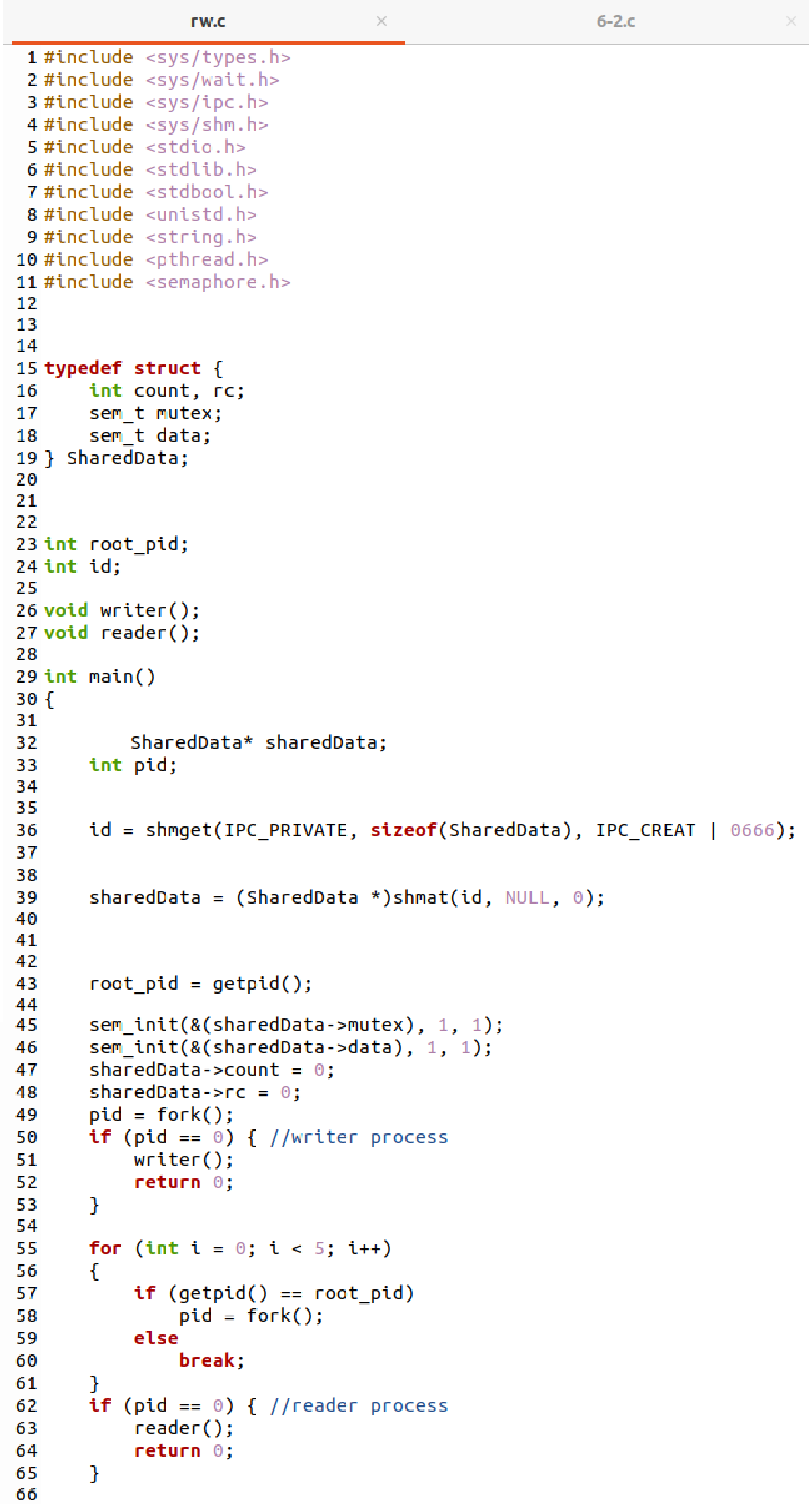
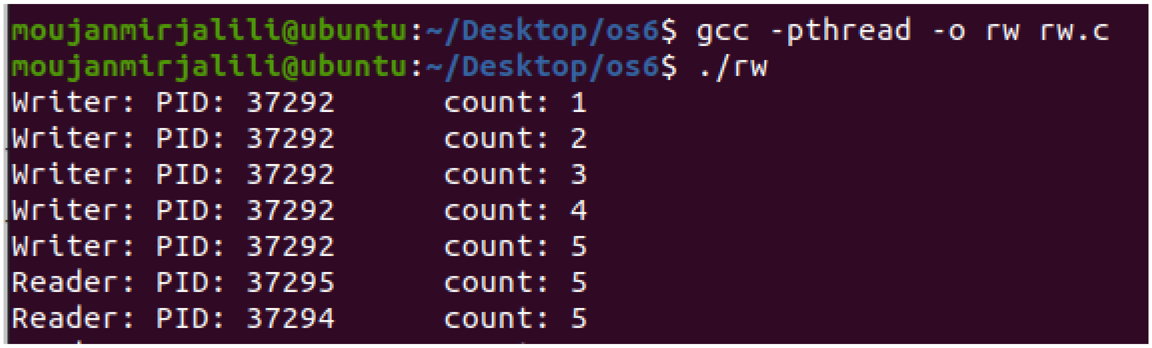
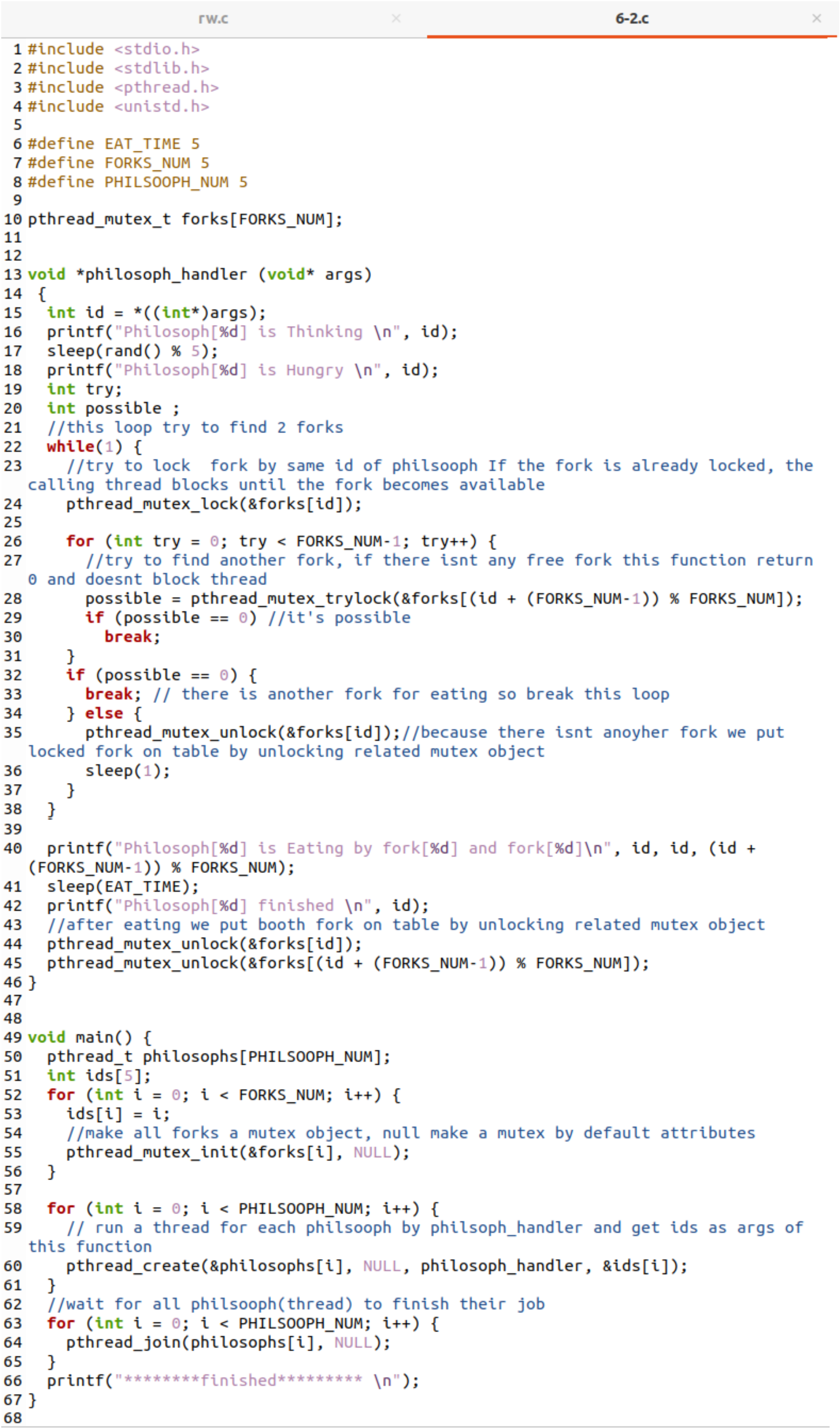


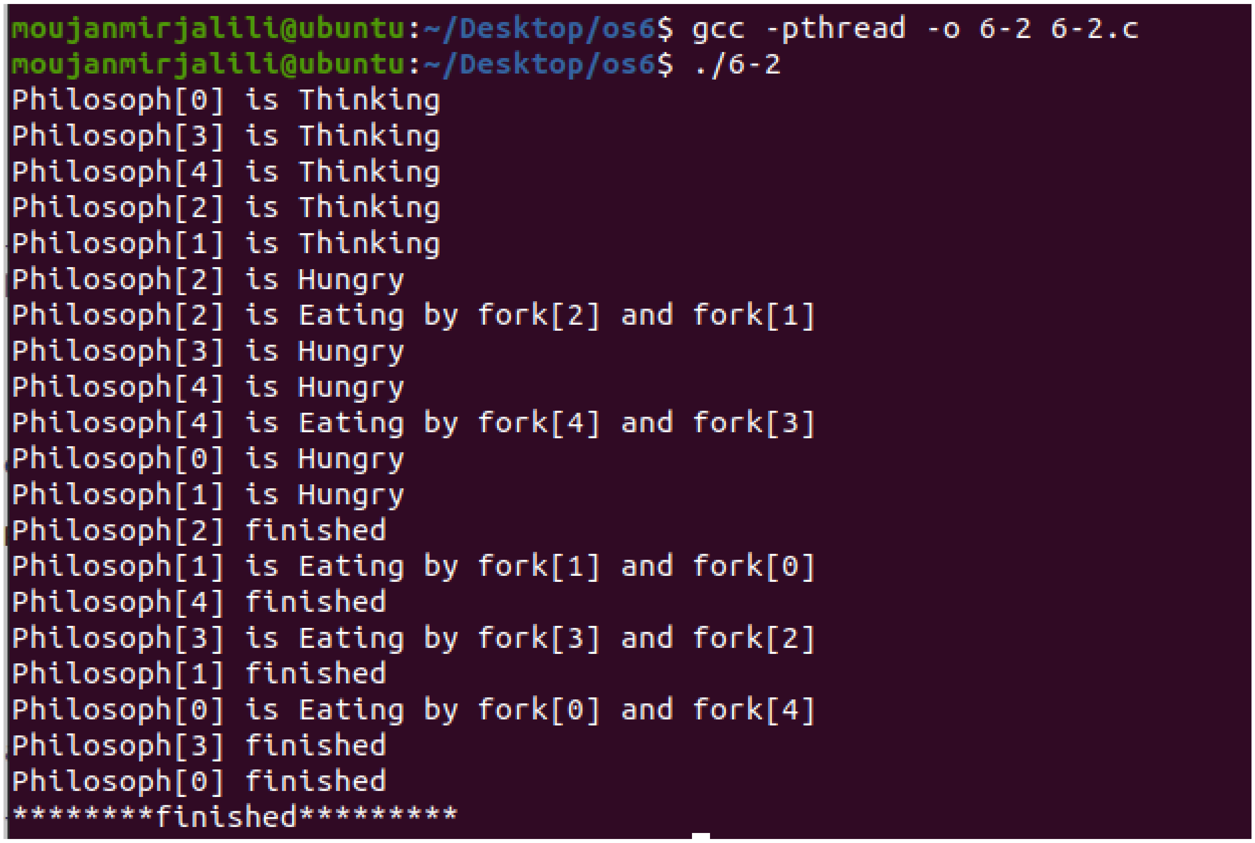
```c
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>



typedef struct {
    int count, rc;
    sem_t mutex;
    sem_t data;
} SharedData;



int root_pid;
int id;

void writer();
void reader();

int main()
{

        SharedData* sharedData;
    int pid;


    id = shmget(IPC_PRIVATE, sizeof(SharedData), IPC_CREAT | 0666);


    sharedData = (SharedData *)shmat(id, NULL, 0);



    root_pid = getpid();

    sem_init(&(sharedData->mutex), 1, 1);
    sem_init(&(sharedData->data), 1, 1);
    sharedData->count = 0;
    sharedData->rc = 0;
    pid = fork();
    if (pid == 0) { //writer process
        writer();
        return 0;
    }

    for (int i = 0; i < 5; i++)
    {
        if (getpid() == root_pid)
            pid = fork();
        else
            break;
    }
    if (pid == 0) { //reader process
        reader();
        return 0;
    }

    if (getpid() == root_pid) // parent process
    {
        wait(NULL); // wait on writer
        for (int i = 0; i < 5; i++) // wait on readers
        {
            wait(NULL);
        }
    }


    return 0;
}

void reader() {

        SharedData* sharedData;
    sharedData = (SharedData *)shmat(id, NULL, 0);

    int pid = getpid();
        bool max = 0;
        while(!max){
                sem_wait((&sharedData->mutex));
                sharedData->rc = sharedData->rc + 1;
                if(sharedData->rc == 1) {
                sem_wait((&sharedData->data));
                }
                sem_post((&sharedData->mutex));
                printf("Reader:\tPID: %d\tcount: %d\n", pid, sharedData->count);
                if(sharedData->count >= 5){
                        max = 1;
                }
                sem_wait((&sharedData->mutex));
                sharedData->rc = sharedData->rc - 1;
                if(sharedData->rc == 0) {
                sem_post((&sharedData->data));
                }
                sem_post((&sharedData->mutex));

        }

}

void writer() {

        SharedData* sharedData;
    sharedData = (SharedData *)shmat(id, NULL, 0);

    int pid = getpid();
        bool max = 0;
        while(!max){
                sem_wait((&sharedData->data));
                sharedData->count++;
                if(sharedData->count >= 5){
                        max = 1;
                }
                printf("Writer:\tPID: %d\tcount: %d\n", pid, sharedData->count);
                sem_post((&sharedData->data));

        }
}
```



```
moujanmirjalili@ubuntu:~/Desktop/os6$ gcc -pthread -o rw rw.c
moujanmirjalili@ubuntu:~/Desktop/os6$ ./rw
Writer: PID: 37292      count: 1
Writer: PID: 37292      count: 2
Writer: PID: 37292      count: 3
Writer: PID: 37292      count: 4
Writer: PID: 37292      count: 5
Reader: PID: 37295      count: 5
Reader: PID: 37294      count: 5
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define EAT_TIME 5
#define FORKS_NUM 5
#define PHILSOOPH_NUM 5

pthread_mutex_t forks[FORKS_NUM];


void *philosoph_handler (void* args)
 {
  int id = *((int*)args);
  printf("Philosoph[%d] is Thinking \n", id);
  sleep(rand() % 5);
  printf("Philosoph[%d] is Hungry \n", id);
  int try;
  int possible ;
  //this loop try to find 2 forks
  while(1) {
    //try to lock  fork by same id of philsooph If the fork is already locked, the calling thread blocks until the fork becomes available
    pthread_mutex_lock(&forks[id]);

    for (int try = 0; try < FORKS_NUM-1; try++) {
      //try to find another fork, if there isnt any free fork this function return 0 and doesnt block thread
      possible = pthread_mutex_trylock(&forks[(id + (FORKS_NUM-1)) % FORKS_NUM]);
      if (possible == 0) //it's possible
        break;
    }
    if (possible == 0) {
      break; // there is another fork for eating so break this loop
    } else {
      pthread_mutex_unlock(&forks[id]);//because there isnt anoyher fork we put locked fork on table by unlocking related mutex object
      sleep(1);
    }
  }

  printf("Philosoph[%d] is Eating by fork[%d] and fork[%d]\n", id, id, (id + (FORKS_NUM-1)) % FORKS_NUM);
  sleep(EAT_TIME);
  printf("Philosoph[%d] finished \n", id);
  //after eating we put booth fork on table by unlocking related mutex object
  pthread_mutex_unlock(&forks[id]);
  pthread_mutex_unlock(&forks[(id + (FORKS_NUM-1)) % FORKS_NUM]);
}


void main() {
  pthread_t philosophs[PHILSOOPH_NUM];
  int ids[5];
  for (int i = 0; i < FORKS_NUM; i++) {
    ids[i] = i;
    //make all forks a mutex object, null make a mutex by default attributes
    pthread_mutex_init(&forks[i], NULL);
  }

  for (int i = 0; i < PHILSOOPH_NUM; i++) {
    // run a thread for each philsooph by philsoph_handler and get ids as args of this function
    pthread_create(&philosophs[i], NULL, philosoph_handler, &ids[i]);
  }
  //wait for all philsooph(thread) to finish their job
  for (int i = 0; i < PHILSOOPH_NUM; i++) {
    pthread_join(philosophs[i], NULL);
  }
  printf("********finished********* \n");
}

```



```
moujanmirjalili@ubuntu:~/Desktop/os6$ gcc -pthread -o 6-2 6-2.c
moujanmirjalili@ubuntu:~/Desktop/os6$ ./6-2
Philosoph[0] is Thinking
Philosoph[3] is Thinking
Philosoph[4] is Thinking
Philosoph[2] is Thinking
Philosoph[1] is Thinking
Philosoph[2] is Hungry
Philosoph[2] is Eating by fork[2] and fork[1]
Philosoph[3] is Hungry
Philosoph[4] is Hungry
Philosoph[4] is Eating by fork[4] and fork[3]
Philosoph[0] is Hungry
Philosoph[1] is Hungry
Philosoph[2] finished
Philosoph[1] is Eating by fork[1] and fork[0]
Philosoph[4] finished
Philosoph[3] is Eating by fork[3] and fork[2]
Philosoph[1] finished
Philosoph[0] is Eating by fork[0] and fork[4]
Philosoph[3] finished
Philosoph[0] finished
********finished*********
```

سوال

بله. اگر در پیاده سازی دقت نشود و الگوریتم مناسبی به کار گرفته نشود، ممکن است حالتی پیش بیاید که هر فلیسوف تنها یک چوب دارد و منتظر چوب دوم است؛ در این وضعیت نه کسی میتواند غذا بخورد و نه کسی چوبش را رها میکند تا دیگری غذا بخورد و به بنبست میرسیم.