HW1 Linda with OpenMP

Overview

In this project, we are going to implement Linda with three operations using multithread by OpenMP.

Tutorials

1.Linda

Linda is a novel system for communication and synchronization.
In **Linda**, independent process communicates via an abstract **tuple space**, unlike objects, tuples are pure data; they do not have any associated methods.

2.OpenMP

OpenMP is an open standard API for Shared Memory parallelization in C, C++ and Fortran which consist of compiler directives, runtime routines and environment variables.

In this project, we are going to design a program which creates multiple threads, including the **server thread** and **client threads** using OpenMP. After your program reads instruction into the shared memory, the server thread should do corresponding operation to the tuple space according to the instruction. If there's a result tuple for the instruction (operation **in** or **read**), the result tuple should be sent to the corresponding client. Client thread should wait for the result tuple if the instruction's operation is **in** or **read**, and save the result tuple to the file.

Specifications

1.Data type:

- String : the data between " and ", for example "abc".
- Integer : the data consists of digits and without "", for example 123.

These are three tuples as examples.

("abc",2,5)
("matrix-1",1.6,3.14)
("family","is-sister","Stephany","Robert")

2.Instruction format:

- [clientID] [in/out/read] [field1] [field2]…
- There is a space between each fields.
- The quantity of field will be less than 200.
- The length of each operation will be less than 1024 characters.
- clientID specifies which client want to do this operation.
- clientID should start from 1.

3.Operation without ?

•1 out "abc" 2 5

This instruction means **client 1** want to put tuple **("abc",2,5)** into tuple space. So the **server thread** should put this tuple into the tuple space.

•2 read "abc" 2 5

This instruction means **client 2** need tuple **("abc",2,5)**. So the **client 2** will wait until the server thread find this tuple in tuple space and send this tuple to **client 2**.

***** This tuple in tuple space won't be removed. *****

•2 in "abc" 2 5

This instruction means **client 2** need tuple **("abc",2,5)**. So the **client 2** will wait until the server thread find this tuple in tuple space and send this tuple to **client 2**.

***** This tuple in tuple space will be removed. *****

NOTE: If there are two ("abc",2,5) in tuple space, one **in** operation will removed only one ("abc",2,5) tuple.

4.Operation with ?

•*3* read "abc" 2 ?i

•4 in "abc" 2 ?i

This instruction "searches" the tuple space for a tuple consisting of the string "abc", the integer 2, and a third field containing anything.

If found, the tuple is removed from the tuple space and the variable *i* is assigned the value of the third field.

The fields of the ***in*** primitive, called the template, are compared to the corresponding fields of every tuple in the tuple space. A match occurs if the following three conditions are all met:

1. The template and the tuple have the same number of fields.
2. The types of the corresponding fields are equal.
3. Each constant or variable in the template matches its tuple field.

5. Suspension

•If no matching tuple is present (operation is **in** or **read**), the client thread will suspend until another client thread inserts the needed tuple.

•When the client is suspended, any instruction related to that client will be ignored.

•If two clients are waiting for the same tuple, the client waits earlier will get the tuple first.

Scenarios

3

This integer specifies the number of clients.

1 out "abc" 2 "x"

Client 1 put tuple ("abc",2,“x") into tuple space.

2 in "abc" "2" ?j

There is no match for the request, so client 2 will suspend.

3 in "abc" 2 ?i

Assign string “x” to variable i, then server send tuple(“abc”,2,”x”) back to client 3, the tuple (“abc”,2,“x”) in the tuple space would be removed. And client 3 will save tuple (“abc”,2,”x”) into file. (named 3.txt)

3 in "abc" "2" 7

There is no match for the request, so client 3 will suspend.

1 out "abc" "2" 7

Assign integer 7 to variable j, then server send tuple(“abc”,”2”,7) back to client 2, the tuple(“abc”,”2”,7) in the tuple space would be removed. And client 2 will save tuple (“abc”,”2”,7) into file. (named 2.txt)

1 out "def" j i

Client 1 put tuple ("def",7,"x") into tuple space.

1 out 1 2 3

Client 1 put tuple (1,2,3) into tuple space

2 read "def" ?a ?b

Assign 7 to variable a and “x” to variable b and client 2 would receive a tuple (“def”,7,”x”) but this tuple won’t be removed. And client 2 will save tuple (“def”,7,”x”) into file. (named 2.txt)

2 in "def" a b

Client 2 would receive a tuple (“def”,7 ,”x”) and this tuple will be removed. And client 2 will save tuple (“def”,7,”x”) into file. (named 2.txt)

1 out "abc" "2" 7

Client 3 would receive a tuple (“abc”,”2”,7) and this tuple will be removed. And client 3 will save tuple (“abc”,”2”,7) into file. (named 3.txt)

exit

terminate the program

Output for Scenarios

•If tuple space changes, server should output the tuple space to a file named “server.txt”.

•If tuple space is null, print ().

•When client get the tuple by in/read, client should output to a file named “clientID.txt”

Client1 -> 1.txt
Client2 -> 2.txt
Client3 -> 3.txt

Input	Output			
3(Client number)				
	1.txt	2.txt	3.txt	server.txt
1 out “abc” 2 “x”				((“abc”,2,”x”))
2 in “abc” “2” ?j				
3 in “abc” 2 ?i			(“abc”, 2 ,“x”)	()
3 in “abc” “2” 7				
1 out “abc” “2” 7		(“abc”,”2”,7)		()
1 out “def” j i				((“def”,7,”x”))
1 out 1 2 3				((“def”,7,”x”), (1,2,3))
2 read "def" ?a ?b		(“def”,7,”x”)		
2 in "def" a b		(“def”,7,”x”)		((1,2,3))
1 out "abc" "2" 7			(“abc”,”2”,7)	((1,2,3))
exit				

Note

1. You should write your code in **C/C++** and **“must”** use **OpenMP**.
2. Your program should be executed on the AWS instance.
3. ClientID.txt **must** output by **client thread**, you cannot output the tuple directly by server thread**.**

File submission

File name:<student_id>.cpp or <student_id>.c
Upload it to the new E3 platform.
TA would validate your source codes by cheating detection. Please finish the assignment by yourself.

Reference

[1]Linda
https://en.wikipedia.org/wiki/Linda_(coordination_language)
http://programmingexamples.wikidot.com/linda
[2]OpenMP
https://hpc-wiki.info/hpc/OpenMP