Task 3: Process Creation with `fork()`

Task Description:

The task requested for `n` number of child processes in to be created in LINUX environment using `fork()` system call in C language.

- Input Requirements:
  - The user must provide the number for the child processes to be created
- Output Requirements:
  - The program must display the output from each process and should print out the process IDs of both the child processes and parent processes indicating their relationship

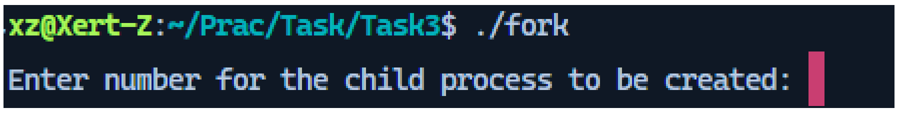
Libraries/Header Files Used:

- `stdio.h` – Provide functions for standard I/O operations
- `stdlib.h` – Provides functions for dynamic memory allocation, process control and conversions of data types (i.e. `string` to `int`)
- `sys/wait.h` – Provides function to work with child processes and their termination
- `sys/types.h` – Defines data types used in system calls.
- `unistd.h` – Provides access to the POSIX operating system API

Detailed Explanation:

➢ **Input:**

- The user must define the number of child processes that he requires. This value is stored in the variable `n` of type `int`. The user must provide `n` greater than 0



```
xz@Xert-Z:~/Prac/Task/Task3$ ./fork
Enter number for the child process to be created: 
```

➢ `fork()` **system call:**

- `fork()` system call is used to create a new process by duplicating the Parent process which is also the calling process. The new process is called the child process, which is exact copy of parent process but has different Process ID
- Both Parent and Child processes execute simultaneously and independently from the same point `fork()` is called
- The child process and parent process both share the same resources but each of them has their own memory space
- Fork returns the following values:
  - `N = 0` in the child process
  - `N > 0` in the parent process. This number is the PID of the child process
  - `N < 0`, no child process is created due to insufficient resources

- In the program, to create n number of child processes from a parent, it will be used in a loop that will keep running until the required number of child processes are not created.

➢ `pid_t` **data type:**
- This data type is used to store the values representing the Process IDs. This helps in identifying processes in OS like Unix.
- We use `childP` to keep track of the process IDs of the child processes created each time `fork()` is called inside the loop

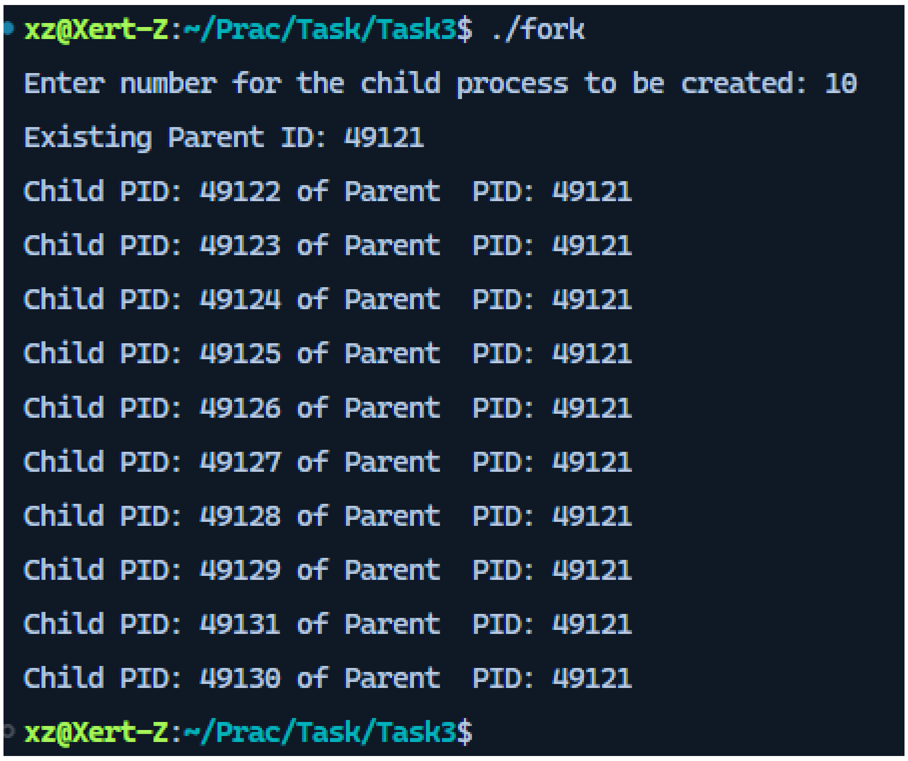
```
pid_t childP = fork();
```

➢ `getpid()` **and** `getppid()` **system calls:**
- `getpid()` returns the value of the calling process or the current running process. If the calling process is the child process `x` in the process tree, it will return the PID of that specific child process.
- `getppid()` returns the value of the parent of the calling process. Used to determine from which process the calling process was created i.e. the PID of the parent process for the child process `x`.

➢ `wait(NULL)` **call:**
- `wait(NULL)` is a function used to force a parent process to wait for any of its child processes in the process tree to terminate. If this function is not used with the `fork()` system calls, then the process creation may also lead to zombie process.
- The `NULL` argument indicates that the parent process doesn't care about the exit status of the terminated child process (i.e whether the child was forcefully terminated or it successfully completed its event)

❖ **Program Flow:**
- The user inputs the required number of child processes to be created.
- The program displays the current process ID which will become the parent process for the n number of children that will be created.
- A `for` loop is used to create `n` child process by calling `fork()  n` times and keeping track of PID of the child process each time `fork()` is called.
- The return value for `fork()` will be checked for each iteration of the `for` loop.
    - For `childP == 0`
        - The PID of both the child and its parent process will be printed
        - The child process will be terminated using `exit()` call.
    - For `childP > 0`
        - The parent process continues to the next iteration to create its next child process
    - For `childP < 0`
        - The parent process will not create a child process due to insufficeitn resources.

- After the loop completes, the parent waits for any remaining child process to complete their termination if there are any.
- The parent completes its execution.



```
xz@Xert-Z:~/Prac/Task/Task3$ ./fork
Enter number for the child process to be created: 10
Existing Parent ID: 49121
Child PID: 49122 of Parent  PID: 49121
Child PID: 49123 of Parent  PID: 49121
Child PID: 49124 of Parent  PID: 49121
Child PID: 49125 of Parent  PID: 49121
Child PID: 49126 of Parent  PID: 49121
Child PID: 49127 of Parent  PID: 49121
Child PID: 49128 of Parent  PID: 49121
Child PID: 49129 of Parent  PID: 49121
Child PID: 49131 of Parent  PID: 49121
Child PID: 49130 of Parent  PID: 49121
xz@Xert-Z:~/Prac/Task/Task3$
```

Figure 1: 10 Child processes created from single parent process