LOVELY PROFESSIONAL UNIVERSITY

OPERATING SYSTEMS

CSE-316-Assignment

SUBMITTED BY : K.BALAKISHAN

REG.NO : 11702854

ROLL.NO : B39

EMAIL ID : kishankorangi14@gmail.com

QUESTION :

This problem demonstrates the use of semaphores to coordinate three types of processes.6 Santa Claus sleeps in his shop at the North Pole and can only be wakenedby 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 arehaving their problems solved, any other elves wishing to visit Santa must wait forthose 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 decidedthat the elves can wait until after Christmas, because it is more important to get hissleigh ready. (It is assumed that the reindeer do not want to leave the tropics, andtherefore they stay there until the last possible moment.) The last reindeer to arrivemust get Santa while the others wait in a warming hut before being harnessed to thesleigh. Using synchronization tools like locks, semaphores and monitorsprovide 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)
```

```
                {
                        printf("Santa Claus: preparing sleigh\n");
                        for (int 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);
        }
                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);
}
```

OUTPUT :