Assignment 8:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<pthread.h>
int sum(int arg1);
int sum1=0;
int main()
{
int res2;
pthread_t tid2;
void *result2;
printf("\nThread ID=%u",(unsigned int)pthread_self());
printf("\nThe process ID=%d",getpid());
printf("\nThe parent process ID=%d",getppid());
res2=pthread_create(&tid2,NULL,(void *)sum,(int *)sum1);
if(res2!=0)
{
perror("Thread joined fail");
exit(EXIT_FAILURE);
}
Sleep(15);
printf(“\n The sum of two number is=%d”,sum1);
printf("\n waiting for thread to finish.........\n");
res2=pthread_join(tid2,&result2);
if(res2!=0)
{
perror("Thread joined failed");
exit(EXIT_FAILURE);
}
printf("\nThread2 joined, it returned %s",(char *)result2);
printf(" %d\n",sum1);
printf("Thread id =%d",tid2);
printf("\nThread ID=%u",(unsigned int)pthread_self());
system("ps -al");
exit(EXIT_SUCCESS);
}
int sum(int arg1)
{
int i,j;
printf ("Enter the value of 1st number");
scanf("%d",&i);
printf ("Enter the value of 1st number");
scanf("%d",&j);
sum1=i+j;
```

```
printf("\nThread ID=%u",(unsigned int)pthread_self());
printf("\nThe process ID=%d",getpid());
printf("\nParent process ID=%d",getppid());
pthread_exit("The addition of two number is:");
}
```

There is a whole set of library calls associated with threads, most of whose names start with pthread. To use these library calls, you must define the macro _REENTRANT, include the file pthread.h, and link with the threads library using -lpthread.
Write the above program in Vi editor with name thread01.c by giving the command in CLI prompt
$ vi thread01.c
Including the file pthread.h provides you with other definitions and prototypes that you will need in your code, much like stdio.h for standard input and output routines. Finally, you need to ensure that you include the appropriate thread header file and link with the appropriate threads library that implements the pthread functions. The Try It Out example later in this section offers more detail about compiling your program, but first let's look at the new functions you need for managing threads. pthread_create creates a new thread, much as fork creates a new process.

After writing the program save it and If you want to run this program you have to give the following command in CLI prompt:
$gcc –D_REENTRANT thread01.c –o thread01 –lpthread

int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);

This pthread_create creates a new thread and this system call is defined in **#include <pthread.h>**
The first argument is a pointer to pthread_t. When a thread is created, an identifier is written to the memory location to which this variable points. This identifier enables you to refer to the thread. The next argument sets the thread attributes. You do not usually need any special attributes, and you can simply pass NULL as this argument. Later in the chapter you will see how to use these attributes. The final two arguments tell the thread the function that it is to start executing and the arguments that are to be passed to this function.
In our program:
We have define the global variable as
int sum1=0;
and the thread function prototype
```
int sum(int arg1);
```
In main, you declare some variables and then call pthread_create to start running your new thread.

int res2;
pthread_t tid2;
void *result2;

```
printf("\nThread ID=%u",(unsigned int)pthread_self());
printf("\nThe process ID=%d",getpid());
printf("\nThe parent process ID=%d",getppid());
```

1st printf will print the thread Id of main program.
2nd printf will print the process Id of the main process.
3rd printf will print the bash process ID on which main thread (or process) is running.

res2=pthread_create(&tid2,NULL,sum,(int)sum1);

You pass the address of a pthread_t object as **tid2** that you can use to refer to the thread afterward. You don't want to modify the default thread attributes, so you pass NULL as the second parameter. The final two parameters are the function i.e **sum** to call and a parameter **sum1** to pass to it.
If the call succeeds, two threads will now be running. The original thread (main) continues and executes the code after pthread_create, and a new thread starts executing in the imaginatively named and declared as **`int sum(int arg1)`**

On successful execution of the pthread_create( ) will return 0.So the next if( ) statement will not be executed for successful creation of thread.

```
Sleep(15);
printf("\n The sum of two number is=%d",sum1);
printf("\n waiting for thread to finish.....\n");
```

Next the main thread will be in sleep mode for 15 unit of time by calling sleep() system call and the system CPU will switch to execute the called thread **int sum(int arg1)** and it will be executed. After 15 unit of time (if CPU is free and allocated to main thread) main thread will be executed 1st printf will print the initial vale of sum1 i.e 0 (provided within that 15 unit of the called thread has not completed its task, otherwise it give the value of present summation value for example like summation of input value 20 and 40 i.e 60) then the 2nd printf will print the string.

Then calls pthread_join.
res2=pthread_join(tid2,&result2);
Here you pass the identifier of the thread i.e tid2 that you are waiting to join and a pointer to a result2. This function will wait until the other thread terminates before it returns. It then prints the return value from the thread and the contents of a variable, and exits.
If the pthread_join successfully executed it will return 0 and that's why the If( ) statement will not execute and main thread will execute the following statement.

```
printf("\nThread2 joined, it returned %s",(char *)result2);
printf(" %d\n",sum1);
printf("Thread id =%d",tid2);
printf("\nThread ID=%u",(unsigned int)pthread_self());
```

The 1st statement will print the string along with the content of result2 which stored during the pthread_exit( ) system call in thread function.
2nd printf will print the value of sum1
3rd printf will print the thread identifier i.e thread ID of the called thread (main process) because this value is still stored in that identifier.
4th printf will print the thread id of the main thread(main process) because the called thread already exit.

```
system("ps -al");
exit(EXIT_SUCCESS);
```

The next above statement will show the existing process running in the system by "ps- al" command and exit by exit() system call.

Simultaneously the CPU will also run the thread the function **int sum(int arg1)**
And it will ask for input value do the summation and store the sum in global variable sum1.

Assignment 9:

Write c program to show two thread are running simultaneously with separate thread ID under single process and check pid of the process and thread ID of the two running thread.

Support material:

Define a global variable with initial value 1 and when the main function is executing and to 2 when your new thread is executing.
In the main function, after the creation of the new thread, add the following code:

```
int print_count1 = 0; //local to main thread
while(print_count1++ < 20) {
if (run_now == 1) {
printf("1");
run_now = 2; // run_now is the global variable accessible to both thread
}
else {
sleep(1);
}
}
```

If run_now is 1, print "1" and set it to 2. Otherwise, you sleep briefly and check the value again. You are waiting for the value to change to 1 by checking over and over again. This is called a *busy wait*, although here it is slowed down by sleeping for a second between checks.
In thread_function, where your new thread is executing, you do much the same but with the values reversed:

```
int print_count2 = 0; // local to thread funtion
while(print_count2++ < 20) {
if (run_now == 2) {
printf("2");
run_now = 1; // run_now is the global variable accessible to both thread
}
else {
sleep(1);
}
}
```

You may find that it takes a few seconds for the program to produce output, particularly on a single-core CPU machine.
I need the output should be like this: