Assignment 4: Allreduce with Pthreads

Due May 8th, 11:59PM

1. Introduction

In this assignment, you will implement a simple allreduce algorithm using pthreads. The required submission includes a C source file (allreduce.c), a Makefile, and a README.

To run this program, you need to launch two separate instances of allreduce at the same time. You can open two terminals.

Each allreduce instance has an array of four integers but with different values: Instance 1 has [0, 1, 2, 3] and Instance 2 has [4, 5, 6, 7]. In the end, each instance should output the sum of these two vectrors: [4, 6, 8, 10].

Each allreduce program creates two threads: one server thread and one client thread. You may use hard-coded port numbers so the two servers do not use the same port. The client should connect to the server in the other instance using the pre-selected port number. The main process should wait for the two threads to finish by calling pthread_join()

You will implement the reduce-scatter algorithm (see Figure 1), which has two phases: the reduce phase and the scatter phase.

1.1 The reduce phase.

In reduce phase, Instance 1 sends [0, 1] to Instance 2 while Instance 2 sends [6, 7] to Instance 1. Upon receiving the data, the server will ADD the received trunks to the corresponding positions in the array.

Upon the finishing of the reduce phase, Instance 1 will have the sum of the last two elements in the array [x, x, 8, 10] and Instance 2 will have the sum of the first two elements in the array [4, 6, x, x].

1.2 The scatter phase.

In the scatter phase, Instance 1 sends [8, 10] to Instance 2 and Instance 2 sends [4, 6] to Instance 1. Upon receiving the data, the server will UPDATE the corresponding elements in position.

When the scatter phase finishes, the main process should print out the values of the array to standard output.

Notes:

1. Be careful about race condition and deadlocks. Use mutex where appropriate to avoid deadlock or unexpected results.

2. Be careful about the synchronization between the reduce and scatter phase. You need to make sure that scatter starts no earlier than the finishing of reduce. You need to analyze the sequence of the operation. If necessary, you may use mutexes to different parts of the array to enforce this sequential operation.
3. Consult Lecture 8 slides for usage of mutex.

2. Submission:

Create a tar.gz file with the source code, Makefile, and the README. Submit it to Canvas before the deadline. The filename for the tarball must be **username-allreduce.tar.gz** where username is your login name on the EIT (netid) system.

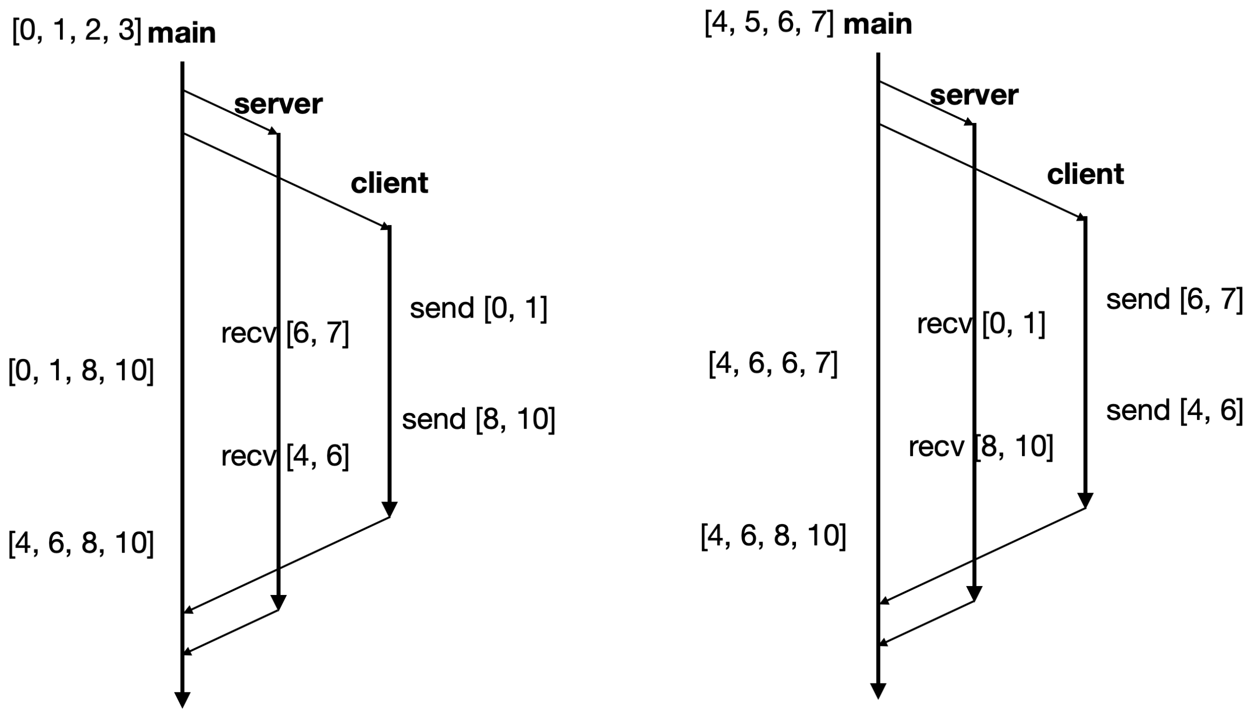

Figure 1: Allreduce Logic Visualization