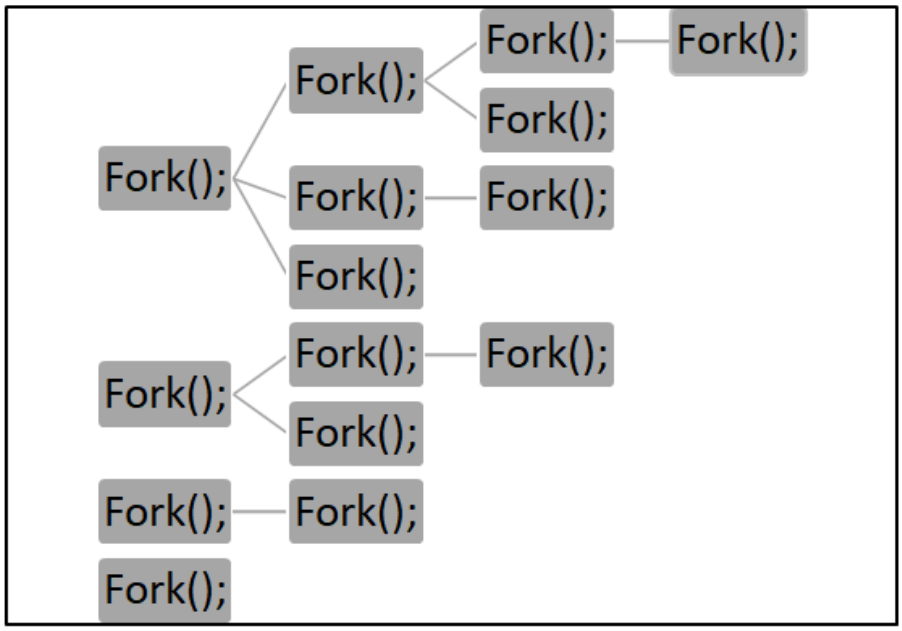
Task 1: Creating some processes

Question 1.1: *How many new processes will a program that contains the code fork(); fork(); fork(); fork(); create? Explain your answer clearly.*

Answer to Question 1.1. The above statements would generate 15 child processes. It will create a tree of child processes that are of the order $((2)^n - 1)$ where (n) is the number of fork(); statements in row, following drawing display the logic:



Question 1.2: *What is the difference between the code snippets in the below listings with respect to the parent-child relationship of the created processes?*

```
pid = fork();
if (pid == 0) {
    fork();
    ...
} else {
    ...
}
```

```
pid = fork();
if (pid == 0) {
    ...
} else {
    fork();
    ...
}
```

Answer to Question 1.2. On the left snipet the main process creates a child process and this child process will create another child process. On the right snippet the main process will create two child processes.

Listing 1.3 - *.C file for the program.*

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>

#include "led_matrix.h"

//Main
int main() {
    if (open_led_matrix() == -1) {
        printf("Failed to initialize LED matrix\n");
        return -1;
    }
    //Declarations
    int row = 0;
    int col = 0;

    pid_t pid = fork();
    if(pid < 0)
      {
        perror("Fork not executed!\n");
        exit(1);
      }
    if(pid == 0)
      {
        row = 7;
        col = 7;
        printf("I'm the child! Lighting LED at (%d, %d)\n", row, col);
        set_led(row, col, RGB565_RED);
      }
    else
      {
        row = 0;
        col = 0;
        printf("I'm the parent! Lighting LED at (%d, %d)\n", row, col);
        set_led(row, col, RGB565_GREEN);
        wait(NULL);
        sleep_ms(2000);
        clear_leds();
      }
    if (close_led_matrix() == -1) {
        printf("Could not properly close LED matrix\n");
        return -1;
    }
}
```

Task 2: Processes and scheduling

Question 2.1: *Describe the progress of the child processes that you can observe on the LED matrix, for the different number of processes that run concurrently. Explain clearly why they behave in this way considering the default scheduling and the number of cores in the Raspberry Pi's processor.*

Answer to Question 2.1. The Raspberry Pi 3 model B has quad core processor architecture.[1] Since all the tasks have equal priority, we see that 4 processes are scheduled on a random basis to run concurrently, we observe that there is not much delay in their execution, and everything runs fine with 4 rows lighting up in a sequence. Once the number of processes become greater than 4 then we start to observe delays as the operating systems uses the queues and schedule the tasks and the LEDs light up in a sequence before terminating.

Question 2.2: *Describe the progress of the child processes that you can now observe on the LED matrix, for the different number of processes that run concurrently. Explain clearly why they behave in this way considering the nice-values and the number of cores in the Raspberry Pi's processor.*

Answer to Question 2.2. It is observed that if the number of tasks are less than or equal to the number of cores on the Raspberry Pi which is 4, then the tasks run simultaneously and the delay cannot be observed by naked eyes. Once the number of tasks become greater than 4 then we could observe that the CPU uses the nice(n) values to schedule these tasks. Nice value is a value that can be used in the user space of linux to change the priority of tasks and has a value ranging between -20 to +19. The lower the nice value implies higher priority. Only the root user can set the nice values in the negative range and by default the CPU uses the value 0 [2]. From the 4th task onwards scheduling and execution can be observed clearly and the tasks run based on the nice values and the LED matrix lights up in a sequence.

Listing 2.3 - *.C file for the program.*

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <unistd.h>

#include <sys/wait.h>
#include "led_matrix.h"

int main()
{
    if (open_led_matrix() == -1) {
        printf("Failed to initialize LED matrix\n");
        return -1;
    }
    clear_leds();

    for (int j = 0; j < COL_SIZE; j++)
      {
        for (int i = 0; i <= j; i++)
          {
        pid_t pid = fork();
        if(pid < 0)
          {
            perror("PID not executed!\n");
            exit(1);
          }
        if(pid == 0)
          {
            nice(i);
            run_child(i);
            pid = fork();
            return 0;
          }
          }

        for(int i = 0; i < COL_SIZE; i++)
          {
        wait(NULL);
          }

        sleep_ms(1000);
        clear_leds();
      }

    if (close_led_matrix() == -1) {
       printf("Could not properly close LED matrix\n");
       return -1;
    }
    return 0;
}
```

Task 3: Write a simple shell

Listing 3.1. - *.C file for the program.*

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>

int main()
```

```
{
  //Declarations.
  char input[20];

 while((strcmp(input, "quit") != 0))
    {
      printf("Group32 > ");
      scanf("%s", input);
      pid_t pid = fork();

      if(pid < 0)
    {
      perror("Fork not executed!");
      exit(1);
    }
      if(pid == 0)
    {
      if(strcmp(input, "ls") == 0)
        {
          execl("/bin/ls", input, (char *) NULL);
          perror("Failed to exec new program");
        }
      else if(strcmp(input, "pstree") == 0)
        {
          execl("/usr/bin/pstree", input, (char *) NULL);
          perror("Failed to exec new program");
        }
      else if(strcmp(input, "htop") == 0)
        {
          execl("/usr/bin/htop", input, (char *) NULL);
          perror("Failed to exec new program");
        }
      else if(strcmp(input, "ifconfig") == 0)
        {
          execl("/sbin/ifconfig", input, (char *) NULL);
          perror("Failed to exec new program");
        }
      else if(strcmp(input, "quit") == 0)
        {
          return 0;
        }
      else
        {
          printf("Command unknown\n");
          return 0;
        }
    }
      else
    {
      wait(NULL);
    }
    }
  wait(NULL);
  return 0;
}
```

References

[1] Raspberry Pi 3 model B specificaion, *[Online] Available:*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed Oct. 03, 2019].

[2] Linux Manual Page, *[Online] Available:*. https://linux.die.net/man/1/nice [Accessed Oct. 03, 2019].