10.a. Understanding File Descriptors and Building a C program to create two processes P1 and P2. P1 takes a string and passes it to P2. P2 concatenates the received string with another string without using string function and sends it back to P1 for printing, send the output to standard output.

Code:

```
// C program to demonstrate use of fork() and pipe()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
	// We use two pipes
	// First pipe to send input string from parent
	// Second pipe to send concatenated string from child

	int fd1[2]; // Used to store two ends of first pipe
	int fd2[2]; // Used to store two ends of second pipe

	char fixed_str[] = "forgeeks.org";
	char input_str[100];
	pid_t p;

	if (pipe(fd1) == -1) {
		fprintf(stderr, "Pipe Failed");
		return 1;
	}
	if (pipe(fd2) == -1) {
		fprintf(stderr, "Pipe Failed");
		return 1;
	}

	scanf("%s", input_str);
	p = fork();

	if (p < 0) {
		fprintf(stderr, "fork Failed");
		return 1;
	}

	// Parent process
	else if (p > 0) {
		char concat_str[100];

		close(fd1[0]); // Close reading end of first pipe
```

```
            // Write input string and close writing end of first
            // pipe.
            write(fd1[1], input_str, strlen(input_str) + 1);
            close(fd1[1]);

            // Wait for child to send a string
            wait(NULL);

            close(fd2[1]); // Close writing end of second pipe

            // Read string from child, print it and close
            // reading end.
            read(fd2[0], concat_str, 100);
            printf("Concatenated string %s\n", concat_str);
            close(fd2[0]);
        }

        // child process
        else {
            close(fd1[1]); // Close writing end of first pipe

            // Read a string using first pipe
            char concat_str[100];
            read(fd1[0], concat_str, 100);

            // Concatenate a fixed string with it
            int k = strlen(concat_str);
            int i;
            for (i = 0; i < strlen(fixed_str); i++)
                concat_str[k++] = fixed_str[i];

            concat_str[k] = '\0'; // string ends with '\0'

            // Close both reading ends
            close(fd1[0]);
            close(fd2[0]);

            // Write concatenated string and close writing end
            write(fd2[1], concat_str, strlen(concat_str) + 1);
            close(fd2[1]);

            exit(0);
        }
}
```

Explanation:

https://www.geeksforgeeks.org/c-program-demonstrate-fork-and-pipe/

Output:

```
administrator@admin:~$ vi 10aa.c
administrator@admin:~$ cc 10aa.c
administrator@admin:~$ ./a.out
www.geeks
Concatenated string www.geeksforgeeks.org
```

b. Create a C program to simulate Grep Unix Command using system calls.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // Check if the correct number of command-line arguments is provided
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Command to simulate 'grep'
    char command[100];
    snprintf(command, sizeof(command), "grep '%s' %s", argv[1], argv[2]);

    // Execute the command using system call
    int system_result = system(command);

    // Check the return value of the system call
    if (system_result == -1) {
        perror("Error using system call");
        exit(EXIT_FAILURE);
    } else if (WIFEXITED(system_result)) {
        // The child process terminated successfully
        int exit_status = WEXITSTATUS(system_result);

        if (exit_status == 0) {
            printf("Pattern found in the file.\n");
        } else {
            printf("Pattern not found in the file.\n");
        }
    } else {
        printf("Child process did not terminate successfully.\n");
    }

    return 0;
}
```

Explanation:

Now, let's go through it step by step:

1. **Header Files:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

- Include necessary header files for standard input/output, dynamic memory allocation, string manipulation, and system calls.

2. **Main Function:**

```
int main(int argc, char *argv[]) {
```

- The `main` function takes command-line arguments: `argc` is the argument count, and `argv` is an array of strings containing the arguments.

3. **Command-Line Argument Check:**

```
if (argc != 3) {
    fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
    exit(EXIT_FAILURE);
}
```

- Checks if the correct number of command-line arguments is provided. If not, it prints a usage message and exits the program with an error code.

4. **Construct 'grep' Command:**

```
char command[100];
snprintf(command, sizeof(command), "grep '%s' %s", argv[1], argv[2]);
```

- Constructs the `grep` command by formatting it as a string with the provided pattern and filename. The command is stored in the `command` array.

5. **Execute 'grep' Command using System Call:**

```
int system_result = system(command);
```

- Uses the `system` function to execute the constructed `grep` command. The result is stored in `system_result`.

6. **Check System Call Result:**

```
if (system_result == -1) {
    perror("Error using system call");
    exit(EXIT_FAILURE);
} else if (WIFEXITED(system_result)) {
```

- Checks if the `system` function encountered an error. If so, it prints an error message and exits the program.

7. **Check Child Process Exit Status:**

```
int exit_status = WEXITSTATUS(system_result);

if (exit_status == 0) {
    printf("Pattern found in the file.\n");
} else {
    printf("Pattern not found in the file.\n");
}
```

- If the child process (executed command) terminated successfully, it checks the exit status. If the exit status is 0, it indicates that the pattern was found, and a corresponding message is printed. Otherwise, it indicates that the pattern was not found.

8. **Handle Child Process Termination:**

```
} else {
    printf("Child process did not terminate successfully.\n");
}
```

- If the child process did not terminate successfully, it prints an appropriate message.

9. **Return from Main Function:**

```
return 0;
```

- Indicates a successful execution of the program.

This program uses the `system` function to execute the `grep` command and then interprets the exit status to determine whether the pattern was found in the specified file.

Execution Procedure:

```
administrator@admin:~$ vi 10bb.c
administrator@admin:~$ cc 10bb.c
administrator@admin:~$ cat>ex.txt
Hi Rohit
Hello Rohit
Bye Rohit
^Z
administrator@admin:~$ cat ex.txt
Hi Rohit
Hello Rohit
Bye Rohit
administrator@admin:~$ ./a.out Rohit ex.txt
Hi Rohit
Hello Rohit
Bye Rohit
Pattern found in the file.
```