Programming Assignment #1: Zero Configuration Service

Due: Check My Courses

IMPORTANT: You are NOT allowed to include code from any other sources (except the sample code given in this handout). You can submit code that is written by you and your project partners in your group. If someone else turns in a copy of your code, your assignment will be flagged as a copy as well. All students with the same copy will be reported to the disciplinary officer.

In this assignment, you are required to create a **Zero Configuration Service (ZCS)** for a local area network (LAN). As networked computing devices become common place, it is becoming important to implement ZCS. For example, your home might have smart speakers in different rooms, smart alarm clocks, etc. Using a voice assistant (e.g., Alexa) or AI chat assistant you might want to control the devices or even route application actions to one or more of them. You might want to play music through speakers in specific rooms and not others. The speakers themselves would be smart devices with their own processing capability and they would advertise their services (in this case the ability to play media at a certain room) over the network. The voice or AI assistant would run in its own computer in the network and get those advertisements. Using the advertisements received over the network, the voice or AI assistant knows all the speaker availability in the network and selects the best devices to route the service requests.

A home network is typically a LAN with all the devices in the same subnetwork. Large condominiums could be having multiple LANs glued together using routers to interconnect all the devices. For now, let's consider a single LAN. A single LAN can still have large number of devices and the devices can temporary attachments to the network (i.e., some devices are available at certain times and not available at other times). Also, the network user does not want to do any setup to get the devices working in the network – it must be pure plug-and-play (use network services without any configuration). The ZCS is way to achieve the plug-and-play.

Let's consider an example scenario. Suppose the voice assistant is already running in the LAN, but no speakers are currently connected to the network. If a request comes to the voice assistant to play music over available speakers, it will report back with a message saying that there are no output devices. Now, if you plug in a smart speaker in the LAN. The voice assistant must get to know the speaker automatically. It needs to know the IP address and port numbers of the speaker device at the very minimum. The IP address is a valid address in the LAN. In addition, these parameters, the speaker will include other parameters such as media format the speaker can handle, location of the speaker in the house, and other attributes of interest.

ZCS and operating scenarios

The ZCS should work with different types of LANs such are wired and wireless networks. The only restriction is that the underlying LAN should support broadcasting. That is, a station (device) on the LAN should be able to send a message on the network that is going to reach everyone in the network. Any LAN technology that does not allow broadcasting would not allow ZCS to work. Fortunately, WiFi, Ethernet, and other popular LAN technologies support broadcasting at the local scope.

The ZCS must work with different capacity networks. We could have a fast or slow LANs and that should not stop ZCS from working. However, if the LAN is very unreliable with messages timing out

frequently (i.e., a sender needs to wait variable amount of time to get a response from the receiver), then ZCS is going to have problems in such a LAN.

In a LAN, we can have two types of failure scenarios: node failure or link failure. With node failures, we can have a fully functional network connecting all the nodes, but the node you want to connect is not in the network or experiencing some anomalous behaviour. The anomalous behaviour can include the node leaving or shutting down and not offering the advertised service. With link failures, we have nodes all present in the network, but the network links could be down, and it could make it impossible for the any node to connect to any other node or certain subset of the nodes. The ZCS must detect both types of failures and remove the unavailable services from each node.

The failures can be permanent or intermittent. With permanent failures, a node that fails would not come back up without a user intervention (e.g., a repair activity). On the other hand, an intermittent failure is something that can change from working to failed and back to working without any kind of user intervention.

ZCS library API

ZCS would be designed as a library that can be used by any application to use zero network configuration. There will be two types of services that would use this library: (a) services that are interested in advertising their presence and (b) services that are interested in querying the network for the presence of other services. The library exposes an API that can be used by both types.

```
int zcs_init(int type);
int zcs_post_ad(char *ad_name, char *ad_value);
int zcs_query(char *attr_name, char *attr_value, char *node_names[]);
int zcs_get_attribs(char *name, zcs_attribute_t attr[], int *num);
int zcs_listen_ad(char *name, zcs_cb_f cback);
int zcs_shutdown();
void zcs_log();
```

`int zcs_init(int type);`

This initializes the ZCS library. The type parameter is used to indicate whether an app or service is being initialized. This library function must be called before issuing any other calls to the library. It sets up the parameters and performs the initializations necessary for the library to work. Returns a 0 if the initialization was a success. Otherwise, it returns a -1.

`int zcs_start(char *name, zcs_attribute_t attr[], int num);`

This call puts the node online. The node has a name, and it is mandatory. It is an ASCII string without spaces that is NULL terminated. It can have a maximum length of 64 characters including the NULL termination. In addition to the name, the node can have optional attributes. In practice, a node would be started with at least one attribute (e.g., the node type). The attributes are specified as key-value pairs and would remain unchanged until the node shuts down. The key and value fields are ASCII characters. If you want to include integers, floats, or other data types in the attribute values, they must be converted to ASCII strings before specified as attributes. The last parameter specifies the number of attributes passed into the node. Returns a 0 if the node start was a success. Otherwise, it returns a -1, which happens if the start was attempted before the initialization was called.

```
int zcs_post_ad(char *ad_name, char *ad_value);
```

This is used to post an advertisement with the given name and value. Advertisements are different from attributes. While attributes are node properties that are valid until the node goes down, advertisements are messages broadcasted by the node as soon as the above function is executed by the node. The advertisement duration and repeat attempts are pre-set in the ZCS library. The node will attempt to deliver the advertisements to other nodes in the network according to the duration and repeat attempts. Returns the number of times the advertisement was posted on the network. It will return 0 (no posting) to indicate an error condition. This will happen if the posting was called before the node was started.

```
int zcs_query(char *attr_name, char *attr_value, char *node_names[]);
```

This function is used to scan for nodes with a given value for a given attribute. If no matching nodes are found, the call returns a 0. Otherwise, the call returns the number of nodes found in the network. The names of the nodes found are stored in the node_names. A call to zcs_query() can fail to find a matching node if there are no nodes with matching attributes or the calling node is not in the same network as the matching node.

```
int zcs_get_attribs(char *name, zcs_attribute_t attr[], int *num);
```

This function is used to get the full list of attributes of a node that is returned by the zcs_query() function. The first argument is the name of the node. The second argument is an attribute array that is already allocated. The third argument is set to the number of slots allocated in the attribute array. The function sets it to the number of actual attributes read from the node. The return value of the function is 0 if there is no error and is -1 if there is an error.

```
int zcs_listen_ad(char *name, zcs_cb_f cback);
```

This function takes two arguments. The first is a name of the target node and the second is the callback that will be triggered when the target posts an advertisement. The callback has two arguments: name of the advertisement and the value of the advertisement. There is no mechanism for un-listening to an advertisement.

```
int zcs_shutdown();
```

This function is called to terminate the activities of the ZCS by a program before it terminates. The call returns a 0 if it is a success. Otherwise, it will return a -1. For example, if the call to shutdown is made before the node was started it will return a -1.

```
void zcs_log();
```

This function prints the node UP and DOWN logs. That is, every time a node fails (goes down) the observing node makes a note of that event in its log. Similarly, every time a node boots up (comes up) the observing node makes a note. This function prints the log that is maintained at the local node. The log can be truncated once it reaches a predefined length in size or time. There is no return value for this function.

Using the ZCS library

There are two types of programs that can be written using the ZCS library. The first type is server and the second type of application. The server is advertising itself and providing some kind of service for the application to use. For instance, a speaker would be a server because it is offering the audio playback service. Any application that wants to do audio playback would be able to use that service.

The figure below shows an example service construction using ZCS. In Line 6, we initialize the ZCS library. This is a mandatory step. The attributes of the service are defined in Line 7. These attributes are

user-defined and not restricted by ZCS. You can specify any attributes and their values. The common attribute that is expected in all cases is the “type” attribute. In this case, we are defining a speaker, and it is specified as the type. The following attributes are providing additional descriptors for the service. Line 11 starts the service with the attributes and names the service as “speaker-X”. Lines 12-17 posts advertisements from the speaker and they notify any listening node that the speaker is changing its state from mute on to off at certain times. Any application on the listening node can respond according to these advertisements.

Support_Files > C service.c > main()

```
1  #include <unistd.h>
2  #include "zcs.h"
3
4  int main() {
5      int rv;
6      rv = zcs_init(ZCS_SERVICE_TYPE);
7      zcs_attribute_t attribs[] = {
8          { .attr_name = "type", .value = "speaker"},
9          { .attr_name = "location", .value = "kitchen"},
10         { .attr_name = "make", .value = "yamaha"} };
11     rv = zcs_start("speaker-X", attribs, sizeof(attribs)/sizeof(zcs_attribute_t));
12     for (int i = 0; i < 1000; i++) {
13         rv = zcs_post_ad("mute", "on");
14         sleep(10);
15         rv = zcs_post_ad("mute", "off");
16         sleep(10);
17     }
18     rv = zcs_shutdown();
19 }
```

The application program that can query the above service is shown below. You can note that the application does not start a service. It uses the zcs_query() function to look for a node with speaker type. We will get up to 10 nodes with such type. If there are more than 10 nodes, the first 10 nodes to detect will be returned by the function. In this case, the first node checked whether it has additional attributes (we retrieve them using the call in Line 15).

Support_Files > C app.c > ...

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include "zcs.h"

void hello(char *s, char *r) {
    printf("Ad received: %s, with value: %s\n", s, r);
    zcs_log();
}

int main() {
    int rv;
    rv = zcs_init(ZCS_APP_TYPE);
    char *names[10];
    rv = zcs_query("type", "speaker", names, 10);
    if (rv > 0) {
        zcs_attribute_t attrs[5];
    int anum = 5;
        rv = zcs_get_attribs(names[0], attrs, &anum);
        if ((strcmp(attrs[0].attr_name, "location") == 0) &&
            (strcmp(attrs[0].value, "kitchen") == 0)) {
                rv = zcs_listen_ad(names[0], hello);
        }
    }
}
```

The zcs_listen_ad() call in Line 18 hooks up a callback that would run every time an advertisement is posted.

Environment for the assignment

The ZCS library needs to be implemented in Linux. We will provide a tutorial that will help you to setup Docker containers in your laptops. You need at least three docker containers running on a single LAN to carry out this portion of the assignment.

Implementing the ZCS library

The following figure shows the header file zcs.h of the ZCS library.

Support_Files > H zcs.h > zcs_init(int)

```c
#ifndef __ZCS_H__
#define __ZCS_H__

#define ZCS_APP_TYPE                1
#define ZCS_SERVICE_TYPE            2

typedef struct {
    char *attr_name;
    char *value;
} zcs_attribute_t;

typedef void (*zcs_cb_f)(char *, char *);

int zcs_init(int type);
int zcs_start(char *name, zcs_attribute_t attr[], int num);
int zcs_post_ad(char *ad_name, char *ad_value);
int zcs_query(char *attr_name, char *attr_value, char *node_names[], int namelen);
int zcs_get_attribs(char *name, zcs_attribute_t attr[], int *num);
int zcs_listen_ad(char *name, zcs_cb_f cback);
int zcs_shutdown();
void zcs_log();

#endif
```

The ZCS library uses UDP multicasting as the primary mechanism to disseminate messages on the network. With UDP multicasting we will use a multicast address that is accessible to all nodes in the LAN. As nodes start a service, they would use the UDP multicasting to tell every other node a NOTIFICATION about their service. The node that are there would get the notification and register that information in a local table. This local table maintains the status of all services that are available in the network. When a node comes up (a node comes up when it executes a zcs_init()) it sends a DISCOVERY message into the network using a UDP multicast. The DISCOVERY message will trigger each service to resend the NOTIFICATION message again. The NOTIFICATION message is only sent at service start up on when a DISCOVERY is sent by an incoming node. The NOTIFICATION will have full information about the service.

Periodically, each service is going to send a HEARTBEAT message to others. It just has the node name and HEARTBEAT count. If the HEARTBEAT is not coming from a node, others assume that the node has gone down.

The query function reads the information from the local registry. Same way the attribute get also reads from the local registry.

The advertisements are sent using UDP multicasting and they are not stored in the local registry. They are directly delivered to any waiting application. If no application is waiting, the advertisements are just discarded.

For more information on implementing the ZCS library, check the discussions in the ED forum.

Deliverables

The deliverable for this project is the ZCS library. It can be implemented in a single zcs.c file. You can also split the functionality into different C files. In that case, you need to provide a Makefile so that the grader can compile them.

Testing and grading

Testing should be carried out in Linux. Because multiple machines are needed, Docker containers on a Ethernet switch (bridge) provide a suitable environment. Therefore, you are asked to use that setup. The tutorial will show you how to use Docker containers if you are not familiar with it.

You are required to document the code very well. Even if your code works you are expected to structure your code and document it properly. The grader reading your code should be able to understand it very easily. If the grader is unable to understand your code very quickly, the grader can take away points (up to 10 points out of 100 points).

Your program part of the assignment would be graded based on the following guidelines.

Your implementation does not compile: 30 points (max – your TA will look at your code and can provide up to 30 points if the whole logic is there).

Your implementation compiles are runs partially: 60 points (basic functionality is node registration and query – works at least once)

Your implementation runs with node registration and advertisement – only for 2 nodes (one server and one application): 80 points

Your implementation runs with many nodes with registration and advertisement: 90 points

Your implementation runs continuously without crashing: 100 points

The rubric shows the **max points** awarded for meeting the milestones. The grader will make the final decision.

There is a demo for this assignment. The TA will ask questions and test your assignment in a short interactive session. The TA will award each member of the team a grade (can be different). The final grade for the assignment is the minimum of the two marks: one awarded based on the demo + questions and the one awarded before the demo based on the code testing.