**Fall**

16

**Northumbria University**

Operating Systems & Concurrency

Jacob Ryan Aylward

Operating Systems & Concurrency Briefcase Alarm: Software and OS Theory & Concepts report.

08

**Fall**

Table of Contents

Software Report 2

Design and Concurrency 2

Design Overview 3

Concurrent Programming Techniques 3

Alternative Approaches 3

Low-Level System Programming 3

Use of Memory-mapped I/O 3

Event-Handling Polling & Interrupts 4

OS Theory & Concepts 4

Process Management 4

Secure Socket Layer (SSL) 4

Potential Security Violations 4

Symmetric & Public-Key cryptography 4

Man-in-the-middle Attack 5

File System 5

FAT File System 5

Disk Allocation Method 5

FAT 12/16/32 5

# Software Report

The overall intent of this project was to successfully create and develop software that utilizes an uC/OS-II operating system to implement a working security ‘Briefcase’ alarm on an LPC-2378STK ARM board. The software that was developed for this project was written in standard C, which works along side efficiently with the ARM boards provided in class. The security mechanism/device once armed requires a 4-digit combination code within a predefined time interval set between 10-120 seconds. If however the combination code isn’t entered before the predefined allotment of time the device will trigger an alarm using the boards LEDs. By using the board’s joysticks the user is able to navigate throughout the systems composition easily and insert the secure combination code

## Design and Concurrency

We where provided the following guidelines for the implementation and design of our individual projects:

* While the device is disabled the briefcase can be locked or unlocked by pressing up/down with the joystick.
* Toggling left with the joystick enables the devices security mechanism.
* If motion is detected while in the security mode a predefined 4-digit code needs to be entered to disable the alarm. If not the alarm sounds within a predefined set time.
* Toggling the joystick left-right allows the user to cycle through the security code, while up-down increments/decrements the selected digit. Selecting center returns the value.

### Design Overview

In designing my program I used the top-down approach allowing me to break down the system requirements to gain an insight to the compositional layout of the sub-systems in a reverse engineering type fashion. Using this approach enabled me to overview how the system could best be formulated, specifying all first level subsystems. The declared tasks given to me in the syllabus for this assignment are listed as follows: BUTTONS, POT, LED1, LED2, LCD, TIME and ACC. Given the functionality of my program the buttons, joystick, potentiometer and accelerometer tasks where defined as input task, allowing the user to interact with the program during the build process through prompts. Whereas the LCD and LED1&2 are defined as output task which give a virtual/visual display for the user to observe.

Once I had a clear understanding of the design of my program and a clear comprehension of the pseudo code I began the coding portion of this project. I started by declaring the task priorities in an order of importance using an enumeration rather than a #define; this is because enum defines syntactical elements allowing the compiler the ability to type-check, thus making the process of debugging easier. Summarizing this process I then created a task stack for each priority task and subsequent declaration of each variable, all with an individual memory type. Once all the variables where defined and created I moved onto the bulk of the code located in the Application Task Definitions section, starting with setting the OS\_ticker to the highest priority task. The OS\_ticker generates interrupt request on a regular basis allowing for context switching to occur, a procedure that permits the devices central processing unit to change between one task/process to another while ensuring conflict doesn’t transpire (supporting multitasking). After the OS\_ticker was generated a while loop was created for the detection of user interaction via button pressed/released events from the joystick.

Each individual function of the program was then placed inside this task in an order of importance/priority to the program: appTaskAcc(), appTaskLED1(), appTaskLED2(), appTaskPot(), appTaskTime() and appTaskLCD(). Each of these functioning task carry some form of overlapping execution, as well as containing cooperating components; therefore threading was introduced into the program, each having separate flows of execution through synchronization. This is why a fixed size ring buffer/circular buffer along with semaphores where implemented into this project, to prevent interference between task as well as controlling access between multiple processes to a common resource in the concurrent operating system.

### Concurrent Programming Techniques

Throughout this program multiple computations of code are executed during overlapping times instead of occurring sequentially or in a parallel format. To ensure that these multiple concurrent processes can co-operate within the critical section of the program at the same time I’ve implemented the use of semaphores to prevent any race conditions from occurring. These undesirable situations occur when a device or system attempts to preform two or more operations at the same time (Deadlock). Using semaphores allows me to carry out these operations in the proper sequences correctly. These semaphores are declared as variables/abstract data types and are used by multiple processes in my code concurrently. This programming technique is known as mutual exclusion.

Bounded buffers are also used throughout my program codification to smooth out the transient differences between the multiple producers and consumers using the safeBufferInit(), safeBufferGet() and safeBufferPut() functions to enforce mutual exclusion and to avoid any interference. Making sure that the order in which different semaphores are incremented/decremented doesn’t result in deadlock.

### Alternative Approaches

An alternative approach that I could have used while developing my program is using the relativistic programming style of concurrent programming. Where instead of attempting to avoid the conflicts between the readers and writers I design the overall program to tolerate the errors and still conclude the correct response disregarding the order in which I prioritize the functions within my program. This style of coding works, however it’s very sloppy and in large scale projects becomes irrational and impossible to predict how the program will respond. In light of this I chose to stick with my approach because using bounded buffers and mutual exclusion techniques is a cleaner methodology.

## Low-Level System Programming

Low-level system programming languages are able to convert machine code without the dependencies of a compiler or interpreter, resulting in the code running directly on the processor. The overall program code is broken down into smaller more efficient system-dependent functions, represented in 8/16/32 bit wide integers.

### Use of Memory-mapped I/O

Memory-mapped (I/O) performs input/output task between the CPU and peripheral devices. To access these task the memory and registers of the I/O device are mapped to atomic address values that are located as required for access/execution.

#### Example 1 LED

|  |  |
| --- | --- |
| Code: | I/O Functionality: |
| *#include “mbed.h”* | Tells the pre-processor to treat the contents of mbed.h as if they appear in the source program at the point where the directive appears. |
| *DigitalOut myled1(LED1);* | DigitalOut interface is used to configure and control the digital output pin for myled1(LED1). |
| *DigitalOut myled2(LED2);* | DigitalOut interface is used to configure and control the digital output pin for myled2(LED2). |
| *int main(){*  *int value = 0;* | Integer variable value equals 0. |
| *uint32\_t myled1\_bit\_mask = 0;*  *while(1) {* | Typedef unsigned long int myled1 bits per long = 0 (loops forever). |
| *myled1\_bit\_mask = 0x00040000;*  *if(value == 0){* | Threshold registry key hex value for myled1. |
| *LPC\_GPIO1->FIOCLR = myled1\_bit\_mask;*  *} else {* | LPC\_GPIO1->FIOCLR used to turn pin to LOW. |
| *LPC\_GPIO1->FIOSET = myled1\_bit\_mask;*  *}* | LPC\_GPIO1->FIOSET used to turn a pin to HIGH. |
| *myled2 = value; //flip value and wait*  *value = ~ value;*  *wait(0.2);*  *}*  *}* | The value of myled2 osculates causing a wait period to occur. |
| LED1 Port address 1 bit 18 (P1.18-MBED\_LED1) : memory-mapped device | |
| LED2 Port address 0 bit 13 (P0.13-MBED\_LED2) : memory-mapped device | |

#### Example 2 Accelerometer

|  |  |
| --- | --- |
| Code: | I/O Functionality: |
| *#include <mbed.h>*  *#include <MMA7455.h>* | Allows reading parallax MMA7455 3-axis accelerometer module. |
| *bool accInit(MMA7455& acc);*  *Display \*d = Display :: theDisplay();*  *Int main(){*  *MMA7455 acc(P0\_27, P0\_28);*  *LM75B lm75b(P0\_27, P0\_28, LM75B :: ADDRESS\_1);* | Boolean accelerometer init.  Call to the display.  Main method of program. |
| *Init32\_t accVal[3];* | Typedef unsigned long int accelerometer z-coordinate. |
| *accInit(acc);*  *lm75b.open();*  *while (true) {*  *acc.read(accVal[0], accVal[1], acc[2]);* | Initialize the accelerometer.  While true accelerometer reads values to display (Should always be true) |
| *d -> printf(“Acc : %05d, %05d, %05d\n”, accVal[0], accVal[1], accVal[2]);*  *}*  *}* | Display “Acc” and accVal[0,1,2] = “ , , ” |
| *bool accInit(MMA7455& acc){*  *bool result = true;*  *if(!acc.setMode(MMA7455 :: ModeMeasurement)) {*  *//d->printf(“unable to set mode for mma7455!\n”);*  *result = false;*  *}*  *if(!acc.calibrate()) {*  *//d->printf(“Failed to calibrate mma7455!\n”);*  *result = false;*  *}*  *return result;*  *}* | Boolean value accelerometer init = TRUE.  If mode for mma7455 cannot be set Boolean value = FALSE.  If mma7455 cannot be calibrated Boolean value = FALSE.  Result is returned |
| MMA7455 Accelerometer, 8-bit I2C address (0x3A/3B): 0.0.1.1.1.0.1.RW | |
| MMA7455 Accelerometer, 7-bit I2C address (0x1D): 0.0.1.1.1.0.1 | |

### Event-Handling Polling & Interrupts

Event-handling by polling means the status register is constantly read until the device’s status changes to indicate that it has completed the request. Bellow is the code for my accelerometer method. The x,y,z-coordinates of the board are constantly being pulled from the device and put into display for the user, this is not however the polling interrupt part of the code. When the board has been locked and the security mechanism has been enabled, the board will constantly poll until it detects movement. Once this has occurred the board register will indicate that the board has moved and the alarm function will begin, indicating that the request has been completed.

|  |  |
| --- | --- |
| Code accelerometer: | Code alarm: |
| Static void appTaskAcc(void \*pdata) {  while(true) {  acc.read(accVal1, accVall2, accVal3);  if((accVal2 > 10) || (accVal1 >10) || (accVal <-10) || (accVal1 <-10) || (accVal3 > 20)){  message\_t msg;  msg.id = RB\_ACC;  msg.fdata[0] = 1;  safeBufferPut(&msg);  OSTimeDlyHMSM(0,0,0,100);  }  else if((accVal2<10) && (accVal1<10) && (accVal2 >-10) && (accVal1 > -10) && (accVal3 <20)){  message\_t msg;  msg.id = RB\_ACC;  msg.fdata[0] = 0;  safeBufferPut(&msg);  OSTimeDlyHMSM(0,0,0,100);  }  }  } | case RB\_TIME : {  if((countTimer < interval) && (secure == 1) && (bMoving == 1)){  countTimer++;  }  else if((countTimer == interval) && (secure == 1)){  strcpy (alarm , "ON ");  flashing[0] = !flashing[0];  flashing[1] = !flashing[1];  }  break;  } |
|  |  |

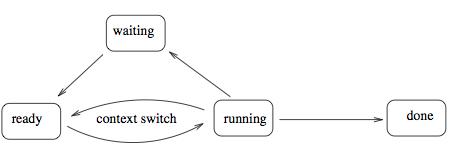
Event-handling by interrupts are produced when a physical state on the device has changed resulting in either an input, error or service call. The joystick on the device triggers this event interrupt whenever its physical location has been moved (up, down, left, right and center) the code for which is as follows:

|  |
| --- |
| Code: |
| while (true) {  message\_t msg;  if (buttonPressedAndReleased(JLEFT)) {  buttonPressed = 1; //Left Joystick Pressed/Released  }  else if (buttonPressedAndReleased(JRIGHT)) {  buttonPressed = 2; //Right Joystick Pressed/Released  }  else if (buttonPressedAndReleased(JUP)) {  buttonPressed = 3; //Up Joystick Pressed/Released  }  else if (buttonPressedAndReleased(JDOWN)) {  buttonPressed = 4; //Down Joystick Pressed/Released  }  else if (buttonPressedAndReleased(JCENTER)) {  buttonPressed = 5; //Center Joystick Pressed/REleased  }  msg.id = RB\_BUTTONS; //Ringbuffer BUTTONS message ID  msg.data[0] = buttonPressed; //Message data Contains Array 0  safeBufferPut(&msg); //Naive buffer implements operation add/put  OSTimeDlyHMSM(0,0,0,100);  } |

# OS Theory & Concepts

## Process Management

My program has multiple processes that occur at the same time (relatively), but in actuality the operating system contains a scheduler that periodically performs context switching at such rapid speeds it gives the appearance of simultaneous processes. It works by taking processes that are in a ‘ready’ state (in a logically concurrent order) and switching them rapidly into a ‘running’ state. Some of these tasks also have a grace period or ‘waiting’ state, which takes them from ‘running’ -> ‘waiting’.



An example of this occurring in my program is in the LED function. If the correct 4-digit combination is not inserted in the pre allocated time frame the alarm will go off, causing LED1 & LED2 to turn on. Instead of having LED1 & LED2 stay constantly on the code implements a flashing delay so that the LEDs osculate between the two states (off/on).

When looking at the user manual for the LPC boards I realized it referred to context switching as having 2 different modes. The thread mode that utilizes the context side of the operating system and the handler mode that deals with the interrupt context.

## Secure Socket Layer (SSL)

The secure socket layer includes two sub-protocols that allow a client to authenticate with a server and establish an encrypted connection. To do this the server presents a digital certificate to the client to authenticate its identity; this is known as the ‘initial handshake’. The initial handshake also allows for the client to authenticate itself to the server as well.) This certificate must follow the standard X.506 format defined by the Public-Key cryptography standards. Once the certificates public-key has been validated and the provider a cipher and shared key are created for the duration of the session. This is to ensure the information shared is encrypted during the exchange for keep the integrity of the data.

### Potential Security Violations

Secure socket layer provides authentication protocols among any client/server communication. To prevent security violations servers must provide a standard X.506 certificate as well as follow all PKCS rules and regulations. Normally data is sent between browsers/servers in plain text, allowing for sniffers or eavesdropping to occur. This is why when a webpage requires a secure connection the Uniform/Universal Resource Locator will change from HTTP (Hypertext Transfer/Transport Protocol) to HTTPS, meaning that the server has been authenticated.

### Symmetric & Public-Key cryptography

Public-Key cryptography is based on encryption/decryption algorithms dependent on keys. The sender encrypts a message they have the intention of sending with the recipients private key, therefore once the message has been encrypted and sent only the intended has the ability to decrypt and read the message. This provides a level of authentication from the sender to the intended just like in SSL. This means that the message cannot be interrupted or tampered with while in transit as well as the sender not being able to subsequently deny having sent the message. This is also used in RSA key-fingerprint encryption between clients and servers; this is because its one of the most secure and reliable forms of encryption. Public keys are able to provide all four assurances while still maintaining easy functionality and distribution. However the algorithms used for creating the keys are based on mathematical functions, which are computationally intensive.

Symmetric/shared key cryptography involves 2 people using the same private key to encrypt and decrypt information, making the process of decrypting messages infinitely faster (1000x). In SSL the cryptographic protocol also limits so that only two computers can exchange methods with one another. This form of cryptography is also easy to implement but it has the added benefit of requiring less processing power. However the methodology of sharing the key between 2 people isn’t guaranteed safe/easy. If someone wants to use this method of cryptography with an individual across the globe then they would need to find a way to share the key securely which defeats the purpose of using symmetric cryptography. Another disadvantage is symmetric cryptography does not support non-repudiativity.

### Man-in-the-middle Attack

Man-in-the-middle attack is when an individual is able to get into a secure communication setup and secretly intercept / alter the communication being sent between the original 2 parties undetected. In doing so an individual is able to read write and distribute messages, while the original 2 parties are under the assumption that everything is still secure. It can also occur when someone poses as an authentic site but publishes a counterfeit public key to intercept messages (usually by sending a person to the real site, encrypted with the real public key).

The Secure socket layer protocol is equipped to constantly check the authentication of both parties using a mutually trusted certification authority. This certification authority determines if a trusted source has signed the digital certificates, and public keys being used. Nevertheless POODLE (Padding Oracle on Downgraded Legacy Encryption) is a method of breaking through the security of secure HTTPS between a client/server. This occurs when SSLv2 & SSLv3 (old versions of SSL) became obsolete individuals who haven’t configured/updated browsers to use TLS/SSL became susceptible to attacks.

## File System

Suppose that an SD card device is added to the LPC2378-STK for additional storage capability to the device, so that a logs can be kept of all access attempts and alarm events that occur. The SD card uses FAT16 as the file system.

### FAT File System

File Allocation Table (FAT) systems use linked allocations where the pointers reside in a separate FAT inside the volume/partition. It provides mapping between the basic units of logical storage on a disk (clusters) at the OS level. Where the physical location of data is addressed by using the drives hardware controller. FAT contains an entry for every file that is stored on the volume (SD), which then contains the addresses of the files starting cluster. Each of these clusters contains a pointer to the next cluster in the file, except the last, which contains an end-of-file indicator.

|  |
| --- |
| FAT log file of alarm device: |
| Macintosh HD:Users:JacobAylward:Desktop:Screen Shot 2015-12-10 at 6.04.39 PM.png |
| Modified, new block were allocated to the file: |
| Macintosh HD:Users:JacobAylward:Desktop:Screen Shot 2015-12-10 at 6.04.48 PM.png |

### Disk Allocation Method

New Technology File System (NTFS) is a proprietary file system that is fully read/write compatible between Windows/Mac products. It’s limited to 16 Exabytes, that’s the equivalent of one billion gigabytes per single Exabyte. An advantage of using this type of formatted hard drive is that they tested faster on benchmark test compared to a FAT16/32 formatted drives. However a disadvantage of writing to a NTFS volume is that unlike a FAT volume, you’re required to use a third party software utility.

### FAT 12/16/32

File Allocation Table 12/16/32 is used almost universally adopted among the majority of non-volatile memory cards. Where memory is used for the task of long-term storage. This is because FAT is compatible universally and is often the default standard for most portable devices.

# Citation:

* Tanmay Patange. "How to Defend Yourself against MITM or Man-in-the-middle Attack." *Hackerspace*. HackerSpace, n.d. Web. 10 Dec. 2015.
* JOEL SANTO DOMINGO. "FAT32 vs. NTFS: Choose Your Own Format." *PCMag UK*. N.p., 16 July 2013. Web. 10 Dec. 2015.