Assessment 2 component: Fire alarm unit

Summary

The fire alarm unit is a separate system to the overseer, and its job is to detect a fire emergency (either triggered by manual call-points or detected from temperature sensors) and to signal the building's fire alarms as well as to open the security doors marked fail-safe in the event of one. The system is kept separate from the overseer to ensure a higher level of redundancy.

The fire alarm unit will register itself with the overseer via TCP in the same way that many of the components do. The overseer will then send **UDP** datagrams to the fire alarm unit giving the IPv4 addresses and ports of fail-safe doors, which will need to be opened during an emergency. The fire alarm unit will respond with a UDP datagram telling the overseer that the message was received (otherwise it will be sent again).

Safety-critical component

Due to the fire alarm unit's vital role in handling fire emergencies, this is considered a **safety-critical component** and therefore must be implemented following appropriate safety-critical software standards. Make sure to include a **block comment** near the top of your program's source code documenting and justifying any deviations or exceptions.

Program name

```
firealarm
```

Command-line arguments

```
{address:port} {temperature threshold} {min detections} {detection period (in microseconds)} {reserved argument} {shared memory path} {shared memory offset} {overseer address:port}
```

(The 'reserved' argument was a delay argument that was deleted as it serves no purpose, but has been left in to not break existing implementations. Ignore it.)

Shared memory structure

```
struct {
    char alarm; // '-' if inactive, 'A' if active
    pthread_mutex_t mutex;
    pthread_cond_t cond;
};
```

Limits

To ensure that there is no need for dynamic memory (as this component is considered **safety-critical**) you may assume the following limits are in place:

- The 'min detections' value will be <= 50 (which means the detections list only needs to be 50 entries large)
- There will be <= 100 fail-safe doors, which means the fail-safe doors list only needs to be 100 entries large

Initialisation

On startup, this component will bind its UDP port. The component will then send the following initialisation message to the overseer via TCP:

`FIREALARM {address:port} HELLO#`

Normal operation

The component will perform the following loop:

1. Receive the next UDP datagram
2. If the first four bytes are {'F', 'I', 'R', 'E'}, this is a fire emergency datagram (see **Assessment 2 component: Fire alarm call-point controller**

(https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-fire-alarm-call-point-controller?wrap=1) for the datagram format):
   A. Skip to 6
3. If the first four bytes are {'T', 'E', 'M', 'P'}, this is a temperature update datagram (see **Assessment 2 component: Temperature sensor controller** (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-temperature-sensor-controller?wrap=1) for the datagram format):
   A. If the temperature is below the temperature threshold, loop back to 1
   B. If the timestamp is more than {detection period} microseconds old, loop back to 1
   C. Delete all timestamps older than {detection period} microseconds from the detection list
   D. Add the new timestamp to the detection list
   E. If there are now at least {min detections} entries in the detection list skip to 6
   F. Loop back to 1
4. If the first four bytes are {'D', 'O', 'O', 'R'}, this is a door registration datagram (see the 'Datagram format' section for details):
   A. If the door is not already in the door list, add it
   B. Send a door confirmation datagram to the overseer (see the 'Datagram format' section for details)
5. Loop back to 1
6. Lock the mutex
7. Set 'alarm' to A
8. Unlock the mutex
9. Signal cond
10. For each door on the door list, open a TCP connection to it, send OPEN_EMERG#, then close the connection
11. Receive the next UDP datagram
12. If the first four bytes are {'D', 'O', 'O', 'R'}, this is a door registration datagram:
    A. Open a TCP connection to the newly-registered door, send OPEN_EMERG#, then close the connection
    B. Send a door confirmation datagram to the overseer
13. Otherwise, ignore the datagram
14. Loop back to 11

Datagram format

Different UDP datagram formats are used by the fire alarm unit for its communications with other components. Due to the use of UDP (to simplify handling, as receiving both UDP and TCP data at the same time is complicated without threads, and threads are unwise in safety-critical programming) confirmation/acknowledgement datagrams are needed in some cases; to ensure that data was received.

Each datagram begins with a 4 character header to identify which type of datagram it is, so the fire alarm unit can tell them apart. The recommended way to handle these is to check the first four characters, then to cast the memory address to an appropriate struct pointer type.

"TEMP": Temperature update datagram

Sent by: Temperature sensor controller
Received by: Temperature sensor controller, overseer, fire alarm unit

See **Assessment 2 component: Temperature sensor controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-temperature-sensor-controller?wrap=1)** for this datagram format.

"FIRE": Fire emergency datagram

Sent by: Fire alarm call-point controller
Received by: Fire alarm unit

See **Assessment 2 component: Fire alarm call-point controller (https://canvas.qut.edu.au/courses/14485/pages/assessment-2-component-fire-alarm-call-point-controller?wrap=1)** for this datagram format.

"DOOR": Door registration datagram

Sent by: Overseer
Received by: Fire alarm unit

This datagram has the following format:

```
struct {
    char header[4]; // {'D', 'O', 'O', 'R'}
    struct in_addr door_addr;
    in_port_t door_port;
};
```

'door_addr' and 'door_port' are in the same formats used with IPv4 sockets, in terms of network byte order and so forth. These are filled with the fail-safe security door controller's address and port, so the fire alarm unit can register it for sending EMERG_OPEN# TCP messages later.

"DREG": Door confirmation datagram

Sent by: Fire alarm unit
Received by: Overseer

This datagram has the following format:

```
struct {
    char header[4]; // {'D', 'R', 'E', 'G'}
    struct in_addr door_addr;
    in_port_t door_port;
};
```

The same as the door registration datagram, but with "DREG" replacing the "DOOR" header. The door address and port are copied in so the overseer knows which door registration we are confirming.

Example operation

The program might be executed from the command-line with the following:

```
./firealarm 127.0.0.1:5000 50 10 5000000 0 /shm 32 127.0.0.1:3000
```

The program will shm_open shared memory segment at /shm with an offset of 32 and the size of the struct defined at the top, and mmap it into memory. It will also bind UDP port 5000.

The program will connect to 127.0.0.1:3000 via TCP and send the following message: `FIREALARM 127.0.0.1:5000 HELLO#` then disconnect. It will then attempt to receive a UDP datagram and block until one is received.

If the datagram begins with "FIRE", it will trigger the alarm.

If the datagram begins with "TEMP", it decodes the datagram and checks the temperature and timestamp. If the temperature is less than 50 or the timestamp is more than 5000000 microseconds old, it ignores it. Otherwise, its timestamp is added to the detections list. If this brings the length of the detections list up to 10 (after removing any detections that are now over 5000000 microseconds old), the alarm will be triggered.

If the datagram begins with "DOOR", it will pull the IPv4 address and port out of the datagram, send a "DREG" confirmation to the overseer (**not** to the address in the datagram) and add that address and port to the door registration list.

Either way, after receiving a datagram, unless it resulted in the alarm being triggered, it will attempt to receive another datagram and repeat the entire process.

If the alarm is triggered, it will lock the mutex, set 'alarm' to A, unlock the mutex and signal the condition variable on it. Next, the program will loop through all registered doors and send them OPEN_EMERG# messages. It will then continue to listen for UDP datagrams, and upon receiving new doors, will immediately send them OPEN_EMERG#.