

Ex. 1: **Parent Child Processes Creation using Fork and communicating between them using Pipe**

Objective

You need to develop a program with Fork system call to create a pair of parent & child processes. Then let them communicate with each other using Pipe.

Fork

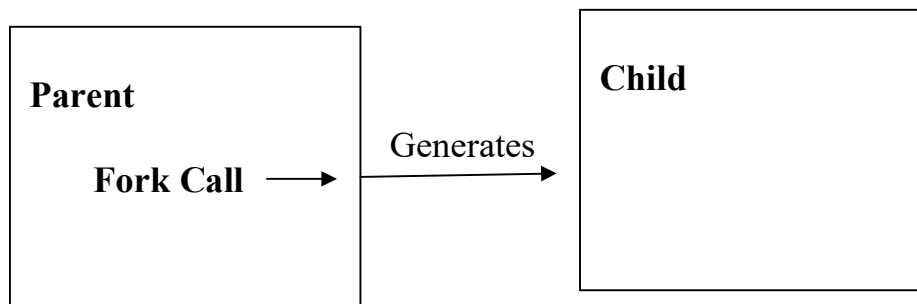
- In computing, particularly in the context of the Unix operating system and its work likes, **fork** is an operation whereby **a process creates a copy of itself** or the parent is replicated. It is usually a system call, implemented in the kernel. Fork is the primary (and historically, only) method of process creation on Unix-like operating systems.
- A new process is created with the *fork* system call.
- When *fork* is called, the operating system creates a new process: it assigns a new process entry in the **process table** and **clones** the content information from the current one.
- All file descriptors that are open in the parent will be open in the child
- The executable memory image is copied as well.
- As soon as the *fork* call returns, both the parent and child are now running at the same point in the program
- Since the parent and child are having exactly the same content, the only way to differentiate them is thorough the return value of fork call.
- The parent gets the **process ID** of the child that was just created. The child gets a return of 0. By checking the return value it could be determined whether what is executed now is the parent or the child.
- If fork returns -1 then the operating system was unable to create the process.
- The crucial thing to note with *fork* is that nothing is shared after the *fork*. Even though the child is running the same code and has the same files open, it maintains its own seek pointers (positions in the file) and it has its own copy of all memory. If a child changes a memory location, the parent won't see the change (and vice versa).

Pipe

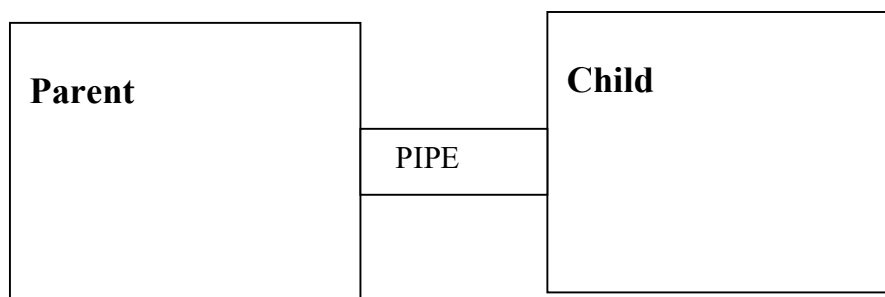
- A **pipe** is a system call that creates a unidirectional communication link between two file descriptors.
- The *pipe* system call is called with a pointer to an array of two integers.

- Upon return, the first element of the array contains the file descriptor that corresponds to the output of the pipe (**stuff to be read**).
- The second element of the array contains the file descriptor that corresponds to the input of the pipe (the place where you **write stuff**).
- **pipe()** creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by *filedes*. *filedes[0]* is for reading, *filedes[1]* is for writing.
- Whatever bytes are sent into the input of the pipe can be read from the other end of the pipe

1. Fork Command



2. Pipe Command



SAMPLE PROGRAM

```

#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    int p[2];
    int pid;
    char inbuf[10],outbuf[10];
    pipe(p); //To send message between parent and child //
  
```

```

pid=fork(); // Fork call to create child process //
if(pid) /// Code of Parent process
{
    printf("In parent process\n");
    printf("type the data to be sent to child");
    scanf("%s",outbuf); // Writing a message into the pipe
    write (p[1],outbuf, sizeof(outbuf)); //p[1] indicates write
    sleep(2); // To allow the child to run
    printf("after sleep in parent process\n");
}
else // Coding of child process //
{
    sleep(2);
    printf("In child process\n");
    read(p[0],inbuf,10); // Read the message written by parent
    printf("the data received by the child is %s\n",inbuf);
}
return 0;
}

```

Sample problem

Pass a list of numbers from parent to the child, sort them and display.