Lab 7 – Signals: "Base to Submarines"

Objectives

In this lab, each student is to write a single program called **`prog7.c`** which has a "base process" which communicates with child processes acting as "submarines" that it creates. The student should exhibit a working knowledge of:

- **Handling processes by use of `fork()`, `getpid()` and `wait()`**
- **Handling signals by use of `signal()`, `alarm()` and `kill()`**
- **Using System Time Functions and generating random numbers.**
- **Opening and accessing a console window buffer**

Program Operation

Before executing the program, the user should open four console (terminal) windows. The first terminal window will represent the base, and the other three will represent the three submarines.

Parent Operation

The first action of the program's main loop is to discover how many terminals are open by trying to open all possible terminal number buffers (e.g. **`/dev/pts/1`**, **`/dev/pts/2`** etc…) as a read only file. Those that are open should be stored so that the first four found can be reopened so they can be written to.

Before spawning child tasks, the main function should display the date and time of the start of the mission.

It should then use **`fork()`** to spawn three child submarine tasks (which are to execute the same exact code).

Child Operation

Each submarine is to be initialized to have a random number between **1000** and **5000 gallons** of diesel fuel, a random number between **six** and **ten ballistic missiles**, and to be a distance of **zero miles** from the sub base.

An **`alarm()`** timer is to be installed and used which:

- Decrements the fuel by a random number between **100** and **200 gallons** each **second**.
- Increments the distance from base by a random number between **5** and **10 miles every two seconds** if going toward target, and decrements the distance from base by a random number between **3** and **8 miles every two seconds** if returning to base.
- Reports the **military time**, **fuel amount**, **ordinance left**, and **distance from base** every **three seconds** in the appropriate terminal. e.g.

  `Time: hh:mm:ss`

  `Sub` *`id`* `to base,` *`g`* `gallons left,` *`b`* `missiles left,`
  `m` **`miles from base.`**

If a submarine runs out of fuel before returning to base or being refueled, it should exit with a code representing an *unsuccessful* mission. As soon as a submarine launches its entire payload it should immediately begin returning to base. If it makes it back to base without running out of fuel, it should exit with a code representing a *successful* mission.

Input/Output

The program should accept the following input (where invalid commands should produce a suitable error message):

- **`l`*`n`*** – Orders submarine ***`n`*** (***`n`*** = 1 to 3) to launch a missile.
- ***`rn`*** – Refuels submarine ***`n`***'s fuel tank to a random number between **1000** and **5000** gallons.

- **`sn`** – Scuttles submarine ***n*** , i.e. kills process for submarine ***n***.
- **`q`** – Exits the program.

Given the **`l`*n*** command, the base should send the signal **`SIGUSR1`** to sub ***n***. Upon receiving the signal **`SIGUSR1`**, the sub should print out that it (submarine ***n***) has launched a missile and display how many missiles it has left. When the sub has no more missiles left, it should print that it has no more ordinance left and that it is returning to base.

Given the **`r`*n*** command, the base process should send the signal **`SIGUSR2`** to sub ***n***. Upon receiving the signal **`SIGUSR2`**, s sub should refuel, resetting its fuel value. Whenever a submarine's fuel drops below **500 gallons**, it should print the message **"`Sub n running out of fuel!`"** If a sub uses all its fuel before being refueled, the sub should send the signal **`SIGUSR2`** to the base, print the message **"`Sub` *`n`* `dead in the water.`"** and then exit with an appropriate code. The base should respond with **"`Rescue is on the way, sub n.`"** in its terminal window.

Given the **`s`*n*** command, the base should terminate the selected child process and note such in its terminal.

When all subs are back at base or a **`q`** command is entered, the base should **1)** terminate any remaining child processes, **2)** display the date and time of the end of the mission, and finally **3)** close the program.

Further Considerations

The program should be structured neatly, easily readable, and well commented. Code should be modularized with functions, logically structured, and written to perform efficiently as possible. Furthermore, variable and function names should be such that the software is as "self-commenting" as possible.

The **`main`** function should mainly contain only functions called to complete the separate tasks given.

Creation and Submission

Each individual student must complete their own program. Copying other students' code will be tested for and will not be tolerated.

Use the following line to compile your program

```
gcc -Wall -g prog7.c -o prog7
```

The code you submit must compile using the **`–Wall`** flag and no compiler errors or warnings should be printed. To receive credit for this assignment the code must compile and at a minimum perform some required function.

Code that does not compile or crashes before performing some required function will not be accepted or graded. **All students must do a final check on one of the CES Ubuntu machines to verify that `gcc` using Ubuntu shows no warning messages before submitting the project.**

Submit your program on Canvas by midnight **Sunday, April 28th**.