# CPS1012: Operating Systems and Systems Programming I

Due on Mon, May 15th, 2017

*Assignment*

**A. Magro, K. Bugeja, K. Vella, J. Ellul**

## Instructions

- This is an individual assignment and carries 40% of the final CPS1012 grade.
- While it is strongly recommended that you start working on the tasks as soon as the related material is covered in class, the firm submission deadline is Monday 15th May 2017. Hard copies are not required to be handed in.
- You may be required to demonstrate and present your work to an exam board.
- A soft-copy of the report and all related files must be uploaded to the VLE upload area by the same deadline. All files must be archived into a single `.zip` or `.tar.gz` archive file. It is the student's responsibility to ensure that the uploaded archive file and all its contents are valid.
- You are to allocate 30 to 40 hours for this assignment.
- Reports (and code) that are difficult to follow due to low quality in the writing-style, organisation or presentation will be penalised.
- You must include all source, make files, any scripts and instructions required to compile the code.

## Background

You have recently completed your undergraduate studies and without much consideration, concluded that a well-earned rest is what you need. It's Monday, the house is empty and you're still in your dressing gown and bed slippers, sipping breakfast coffee at 1pm. Casually flipping through the pages of a local newspaper, an advert catches your eye: "Systems programming is the activity of programming system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user (e.g. word processor), whereas systems programming aims to produce software which provides services to the computer hardware (e.g. disk defragmenter). It requires a greater degree of hardware awareness."

It is not the straight lifting of text from Wikipedia that causes the glint in your eye, but what comes next: "At the FBI, we will challenge you to forge a career with other dedicated employees working for the same critical mission. The ideal candidate will extend the Bureau's Linux systems software with a new interactive shell, according to the provided specifications."

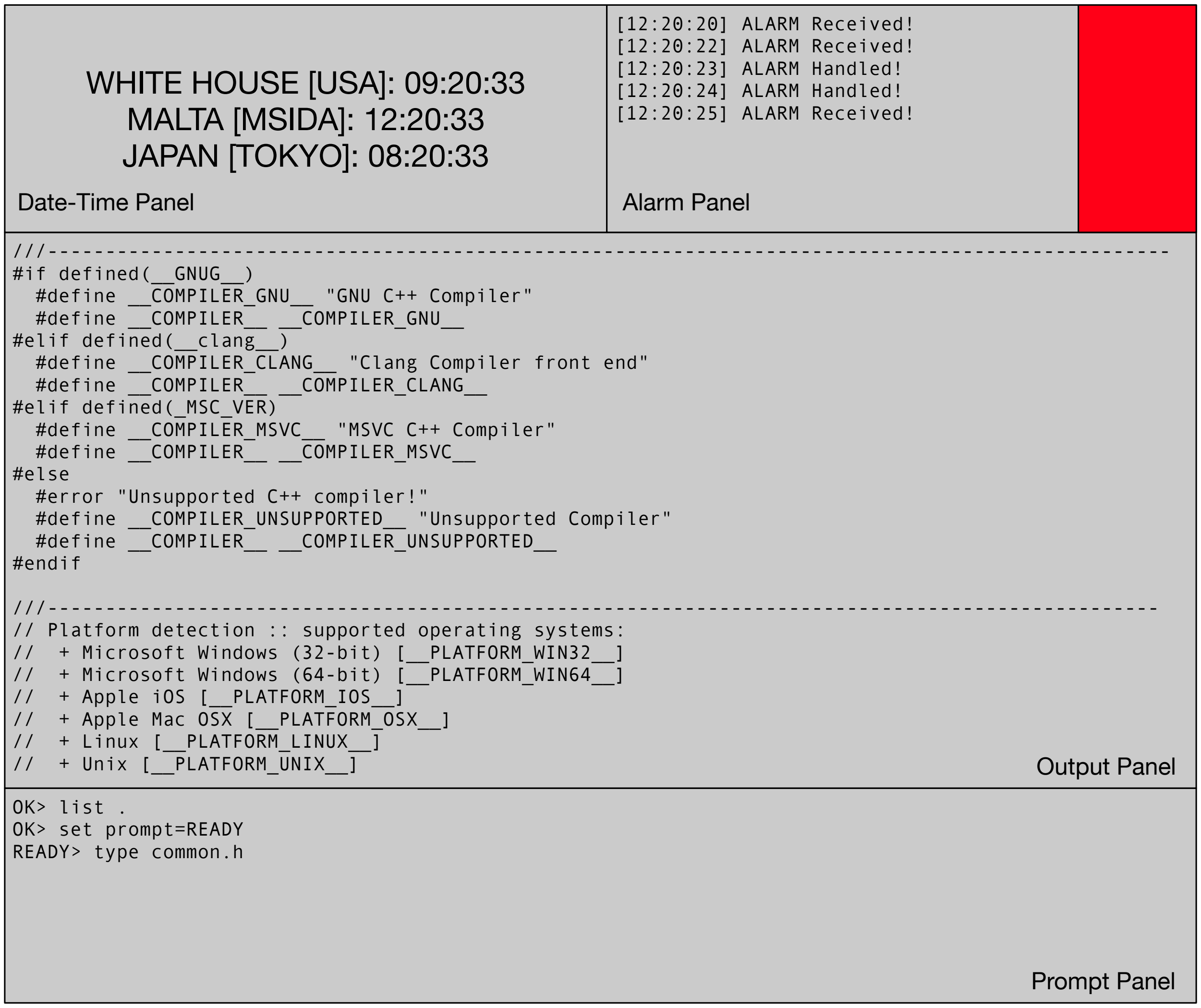
That was all you needed to hear, uh, read. It is `$current_year` and you are $100% sure that calling the embassy will net you the job. All those years tweeting pictures of your basement have earned you street cred as a pedigreed ~~nerd~~ systems programmer, to a level over 9000. You call, demand the job and get it. You half expect "Small Government" to fly into the room and perch itself atop the flag in the kitchen.

## Briefing

On your first day, you are briefed on your mission. The president of the United States of Boletaria has been caught by the Bureau using a nondescript email server for top-secret correspondence, doing questionable dealings with Russia and approving dubious immigration policies. The Bureau has decided that preempting such actions would be the best course of action. Accordingly, their top specialists have designed a specification for a new Linux shell, which they believe to be their trump card in solving the problem. The shell operates under what the agents refer to as the "Big Red Button" paradigm. The folk at the CIA have installed stalling devices and software such that whenever the president interacts with social media or the big red button on his table in the oval office (to nuke a country into orbit), the action is delayed and an alarm raised and pushed to the proposed shell, "Orange Wave".

## Your Mission

Your mission, agent, is to design and develop the shell using your system programming skills. Logically, the shell is comprised of four (4) different components: (i) Date-Time Panel, (ii) Alarm Panel, (iii) Output Panel and (iv) Prompt Panel. These components map to the screen layout shown in Figure 1. For your convenience, the project has been subdivided into a number of smaller tasks, found overleaf. The Bureau highly suggests the use of the **ncurses** library[1] for fine-grained control over the text terminal.



```
WHITE HOUSE [USA]: 09:20:33
MALTA [MSIDA]: 12:20:33
JAPAN [TOKYO]: 08:20:33
```

```
[12:20:20] ALARM Received!
[12:20:22] ALARM Received!
[12:20:23] ALARM Handled!
[12:20:24] ALARM Handled!
[12:20:25] ALARM Received!
```

```
///-------------------------------------------------------------------------------------------
#if defined(__GNUG__)
  #define __COMPILER_GNU__ "GNU C++ Compiler"
  #define __COMPILER__ __COMPILER_GNU__
#elif defined(__clang__)
  #define __COMPILER_CLANG__ "Clang Compiler front end"
  #define __COMPILER__ __COMPILER_CLANG__
#elif defined(_MSC_VER)
  #define __COMPILER_MSVC__ "MSVC C++ Compiler"
  #define __COMPILER__ __COMPILER_MSVC__
#else
  #error "Unsupported C++ compiler!"
  #define __COMPILER_UNSUPPORTED__ "Unsupported Compiler"
  #define __COMPILER__ __COMPILER_UNSUPPORTED__
#endif

///-------------------------------------------------------------------------------------------
// Platform detection :: supported operating systems:
//  + Microsoft Windows (32-bit) [__PLATFORM_WIN32__]
//  + Microsoft Windows (64-bit) [__PLATFORM_WIN64__]
//  + Apple iOS [__PLATFORM_IOS__]
//  + Apple Mac OSX [__PLATFORM_OSX__]
//  + Linux [__PLATFORM_LINUX__]
//  + Unix [__PLATFORM_UNIX__]
```

```
OK> list .
OK> set prompt=READY
READY> type common.h
```

Figure 1: Proposed layout for "Orange Wave".

[1]Official website: https://www.gnu.org/software/ncurses/ncurses.html

## Task 1 — Prompt and Output Panels

The `prompt panel` is how a user interacts with the shell, via command input. The following commands should be supported by the shell as built-in functions:

**chdir** change current directory, followed by the desired directory
(e.g. `chdir /home/sherlock`)

**shdir** output the current directory
(e.g. `shdir`)

**print** output the text that follows the command
(e.g. `print This is a message!`)

**printvar** outputs the value of the specified shell variable
(e.g. `printvar path`)

**set** set internal variables of the shell
(see below for a more detailed description)

**move** scroll through the output buffer by the specified number of lines (argument is an offset, in number of lines; negative values move backwards in history, positive values move forwards)
(e.g. `move -5`)

**exit** close the shell and quit
(e.g. `exit`)

By default, any output goes to the `output panel`; this can be overridden through output redirection: `cmd > $filename`; so for example, typing `shdir` will output `/home/user/development` to the `output panel`, but typing `shdir > myfile.txt` will create or overwrite a file called `myfile.txt` with the text.

### The `set` command

The `set` built-in command is special in that it allows customisation of parts of the shell. The command syntax is `set variable=value`, and the following variables need to be supported:

**`prompt`** changes the prompt string; the default prompt string is `OK>`.

**`path`** string the system uses to locate and launch external binaries; each entry (a path) is separated by **:** from the next, e.g.: `/usr:/usr/bin:`

**`refresh`** changes the refresh rate of the clock in the `date-time panel`. The default value is `1` second.

**`buffer`** changes the size of the output buffer capacity in terms of characters. The default value is `80 by 256` characters.

### External commands

The shell should be able to run external commands using a simple input mechanism: if the entered command is not recognised as a built-in function, it is assumed to be an external command. The system traverses the path string from left to right, one entry at a time, and tries launching the external command using the current path substring. If no such command can be found, the user should be alerted by printing a message in the `output panel`. Example: typing `ls -la` will execute the Unix list command and redirect its output to the `output panel`.

### Output Panel

The `output panel` is a large horizontal strip (multiple lines) on the screen displaying the output buffer in part or whole. The output buffer size (determined by the `buffer` shell variable) can span the size of multiple panels; your buffer might contain more lines that are being displayed by the `output panel`. Therefore the `move` shell command determines which part of the buffer is displayed in the panel.

**[20 Marks]**

## Task 2 — Alarm Panel

The `alarm panel` is the backbone of "Orange Wave". Whether it is a new Tesco discount coupon code added by the NSA to their database by illegally scanning your personal email, or the president ordering a drone strike on Edward Snowden, nothing escapes the gaze of the `alarm panel`. George Orwell would be proud.

Structurally, the `alarm panel` is an independent (child) process, forked off the shell. Its primary job is that of trapping `SIGALRM` signals, from the presidential-blocker daemon. Similarly to the `date-time panel`, the `alarm panel` shares a (shared) memory segment with its parent, the size in characters of its panel on screen. Whenever an alarm is intercepted, this memory is updated with a message reflecting the action. A bar should indicate the frequency of incoming alerts:

**blue** alarm interarrival time greater than 20 s

**green** alarm interarrival time between 16 s and 20 s

**orange** alarm interarrival time between 11 s and 15 s

**red** alarm interarrival time between 5 s and 10 s

**white** alarm interarrival time less than 5 s

**Note:** For two consecutive alarms, $a_{n-1}$ and $a_n$, the interarrival time is computed as follows:

$$time(a_n) - time(a_{n-1})$$

### NSA Support

The hardware and software installed by the NSA can be interfaced with through the presidential-blocker daemon, a program written by the NSA themselves and graciously made available to us at the Bureau. The software takes as an argument a PID (the process id of your shell, which you can acquire using the `ps` command), which it uses to send `SIGALRM` to.

**Note:** The code is supplied by the NSA in bona fide, as is; it is your responsibility to make sure the solution works, even it this means changing the supplied code. The code can be downloaded from:

`https://gitlab.com/cs_prj/nsa_support`

**[10 Marks]**

## Task 3 — Date-Time Panel

The `date-time panel` keeps an up-to-date display of the current time and date for the following time zones:

1. White House [Central Time Zone UTC-06:00]
2. Malta [Central European Time GMT+1]
3. Tokyo [Japan Time Zone UTC+09:00][2]

An independent child process (forked off the shell parent) is responsible for updating a shared memory segment (shared with the parent) with date and time information. The shared segment is the same size, in characters, of the screen area allotted to the panel. Thus, the panel can decide the visual layout of the three clocks.

**Note:** There is no need to factor in daylight saving time.

**[5 Marks]**

## Task 4 — "Orange Wave" Solution

Your superiors at the FBI expect you to write a detailed report for debriefing of this mission. The report should describe the approach taken in solving the different tasks and merging the modules into a single solution. You are expected to use diagrams to illustrate the architecture of your solution and the communication between the different modules. It is important to highlight the (direct or indirect) use of any system calls.

**[5 Marks]**

[2] Although agents in the Bureau train their marksmanship skills on Call of Duty and Battlefield, they hone their martial arts skills exclusively on Japanese games such as Dark Souls, Nioh, Street Fighter and Super Smash Bros. This makes them experts in "Gorilla Warfare," with most having "300 Confirmed Kills". They can kill you in over 700 ways with just their bare hands.