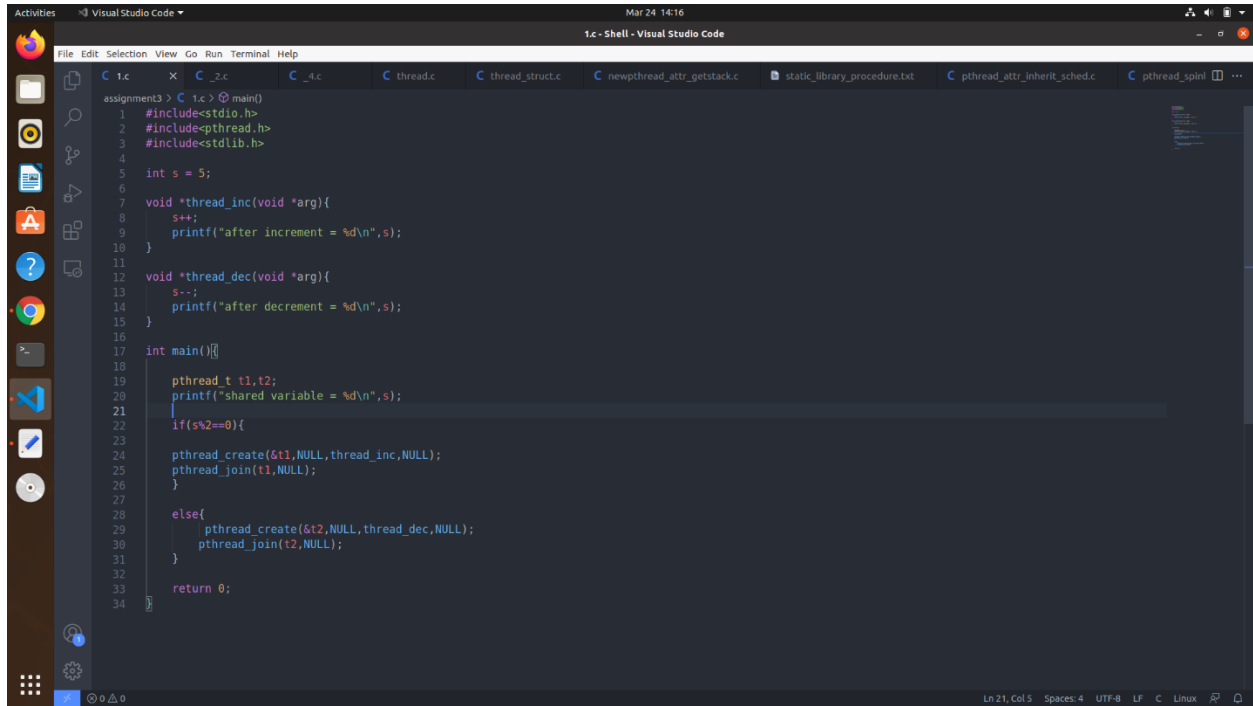
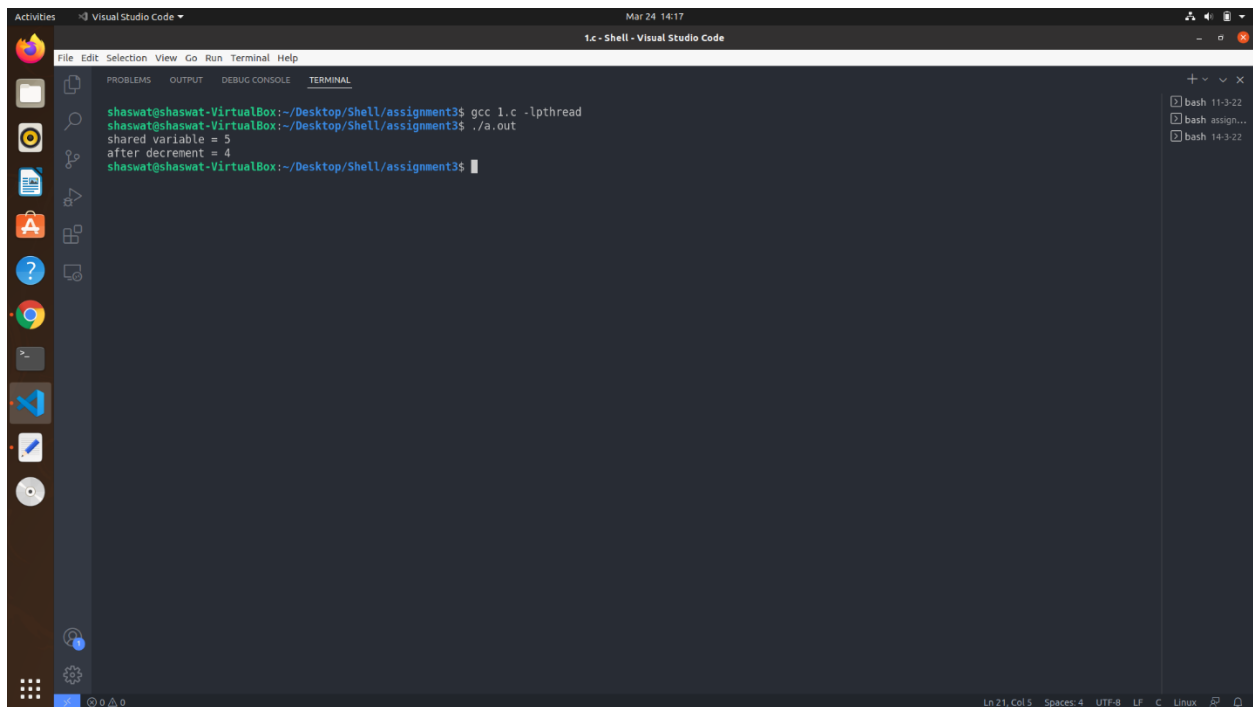


1. Write a multithreading program, where threads runs same shared global variable of the process between them.



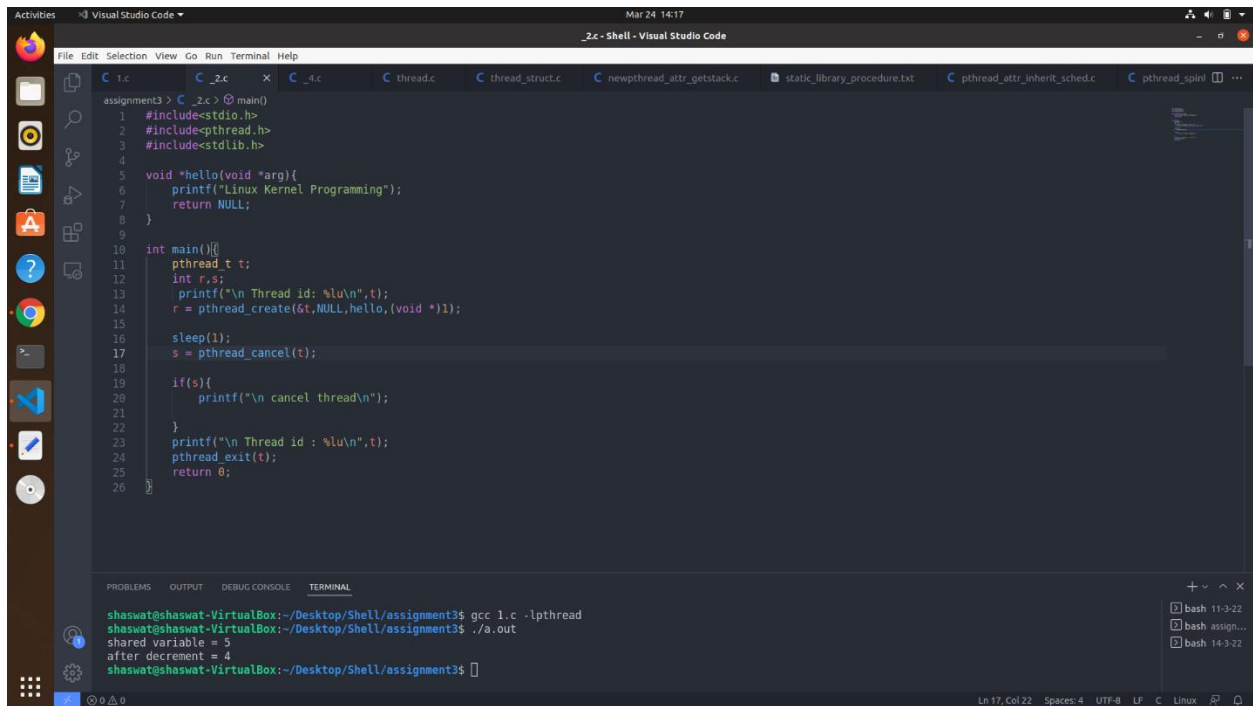
```
1  #include<stdio.h>
2  #include<pthread.h>
3  #include<stdlib.h>
4
5  int s = 5;
6
7  void *thread_inc(void *arg){
8      s++;
9      printf("after increment = %d\n",s);
10 }
11
12 void *thread_dec(void *arg){
13     s--;
14     printf("after decrement = %d\n",s);
15 }
16
17 int main(){
18
19     pthread_t t1,t2;
20     printf("shared variable = %d\n",s);
21     if(s%2==0){
22
23         pthread_create(&t1,NULL,thread_inc,NULL);
24         pthread_join(t1,NULL);
25     }
26
27     else{
28         pthread_create(&t2,NULL,thread_dec,NULL);
29         pthread_join(t2,NULL);
30     }
31
32     return 0;
33 }
34
```

Output:-



```
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ gcc 1.c -lpthread
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
shared variable = 5
after decrement = 4
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$
```

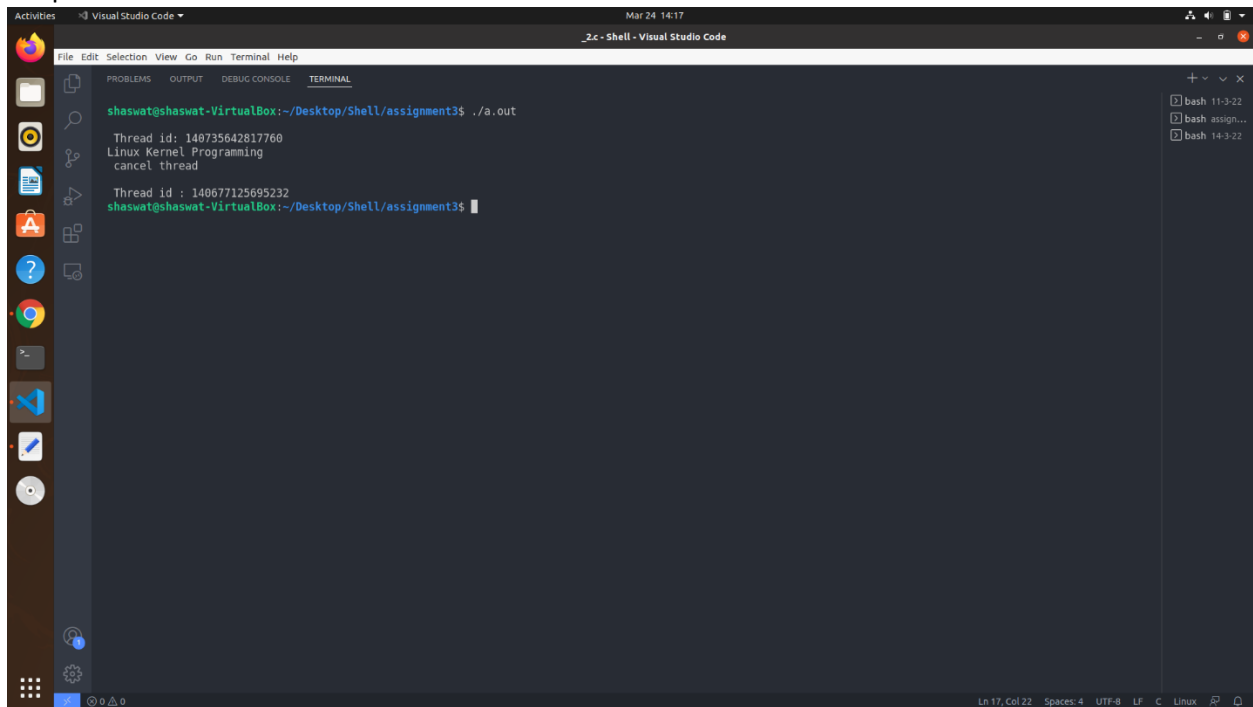
2. Write a program where thread cancel itself.(use pthread_cancel())



```
assignment3 > C _2.c > @ main()
1 #include<stdio.h>
2 #include<pthread.h>
3 #include<stdlib.h>
4
5 void *hello(void *arg){
6     printf("Linux Kernel Programming");
7     return NULL;
8 }
9
10 int main(){
11     pthread_t t;
12     int r,s;
13     printf("\n Thread id: %lu\n",t);
14     r = pthread_create(&t,NULL,hello,(void *)1);
15
16     sleep(1);
17     s = pthread_cancel(t);
18
19     if(s){
20         printf("\n cancel thread\n");
21     }
22     printf("\n Thread id : %lu\n",t);
23     pthread_exit(t);
24     return 0;
25 }
26
```

```
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ gcc 1.c -lpthread
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
shared variable = 5
after decrement = 4
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$
```

Output:-



```
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
Thread id: 140735642817760
Linux Kernel Programming
cancel thread

Thread id : 140677125695232
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$
```

3. Write a program that changes the default properties of newly created posix threads.(ex: to change default pthread stack size)

```
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>

void* proc(void* param){
    sleep(2);
    return 0;
}

int main(){
    pthread_attr_t Attr;
    pthread_t id;
    void *stk;
    size_t siz;
    int err;
    size_t my_stksize= 0x3000000;
    void * my_stack;
    struct sched_param p;

    pthread_attr_init(&Attr);
    int *sc;
    pthread_attr_getstacksize(&Attr,&siz);
    pthread_attr_getstacksize(&Attr,&stk);
    pthread_attr_getschedpolicy(&Attr,&sc);
    pthread_attr_getschedparam(&Attr,&p);
    printf("Default: Addr =%08x\tdefault Size = %d\tScheduling=
%d\tPriority = %d\n",stk,siz,sc,p.sched_priority);

    my_stack = (void*)malloc(my_stksize);
    sc=1;
    p.sched_priority=18;

    pthread_attr_setstack(&Attr,my_stack,my_stksize);
    pthread_create(&id,&Attr,proc,NULL);
```

```

pthread_attr_setschedpolicy(&Attr,sc);
pthread_attr_setschedparam(&Attr,&p);

pthread_attr_getstack(&Attr,&stk,&siz);

pthread_attr_getschedparam(&Attr,&p);
pthread_attr_getschedpolicy(&Attr,&sc);
printf("new defined stack: Addr = %08x and Size = %d\tScheduling =
%ld\tPriority = %ld\n",stk,siz,sc,p.sched_priority);
}

```

Output:-

```

shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
Default: Addr =00000000 default Size = 8388608 Scheduling= 0 Priority = 0
new defined stack: Addr = 77e4c010 and Size = 50331648 Scheduling = 1 Priority = 18
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$

```

4. Write a program where pthread task displays the thread id and also prints the calling process pid.

The screenshot shows the Visual Studio Code editor with a C program named `assignment3.c`. The program includes `stdio.h`, `pthread.h`, and `stdlib.h`. It defines a function `hello` that prints the process ID and returns `NULL`. The `main` function creates a thread `t` using `pthread_create` and then calls `pthread_exit`. The terminal output shows the execution of `./a.out`, displaying the thread ID and process ID.

```
1 #include<stdio.h>
2 #include<pthread.h>
3 #include<stdlib.h>
4
5 void *hello(void *arg){
6     printf("Process id is %d\n",getpid());
7     return NULL;
8 }
9
10 int main(){
11     pthread_t t;
12     int r,s;
13     printf("\nThread id: %lu\n",t);
14     r = pthread_create(&t,NULL,hello,(void *)1);
15     pthread_exit(t);
16 }
17
18
```

Terminal Output:

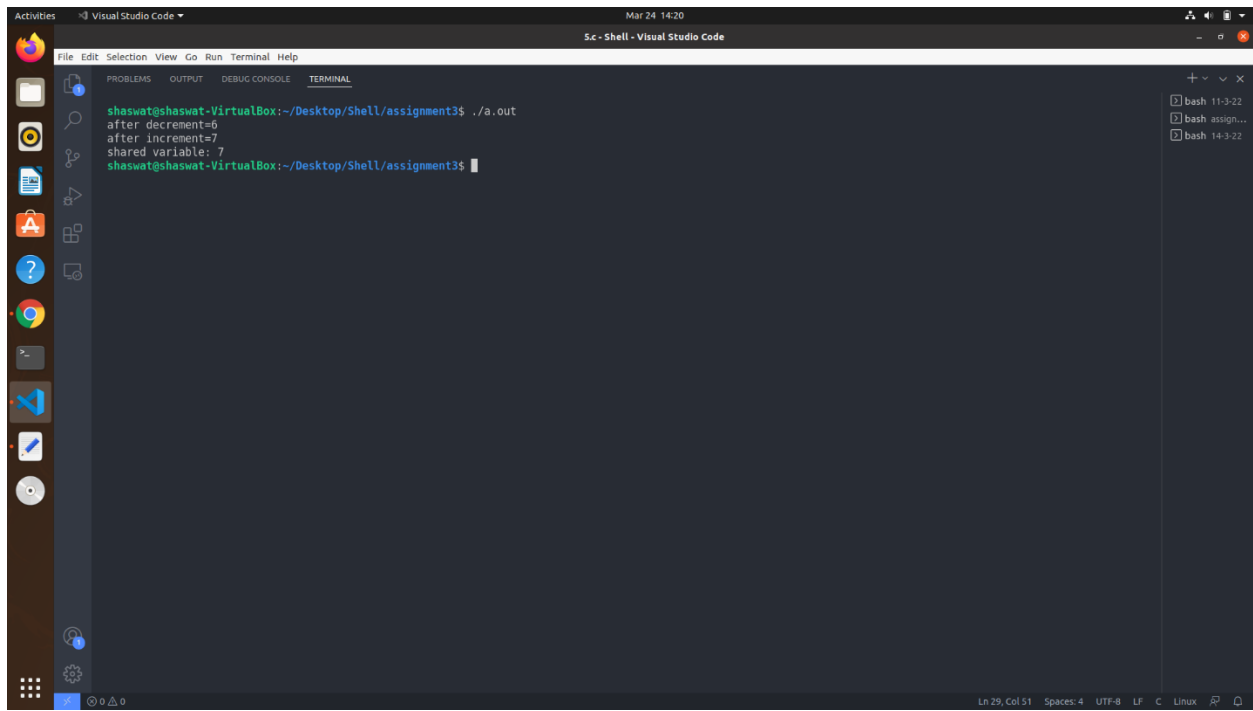
```
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
Thread id: 140723833829408
Process id is 38562
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$
```

5. Write a program that implements threads synchronization using mutex techniques.

The screenshot shows the Visual Studio Code editor with a C program named `assignment3.c`. The program includes `pthread.h` and `stdio.h`. It defines a shared variable `s` and a mutex `mut`. Two threads, `thread1_inc` and `thread2_dec`, are created. `thread1_inc` increments `s` and `thread2_dec` decrements `s`, both using the mutex for synchronization. The `main` function initializes the mutex, creates the threads, and joins them before printing the final value of `s`.

```
1 // 5. Write a program that implements threads synchronization using mutex techniques. we can perform threading operations using posix
2 #include<pthread.h>
3 #include<stdlib.h>
4 #include<stdio.h>
5
6
7 int s = 7;
8 pthread_mutex_t mut;
9
10 void * thread1_inc(void *arg){
11     pthread_mutex_lock(&mut);
12     s++;
13     pthread_mutex_unlock(&mut);
14     printf("after increment=%d\n", s);
15 }
16
17 void * thread2_dec(void *arg){
18     pthread_mutex_lock(&mut);
19     s--;
20     pthread_mutex_unlock(&mut);
21     printf("after decrement=%d\n", s);
22 }
23
24 int main(){
25     pthread_t tid, tid1;
26     pthread_mutex_init(&mut, NULL);
27
28     pthread_create(&tid, NULL, thread1_inc, NULL);
29     pthread_create(&tid1, NULL, thread2_dec, NULL);
30
31     pthread_join(tid, NULL);
32     pthread_join(tid1, NULL);
33     printf("shared variable: %d\n", s);
34     return 0;
35 }
36
```

Output:-



The screenshot shows a Visual Studio Code window with a terminal open. The terminal output is as follows:

```
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$ ./a.out
after decrement=6
after increment=7
shared variable: 7
shaswat@shaswat-VirtualBox:~/Desktop/Shell/assignment3$
```

The terminal window has a title bar that reads "s.c - Shell - Visual Studio Code". The status bar at the bottom indicates "Ln 29, Col 51", "Spaces: 4", "UTF-8", "LF", "C", "Linux", and a file icon.