2018

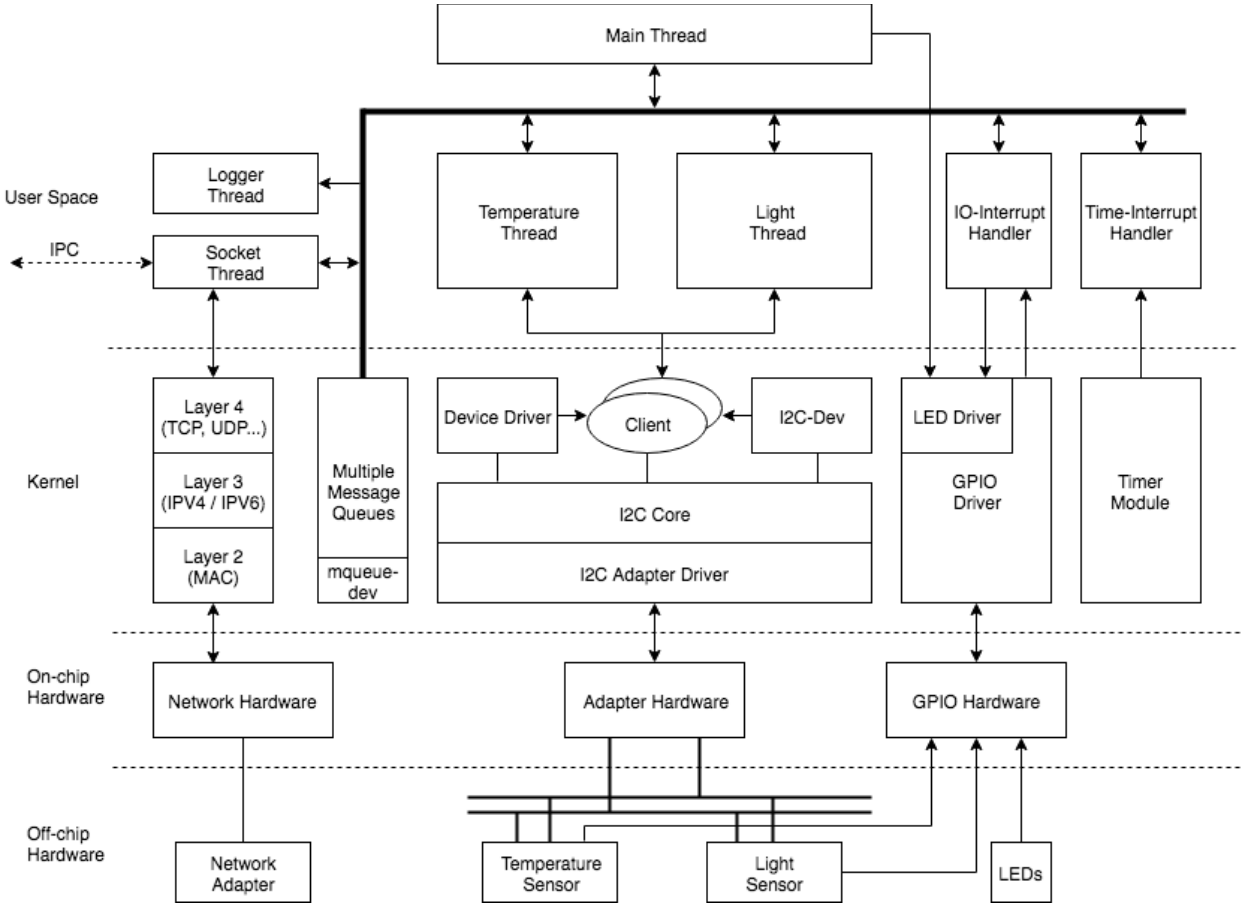
Project 1 Software Architecture

ADVANCED-PES
AKSHIT SHAH | SHUTING GUO

ECEN 5013

Software Architecture for Project 1

1. System Diagram



2. System Tasks Description

As shown in the system diagram, there are mainly five tasks. The task name and their corresponding responsibilities are listed below. Each task will have its own message queue. We will use mutex locks to synchronise the communication.

- Main_thread
    1) Start up tests;
    2) Create all the children tasks;
    3) Ensure that all tasks are still alive and running on regular interval;
    4) Task cleanup;
    5) Open message queue
    6) Push and pop messages to and from the queue

- Temperature thread
    1) Configure the sensor
    2) Open message queue
    3) Periodic temperature reading
    4) Response to external temperature reading request
    5) Temperature conversion
    6) Push and pop message to and from the queue
    7) Send out heartbeat to main

- Light thread
    1) Configure the sensor
    2) Open message queue
    3) Response to external light state request
    4) Push and pop message to and from the queue
    5) Send out heartbeat to main

- Logger thread
    1) Open message queue
    2) Push and pop messages to and from the queue
    3) Log program logs to file
    4) Send out heartbeat to main

- Socket thread
    1) Open message queue
    2) Push and pop messages to and from the queue
    3) Open TCP socket
    4) Receive external API request and response
    5) Send out heartbeat to main

- Timer Interrupt
    1) Provide timing for the periodic temperature reading
    2) Provide heartbeat timing for other threads

- IO Interrupt
    1) Monitor pin interrupt signals from the sensors
    2) Timely manage LEDs for notifications

3. Software Modules and APIs Description

3.1 I2C Device Driver for the Sensors

A kernel device driver will be developed which wraps around the i2c driver and provides mutex as a synchronization mechanism that arbitrates interaction requests from the software to the sensors.

Besides the init() function and the exit() function, additional file_operations functions are also needed:

- dev_open(): Called each time the device is opened from user space.
- dev_read(): Called when data is sent from the device to user space.
- dev_write(): Called when data is sent from user space to the device.
- dev_release(): Called when the device is closed in user space.

3.2 Message Queues and Structures for Inter-thread Communication

The kernel message queue will be utilized for inter-thread communication. Each task described in the previous section will each be dedicated a m_queue as shown below:

#	***Queue Name***	***Queue Item types***	***Publisher***	***Consumer***
1	MAIN	heart beats of other threads	temperature_thread, light_thread, logger thread, socket_thread	main_thread
2	TEMP	External socket request to read temperature	socket_thread, main_thread	temperature_thread
		Periodic timer activation to read temperature		
		Close request from main		
3	LIGHT	External socket request for light state	socket_thread, main_thread	light_thread
		Close request from main		
4	SOCKET	Temperature response to request	temperature_thread, light_thread, main_thread	socket_thread
		Light state response to request		
		Close request from main		
5	LOGGER	Startup test result notification	main_thread, temperature_thread, light_thread, socket_thread	logger_thread
		Errors reporting		
		temperature data		

		light state data		
		Shutdown notification		
		Close request from main		

message APIs will also be created to provide each task an universal interface to interact with each message queue
mq_send(queue,(const char*)&messageStruct, sizeof(Message_t),1)
mq_receive(queue,(const char*)&messageStruct, sizeof(Message_t),1)

Message Structure

```
typedef enum states
{
	SUCCESS,
	ERROR_READ,
	ERROR_WRITE,
	ERROR_OPEN,
	ERROR_ADDR,
	ERROR_DATA,
	NULL_PTR
}Status_t;

typedef enum reqId
{
	HEARTBEAT,
	STARTUP_TEST,
	INIT,
	LOG_MSG,
	GET_TEMP,
	GET_LIGHT,
	GET_LUX,
	CLOSE_THREAD
}RequesId_t;

typedef enum source
{
	MAIN_THREAD,
	TEMP_THREAD,
	LIGHT_THREAD,
	LOGGER_THREAD,
	SOCKET_THREAD
}Source_t;
```

```
typedef enum destination
{
	MAIN_THREAD,
	TEMP_THREAD,
	LIGHT_THREAD,
	LOGGER_THREAD,
	SOCKET_THREAD
}Dest_t;

typedef enum logtype
{
	INFO,
	WARNING,
	ERROR,
	HEARTBEAT,
	INIT
}LogType_t;

typedef struct message
{
	Source_t sourceId;
	Dest_t destId;
	LogType_t logtype;
	RequesId_t requestId;
	time_t timeStamp;
	Status_t status;
	char dataPayload[256];
}Message_t;
```

3.3 Temperature Sensor APIs

err_t tmp_init(void): Initialize the temperature sensor device
err_t tmp_deinit(void): Deinitialized the temperature sensor device
err_t tmp_writeRegister(uint8_t reg, uint8_t value): Write to a register
err_t tmp_readRegister(uint8_t reg, uint8_t num, uint8_t *data): read ‘num’ of registers
int8_t tmp_getTemperature(void): get temperature reading

3.4 Light Sensor APIs

err_t apds_init(void): Initialize the light sensor device
err_t apds_deinit(void): Deinitialized the light sensor device
err_t apds_writeRegister(uint8_t reg, uint8_t value): Write to a register
err_t apds_readRegister(uint8_t reg, uint8_t num, uint8_t *data): read ‘num’ of registers
uint8_t apds_getLightState(void): get current light state

3.5 Led Driver APIs

err_t ledPinSet(char *devAddr): Set the led

err_t ledPinReset(char *devAddr): Reset the led