# CS/CE/TE 6378: Advanced Operating Systems
# Section 002
# Project 1

Instructor: Neeraj Mittal

Assigned on: Wednesday, January 27, 2016
Due date: Wednesday February 24, 2016 (extended)

This is an individual project. *Code sharing among students is strictly prohibited and will result in disciplinary action being taken.*

You can do this project in C, C++ or Java. Each student is expected to demonstrate the operation of this project to the instructor or the TA. Since the project involves socket programming, you can only use machines dcXX.utdallas.edu, where XX $\in \{01, 02, \ldots, 45\}$, for running the program. Although you may develop the project on any platform, the demonstration has to be on dcXX machines; otherwise, you will be assessed a penalty of 20%.

## 1 Project Description

Build a distributed system consisting of $n$ nodes arranged in a certain topology. The value of $n$, the location of each node and its set of neighbors is specified in a configuration file. You can assume that the communication topology specified in the configuration file corresponds to an undirected (symmetric) graph. That is, if node $u$ is a neighbor of node $v$, then $v$ is also a neighbor of $u$. Further, the communication topology generates a connected graph.

Develop and implement a *distributed* algorithm that builds a spanning tree of the system rooted at a given node (specified in the configuration file). Note that, when the spanning tree construction algorithm terminates, each node should know its parent and its children in the tree.

**Bonus Points:** You will get 20% bonus points if the spanning tree your algorithm constructs is also guaranteed to have the smallest height. You will get 10% bonus points if you use SCTP sockets for exchanging messages among nodes.

## 2 Submission Information

**IMPORTANT:** Read all instructions concerning submission **carefully**. Any deviation from these instructions can result in your project being returned **ungraded**. If you have *any* questions, please do not hesitate to e-mail the TA at `k.alex.mills@utdallas.edu`.

All the submission will be through eLearning. Submit a single `.zip` file named `<net-id>.zip` which contains a single folder, named `<net-id>`, which contains all the source files necessary to compile the program and run it, along with files named `cleanup.sh` and `launcher.sh`. You **must** also include `launcher.sh` and `cleanup.sh` scripts. `launcher.sh` will compile and run your program

on the machines as configured. It will automatically login to all the configured machines and launch your processes. An example script is included. `cleanup.sh` will login to the configured machines and kill all running processes started by `launcher.sh` Both `launcher.sh` and `cleanup.sh` should accept two command line arguments: the relative path to a configuration file, and a username used to login to the dcXX machines. Examples of launcher.sh and cleanup.sh are included on eLearning, but are not guaranteed to work correctly with your project. You are responsible for ensuring that the version of these scripts which you submit work properly with your project. You will need to setup passwordless-login on the dcXX machines in order to test that your scripts work properly. Instructions for this will be available on eLearning.

Although the given examples use the bash scripting language, you are not required to use the bash scripting language to write your launcher scripts. you may write your launcher scripts in whatever language you choose (i.e. Python). But you must still include `launcher.sh` and `cleanup.sh`. These scripts may include a single invocation which will then run your script. *You must be **absolutely** certain that the scripts which you submit will run on* `csgrads1`, ***without** any additional configuration by the TA.*

## 3 Configuration Format

Your program **MUST** run using a configuration file in the following format:

The configuration file will be a plain-text formatted file no more than 100kB in size. Only lines which begin with an unsigned integer are considered to be valid. Lines which are not valid should be ignored. The configuration file will contain $2n + 1$ valid lines. The first valid line of the configuration file contains two tokens. The first token is the number of nodes in the system. The second is the ID of the root node. After the first valid line, the next $n$ lines consist of three tokens. The first token is the node ID. The second token is the host-name of the machine on which the node runs. The third token is the port on which the node listens for incoming connections. After the first $n + 1$ valid lines, the next $n$ lines consist of a space delimited list of at most $n - 1$ tokens. The $k^{th}$ valid line after the first line is a space delimited list of node IDs which are the neighbor of node $k$. Your parser should be written so as to be robust concerning leading and trailing white space or extra lines at the beginning or end of file, as well as interleaved with valid lines. The `#` character will denote a comment. On any valid line, any characters after a `#` character should be ignored.

You are responsible for ensuring that your program runs correctly when given a valid configuration file. **MAKE NO ADDITIONAL ASSUMPTIONS** concerning the configuration format. If you have any questions about the configuration or output format, please ask the TA.

Listing 1: Example configuration file named `config.txt`

```
# two global parameters (see above)
5 2

0 dc02 1234   # nodeID hostName listenPort
1 dc03 1233
2 dc04 1233
3 dc05 1232
4 dc06 1233

1 3 4    # space delimited list of neighbors for node 0
0 2 3 4  # space delimited list of neighbors for node 1
```

```
1 3       # ...                    node 2
0 1 2 4   # ...                    node 3
0 1 3     # ...                    node 4
```

The adjacency list you are given in this configuration file is for an *undirected* graph. You **will** be given a symmetric adjacency list for grading, but you should not (in general) assume the input will be symmetric (on general principle). Be wary of double-digit issues in case of a large number of nodes, and do not assume that the adjacency lists will be given in order of node ID.

# 4 Output Format

Your submitted programs should not print **any** output to the command line/shell during their execution. Anytime that an output statement is printed (using for example `System.out.println()`, it generates a system interrupt which results in additional delays between sending messages. In a distributed system, these extra delays result in message delays which can make it appear that your program runs correctly, *when in fact it does not*. Any submission which prints spurious output or makes use of `sleep()` commands to introduce extra delays *will be penalized*.

If the configuration file is named `<config_name>.txt`, then your program **must** output $n$ output files, named in according to the following format: `<config_name>-<node_id>.out`, where `node_id` $\in \{0, ..., n-1\}$.

The output file for process $j$ should be named `<config_name>-j.out` and should contain two lines. The first line contains a single token, which is the process id of process $j$'s parent. If process $j$ has no parent, this line should be an asterisk: `*`. The second line should consist of a space-delimited list of $n-1$ tokens which are the children of node $j$, in order of process id. If process $j$ has no children, this line should be an asterisk: `*`.

Listing 2: Example output file named `config-0.out`

```
1
*
```

Listing 3: Example output file named `config-1.out`

```
*
0 3
```

Listing 4: Example output file named `config-2.out`

```
3
*
```

Listing 5: Example output file named `config-3.out`

```
1
2 4
```

Listing 6: Example output file named `config-4.out`

```
*
3
```

NOTE: It is a little silly to have separate files for this information, but since you have $n$ processes, it is simpler to write to $n$ separate files.

Ensure that this output format is **strictly** followed, because the TA will be using an automated script which expects proper formatting. You can be penalized for failing to follow the proper output format.

# 5 Testing

You are responsible for ensuring that your project runs correctly on any valid input file. You should test your project on multiple topologies and delays, because the TA **will** test your project on multiple configuration files (usually 2-3 of them).