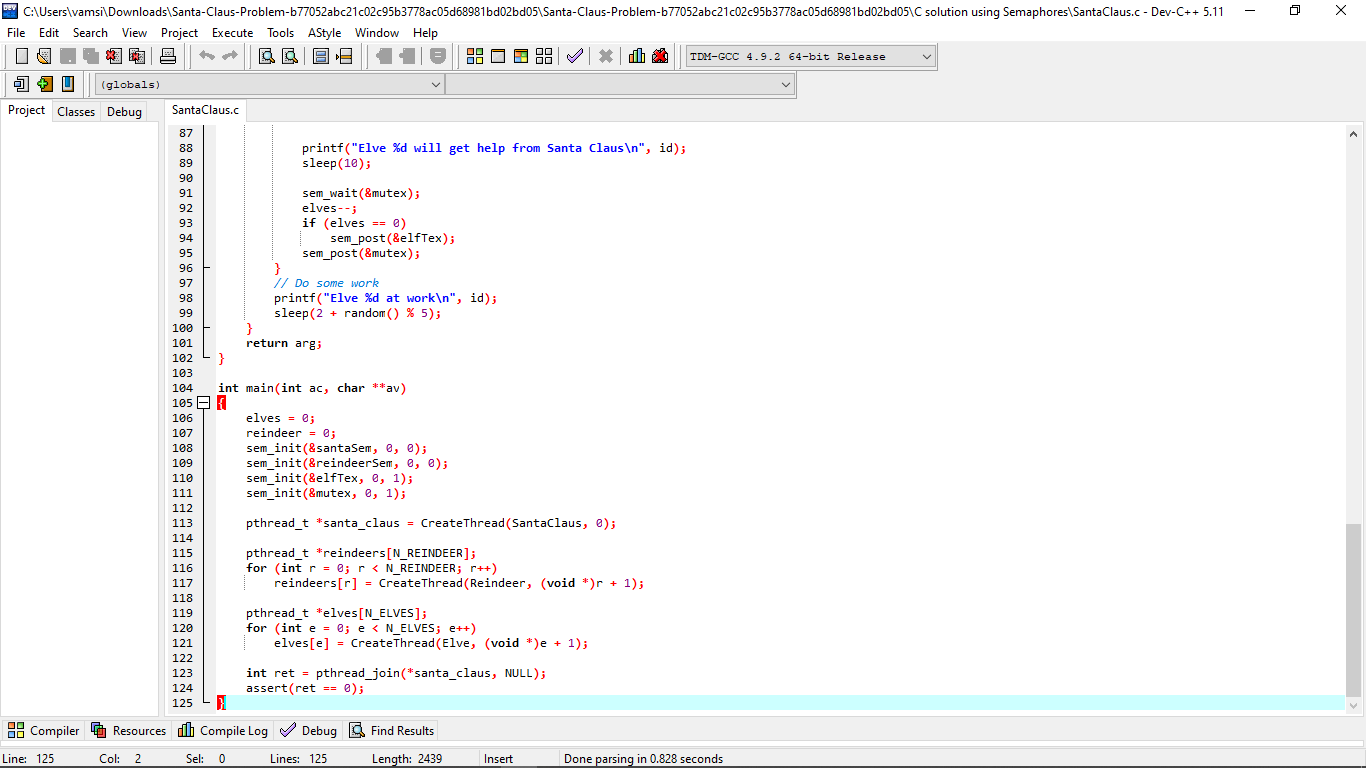
} }

**ALGORITHM :**









The following methods can be called by:

* Santa: helpElves( ), prepareSleigh( )
* Reindeer: getHitch( )
* Elves: getHelp( )

The methods getHitch ( ) and getHelp ( ) are thread safe and can be called outside the critical sections. As per instruction give the problem should be solved using Semaphores.

* The Santa must call prepareSleigh ( ) after the arrival of ninth reindeer and all these nine reindeers must call getHitch ( ).
* The Santa must invoke helpElves( ) after the arrival of 3rd Elf and all the three elves invokes getHelp( )
* All the three elves must invoke getHelp ( ) before any other elves enters.

Initialization of variables (global) required for the program:

* int num\_elf =0;
* int num\_reindeer=0;
* santaSema=Semaphore(0);
* reindeerSema=Semaphore(0);
* elfSema=Semaphore(1);
* lock=Semaphore(1);