**Question 1**

**Mutex**

* Mutex is a mutual exclusion lock. If there is a shared resource between different threads, mutex enforces exclusive access to that variable to only one thread at one time, hence protecting the **critical section** of the memory.
* Mutex doesn’t work between processes and only is applied for threads.
* Only one thread at a time can take the ownership of the mutex. Once it has the ownership, no other thread can access the shared resources or the critical section until that mutex is done with its job. Once that mutex has performed its job, it releases the mutex lock, making it possible for others to take ownership of the mutex.

**Working of the Mutex**

* To begin with, there is an integer in memory that provides the locked state. It can either be one or zero.
* Initially, the locked state is set to zero. As soon as one thread attempts to gain the access of a critical section, it gains the ownership of the mutex and locked state of mutex is turned to one.
* If now a content switch is performed by the scheduler, all other threads that are ready to execute the critical section are awaken. However, since the region is already locked by mutex (because of another thread having the ownership of mutex), all the other threads are denied permission to enter critical section and are put to sleep again.
* Content switch takes place again but again as long as the mutex is locked, no thread is allowed execution.
* As soon as the first thread finishes executing the critical section, it unlocks the mutex, turning the locked state to zero from one. Now, content switch occurs and the first thread to request permission for critical section is given entry into the critical section, with others again placed into sleep due to locked state turning into one.

**Example**

* Question number 5 is a very good example of Mutex.
* Different threads at the same time try to gain access to the shared resource of “bank balance” integer, however only one is given access to it at one time. Once one thread has entered the critical section, locked state turns to one and other threads are put to sleep.
* Now, once the first thread has finished depositing its value into the bank balance, it unlocks the mutex, turning the locked state value to 0.
* Content switch occurs and other threads are awakened. The first thread to ask permission to critical section then gains the ownership of the mutex and the remaining are put to sleep.
* This process repeats until all the threads have finished their execution.
* Implementation of the code is also given below,

**Implementation**

#include <iostream>

#include <pthread.h>

#include <string>

#include <unistd.h>

using namespace std;

pthread\_mutex\_t m = PTHREAD\_MUTEX\_INITIALIZER;

int bank\_balance = 100; // initially bank balance is set to 100

void \*deposit\_amount(void \*arg) {

int deposited\_amount = \*(int \*)arg;

pthread\_mutex\_lock(&m);

int old\_balance = bank\_balance;

int new\_balance = old\_balance + deposited\_amount;

bank\_balance = new\_balance;

pthread\_mutex\_unlock(&m);

pthread\_exit(NULL);

}

int main() {

pthread\_t brother, dad, cousin;

int bro, d, co;

cout << "Enter amount deposited by Dad: ";

cin >> d;

cout << "Enter amount deposited by Brother: ";

cin >> bro;

cout << "Enter amount deposited by Cousin: ";

cin >> co;

pthread\_create(&dad, NULL, deposit\_amount, &d);

pthread\_create(&brother, NULL, deposit\_amount, &bro);

pthread\_create(&cousin, NULL, deposit\_amount, &co);

pthread\_join(dad, NULL);

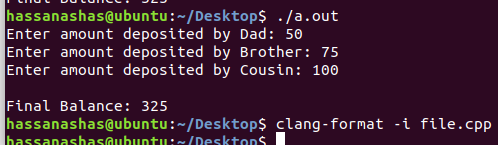
pthread\_join(brother, NULL);

pthread\_join(cousin, NULL);

cout << endl << "Final Balance: " << bank\_balance << endl;

return 0;

}



**Question 2**

#include <iostream>

#include <pthread.h>

#include <semaphore.h>

#include <string>

#include <unistd.h>

using namespace std;

string message;

sem\_t s;

void \*send\_message(void \*arg) {

cout << "Enter Message to Send: ";

getline(cin, message);

cout << "Message Sent to Receiver: " << message << endl;

sem\_post(&s);

pthread\_exit(NULL);

}

void \*receive\_message(void \*arg) {

sem\_wait(&s);

cout << "Message Recieved from Sender: " << message << endl;

pthread\_exit(NULL);

}

int main() {

pthread\_t sender, receiver;

sem\_init(&s, 0, 0);

pthread\_create(&sender, NULL, send\_message, NULL);

pthread\_create(&receiver, NULL, receive\_message, NULL);

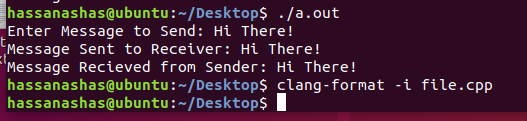
pthread\_join(sender, NULL);

pthread\_join(receiver, NULL);

sem\_destroy(&s);

return 0;

}



**Question 3**

#include <iostream>

#include <pthread.h>

#include <string>

#include <unistd.h>

using namespace std;

string message;

pthread\_cond\_t c = PTHREAD\_COND\_INITIALIZER;

pthread\_mutex\_t m = PTHREAD\_MUTEX\_INITIALIZER;

int done = 0;

void \*send\_message(void \*arg) {

pthread\_mutex\_lock(&m);

cout << "Enter Message to Send: ";

getline(cin, message);

cout << "Message Sent to Receiver: " << message << endl;

done = 1;

pthread\_cond\_signal(&c);

pthread\_mutex\_unlock(&m);

pthread\_exit(NULL);

}

void \*receive\_message(void \*arg) {

pthread\_mutex\_lock(&m);

while (done == 0)

pthread\_cond\_wait(&c, &m);

cout << "Message Recieved from Sender: " << message << endl;

pthread\_mutex\_unlock(&m);

pthread\_exit(NULL);

}

int main() {

pthread\_t sender, receiver;

pthread\_create(&sender, NULL, send\_message, NULL);

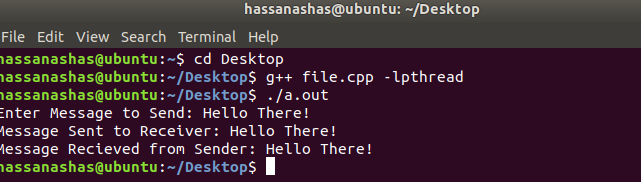
pthread\_create(&receiver, NULL, receive\_message, NULL);

pthread\_join(sender, NULL);

pthread\_join(receiver, NULL);

return 0;

}



**Question 4**

#include <iostream>

#include <pthread.h>

using namespace std;

int var;

int result;

void \*fun(void \*arg)

{

pthread\_detach(pthread\_self());

var++;

result = result + var;

pthread\_exit(NULL);

return NULL;

}

int main()

{

var = 0;

result = 0;

pthread\_t p1;

pthread\_create (&p1, NULL, &fun, NULL);

pthread\_join(p1, NULL);

pthread\_create (&p1, NULL, &fun, NULL);

pthread\_join(p1, NULL);

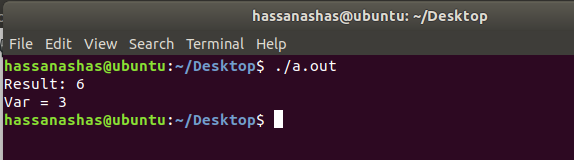
pthread\_create (&p1, NULL, &fun, NULL);

pthread\_join(p1, NULL);

cout << "Result: " << result << endl << "Var = " << var << endl;

return 0;

}



**Question 5**

#include <iostream>

#include <pthread.h>

#include <string>

#include <unistd.h>

using namespace std;

pthread\_mutex\_t m = PTHREAD\_MUTEX\_INITIALIZER;

int bank\_balance = 100; // initially bank balance is set to 100

void \*deposit\_amount(void \*arg) {

int deposited\_amount = \*(int \*)arg;

pthread\_mutex\_lock(&m);

int old\_balance = bank\_balance;

int new\_balance = old\_balance + deposited\_amount;

bank\_balance = new\_balance;

pthread\_mutex\_unlock(&m);

pthread\_exit(NULL);

}

int main() {

pthread\_t brother, dad, cousin;

int bro, d, co;

cout << "Enter amount deposited by Dad: ";

cin >> d;

cout << "Enter amount deposited by Brother: ";

cin >> bro;

cout << "Enter amount deposited by Cousin: ";

cin >> co;

pthread\_create(&dad, NULL, deposit\_amount, &d);

pthread\_create(&brother, NULL, deposit\_amount, &bro);

pthread\_create(&cousin, NULL, deposit\_amount, &co);

pthread\_join(dad, NULL);

pthread\_join(brother, NULL);

pthread\_join(cousin, NULL);

cout << endl << "Final Balance: " << bank\_balance << endl;

return 0;

}

