1. Program a multi-threaded producer and consumer with locks on concurrent access to a memory location using C/C++/Golang.

Explanation:

Have a global list / array where two or more producers append randomly generated numbers concurrently. A consumer thread must dequeue from the global list and print it!

2. Write a C/C++/Go program to iterate a linux directory tree and count the number of files and number of subdirectories in each directory. Also Count the total number of files and directories in the directory tree. Implement the logic using multi-threading to improve the scanning of the directory.

3. Write a script on any language to get the list of running PID by parsing “top” command

Explanation:

top is a Linux command, which shows all the processes in the system. The problem is to write a script that parses the output of top and provides the PIDs and users in the system . Consider the output of top is given a file input to you.

3. Find and fix the bug in the following code. Write down the explanation about the bug and fix the code.

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

#define BUFFER_SIZE 10

typedef struct node {
    int data;
    struct node *next;
} Node;

Node *head = NULL, *tail = NULL;
int count = 0;

sem_t full, empty;
pthread_mutex_t lock;

void add_data(int data) {
    Node *new_node = (Node*)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = NULL;
    if(tail == NULL) {
        head = tail = new_node;
    } else {
        tail->next = new_node;
        tail = new_node;
    }
    count++;
}
```

```
int remove_data() {
    Node *temp = head;
    int data = temp->data;
    head = head->next;
    count--;
    return data;
}

void *producer(void *arg) {
    int i, data;
    for(i=0; i<100; i++) {
        data = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&lock);
        add_data(data);
        printf("Produced: %d\n", data);
        pthread_mutex_unlock(&lock);
        sem_post(&full);
    }
}

void *consumer(void *arg) {
    int i, data;
    for(i=0; i<100; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&lock);
        data = remove_data();
        printf("Consumed: %d\n", data);
        pthread_mutex_unlock(&lock);
        sem_post(&empty);
    }
}

int main() {
    pthread_t producer_thread, consumer_thread;
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE);
    pthread_mutex_init(&lock, NULL);
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);
    return 0;
}
```