**Operating Systems**

**LAB 7**

**Spring 2022**

**Operating Systems**

**Submitted by: Maaz Habib, Alishba Orakzai, Nida Asmat Burki**

**Registration no. 20PWCSE1952, 20pwcse1953, 20pwcse1955**

**Class Section: C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

**Submitted to:**

SIR IBAD

**June 7, 2022**

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**Threads Creation and Execution:**

**Objective:**

This lab examines aspects of threads and multiprocessing (and multithreading). The

primary objective of this lab is to implement the Thread Management.

Functions:

4

.

Creating Threads

5

.

Terminating Thread Execution

6

.

Passing Arguments To Threads

7

.

Thread Identifiers

8

.

Joining Threads

**What is thread?**

A thread is a semi-process, that has its own stack, and executes a given piece of code. Unlike a real process, the thread normally shares its memory with other threads (where as for processes we usually have a different memory area for each one of them). A Thread Group is a set of threads all executing inside the same process. They all share the same memory, and thus can access the same global variables, same heap memory, same set of file descriptors, etc. All these threads execute in parallel (i.e. using time slices, or if the system has several processors, then really in parallel).

**What are pthreads?**

Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications. In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. Pthreads are defined as a set of C language programming types and procedure calls. Vendors usually provide a Pthreads implementation in the form of a header/include file and a library which you link with your program.

**Why pthreads?**

The primary motivation for using Pthreads is to realize potential program performance gains.  When compared to the cost of creating and managing a process, a thread can be created with much less operating system overhead. Managing threads requires fewer system resources than managing processes. All threads within a process share the same address space. Inter-thread communication is more efficient and, in many cases, easier to use than inter-process communication.  Threaded applications offer potential performance gains and practical advantages over non-threaded applications in several other ways: Overlapping CPU work with I/O: For example, a program may have sections where it is performing a long I/O operation. While one thread is waiting for an I/O system call to complete, CPU intensive work can be performed by other threads. Priority/real-time scheduling: tasks which are more important can be scheduled to supersede or interrupt lower priority tasks. Asynchronous event handling: tasks which service events of indeterminate frequency and duration can be interleaved. For example, a web server can both transfer data from previous requests and manage the arrival of new requests. Multi-threaded applications will work on a uniprocessor system, yet naturally take advantage of multiprocessor system, without recompiling.  In a multiprocessor environment, the most important reason for using Pthreads is to take advantage of potential parallelism. This will be the focus of the remainder of this session.

**The pthreads API:**

The subroutines which comprise the Pthreads API can be informally grouped into three;

**Major classes:**

Thread management: The first class of functions work directly on threads - creating, detaching, joining, etc. They include functions to set/query thread attributes (joinable, scheduling etc.)

**Mutexes:**

The second class of functions deal with a coarse type of synchronization, called a &quot; mutex&quot;, which is an abbreviation for &quot;mutual exclusion&quot;. Mutex functions provide for creating, destroying, locking and unlocking mutexes. They are also supplemented by mutex attribute functions that set or modify attributes associated with mutexes.

**Condition variables:**

The third class of functions deal with a finer type of synchronization - based upon programmer specified conditions. This class includes functions to create, destroy, wait and signal based upon specified variable values.

Functions to set/query condition variable attributes are also included.

**Naming conventions:**

All identifiers in the threads library begin with pthread\_

pthread\_  Threads themselves and miscellaneous subroutines

pthread\_attr  Thread attributes objects

pthread\_mutex  Mutexes

pthread\_mutexattr  Mutex attributes objects.

pthread\_cond  Condition variables

pthread\_condattr Condition attributes objects

pthread\_key  Thread-specific data keys

**Thread Management Functions:**

The function pthread\_create is used to create a new thread, and the function

pthread\_exit is used by a thread to terminate itself. The function pthread\_join is used

by a thread to wait for termination of another thread.

**Function:**

int pthread\_create (pthread\_t   \*threadhandle, /\*Thread handle

returned by reference \*/ pthread\_attr\_t   \*attribute, /\* Special Attribute

for starting thread, may be NULL \*/ void  \*(\*start\_routine)(void \*), /\*

Main Function which thread executes \*/ void  \*arg /\* An extra

argument passed as a pointer \*/ );

**Info:**

Request the PThread library for creation of a new thread. The return

value is 0 on success. The pthread\_t is an abstract datatype that is used

as a handle to reference the thread.

Function: Void pthread\_exit (void \*retval /\* return value passed as a pointer \*/);

**Info:**

This Function is used by a thread to terminate. The return value is

passed as a pointer. This pointer value can be anything so long as it

does not exceed the size of (void \*). Be careful, this is system

dependent. You may wish to return an address of a structure, if the

returned data is very large.

Function: Int pthread\_join (pthread\_t    threadhandle, /\* Pass threadhandle \*/

void   \*\*returnvalue /\* Return value is returned by ref. \*/);

**Info:**

Return 0 on success, and negative on failure. The returned value is a

pointer returned by reference. If you do not care about the return

value, you can pass NULL for the second argument.

**Thread Initialization:**

Include the pthread.h library :

#include &lt;pthread.h&gt;

Declare a variable of type pthread\_t :

pthread\_t    the\_thread

When you compile, add -lpthread to the linker flags :

gcc   threads.c   -o   threads  -lpthread

Initially, threads are created from within a process. Once created, threads are peers,

and may create other threads. Note that an &quot;initial thread&quot; exists by default and is

the thread which runs main( ).

+

**Thread Identifiers:**

pthread\_self ( )

Returns the unique thread ID of the calling thread. The returned data object is opaque can

not be easily inspected.

pthread\_equal ( thread1, thread2 )

**Compares two thread IDs:**

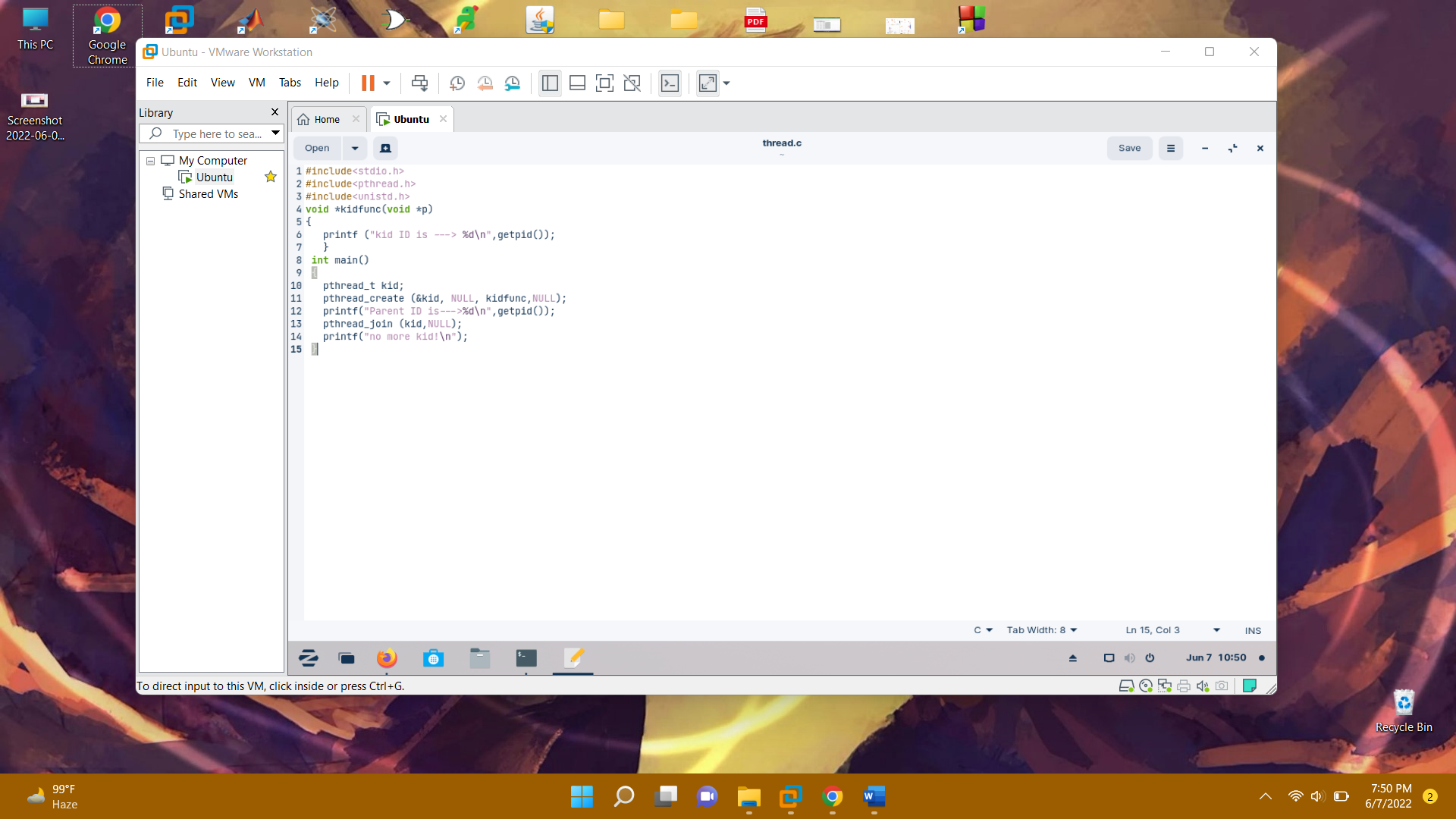
If the two IDs are different 0 is returned, otherwise a non-zero value is returned.

Because thread IDs are opaque objects, the C language equivalence operator == should

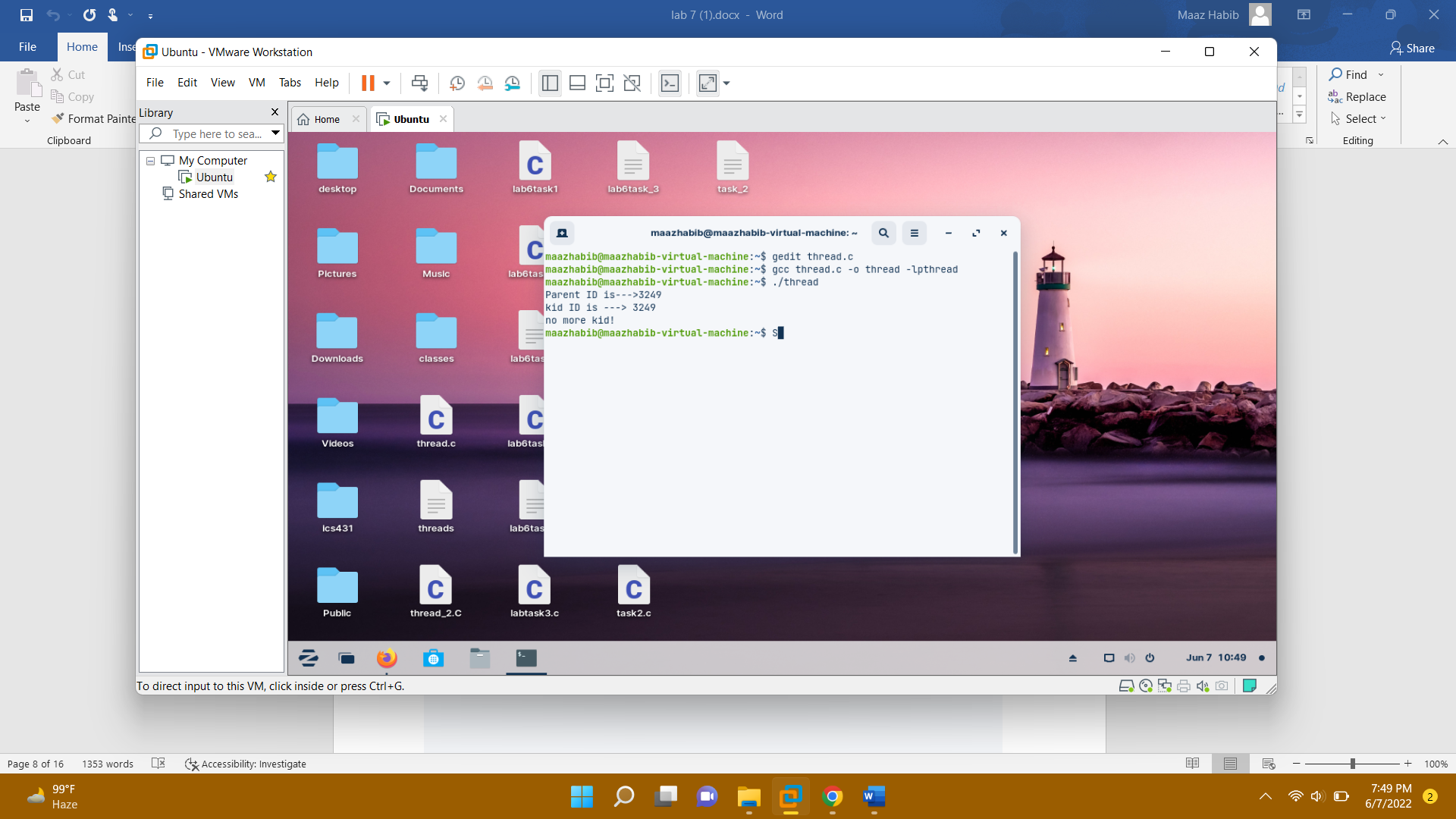
not be used to compare two thread IDs.

**Example1:**

**Code:**

****

**Output:**

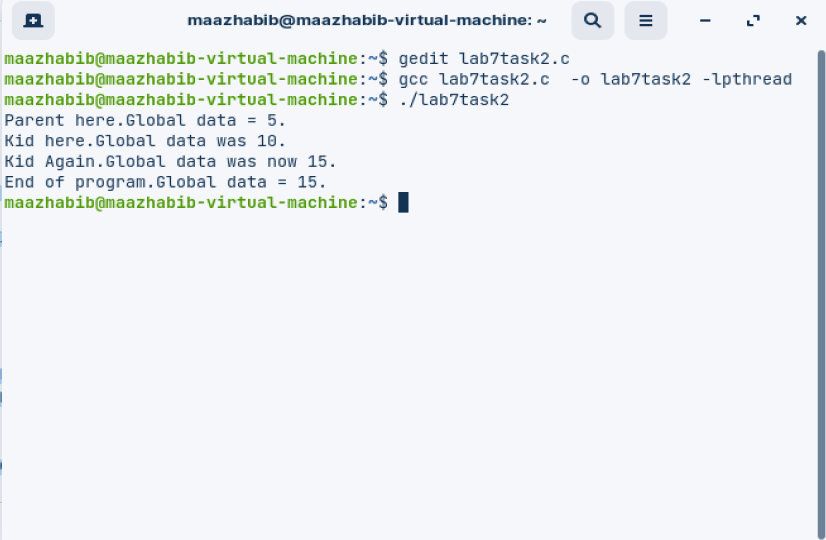
****

**Example2:**

**Code:**

****

**Output:**

****

**Example3:**

**Code:**

****

**Output:**

****

**Example4:**

**Code:**

****

****

**Output:**

****

**Task1:**

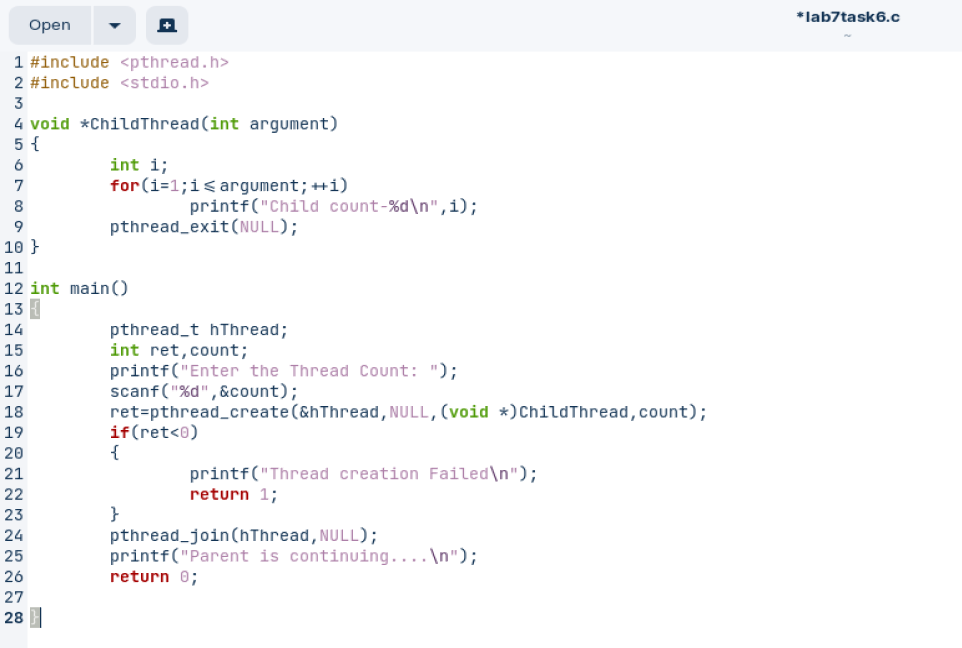
**Code:**

****

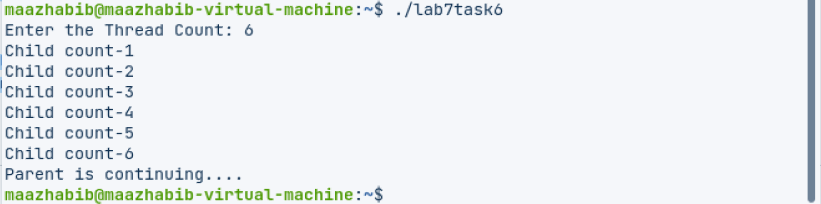
**Output:**

**Task2:**

**Code:**

****

**Output:**

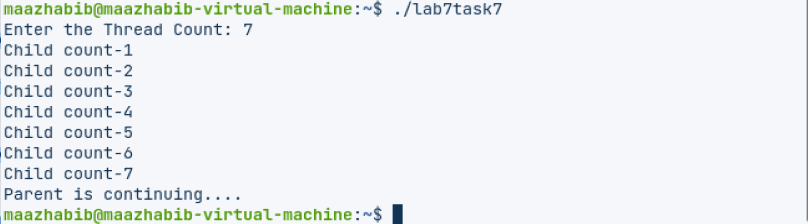
****

**Task3:**

**Code:**

****

**Output:**

****