Anonymous Pipes

A pipe has two sides one is reading side and the other is writing side

The reading side is specified by 0 and writing side is specified by 1

Pipe is used for inter-process communication.

One process write in a pipe other reads from the pipe.

So how we will be able to write a code?

We should be using:

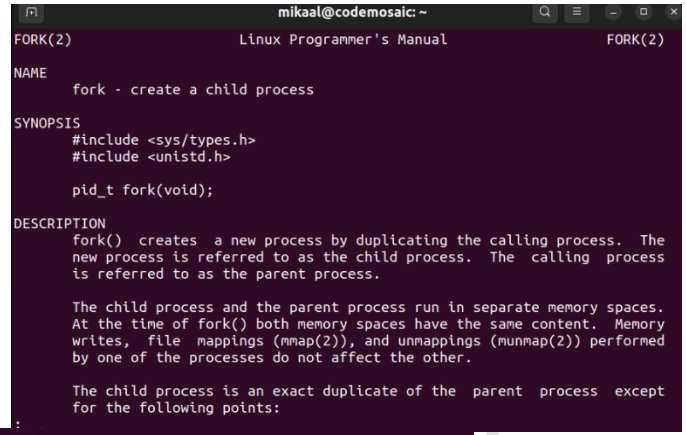
- **Fork()// for creating process**
- **pid_t// data type of a process**
- **pipe(array)//for creation of pipe**
- **close(array with index)**
- **dup2(where it should be directed?,STDOUT_FILENO/STDIN_FILENO)**
- **execl(address of command like "bin/ls",)**
    - **if you don't know the address you can write *which ls* to get address**
- **creat(filename argument eg argv[1],0644 means) for file specifically 0644 allows the owner of file to read and write but others to just read the file**
- **perror(automatically gives an error we can also add a prompt over here)**

So how should we know what should be the header files?

<u>man function/type | less</u>

For example:

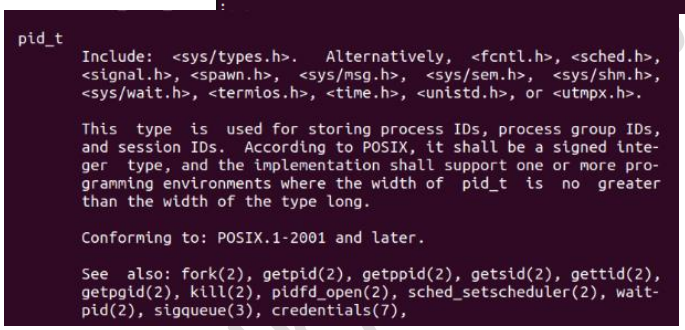
man fork|less: this will show the detail of fork function the header files the usage etc.



```
mikaal@codemosaic: ~
FORK(2)                    Linux Programmer's Manual                    FORK(2)

NAME
       fork - create a child process

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t fork(void);

DESCRIPTION
       fork()  creates  a new process by duplicating the calling process.  The
       new process is referred to as the child process.  The  calling  process
       is referred to as the parent process.

       The child process and the parent process run in separate memory spaces.
       At the time of fork() both memory spaces have the same content.  Memory
       writes,  file  mappings (mmap(2)), and unmappings (munmap(2)) performed
       by one of the processes do not affect the other.

       The child process is an exact duplicate of the  parent  process  except
       for the following points:
:
```

same as for pid_t

man pid_t|less



```
pid_t
       Include:  <sys/types.h>.   Alternatively,  <fcntl.h>, <sched.h>,
       <signal.h>, <spawn.h>,  <sys/msg.h>,  <sys/sem.h>,  <sys/shm.h>,
       <sys/wait.h>, <termios.h>, <time.h>, <unistd.h>, or <utmpx.h>.

       This  type  is  used for storing process IDs, process group IDs,
       and session IDs.  According to POSIX, it shall be a signed inte-
       ger  type, and the implementation shall support one or more pro-
       gramming environments where the width of  pid_t  is  no  greater
       than the width of the type long.

       Conforming to: POSIX.1-2001 and later.

       See  also: fork(2), getpid(2), getppid(2), getsid(2), gettid(2),
       getpgid(2), kill(2), pidfd_open(2), sched_setscheduler(2), wait-
       pid(2), sigqueue(3), credentials(7),
```

So to write code now we are aware what should be the header files.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, char *argv[]){
        pid_t child; // this is for process
        int fd[2]; // For the ends of pipe fd[0] for reading end fd[1] for writing end
        if(pipe(fd)==-1){ //pipe function creates pipe with fd[0] being reading end
        perror("Unable to execute");
        return 1;
        }
        if((child=fork())==-1){// new process initiates if it is child then fork will return 1
```

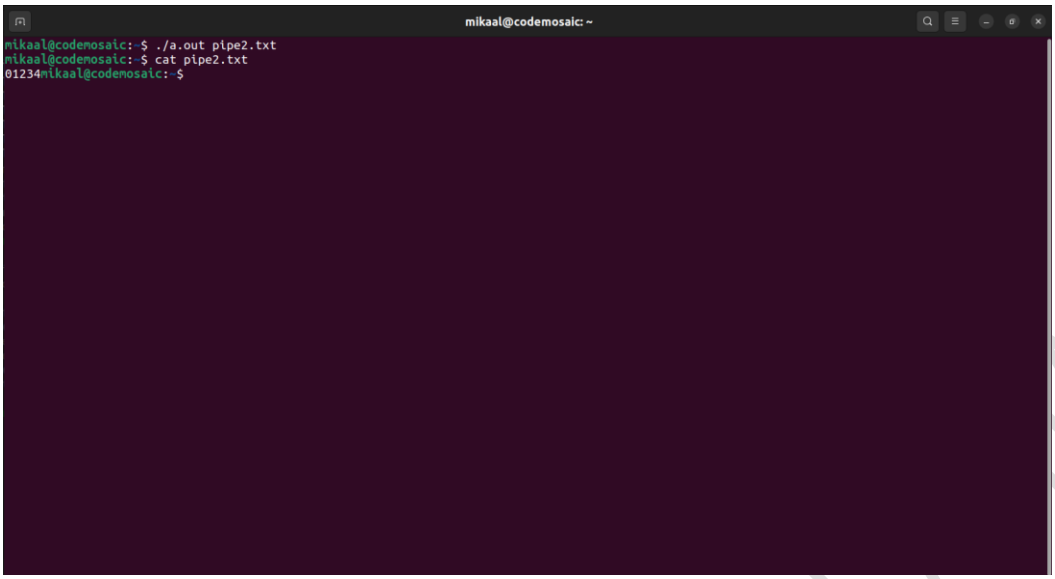
```
        perror("Unable to create child");
        return 1;
        }
        if(child==0){
        dup2(fd[1],STDOUT_FILENO);    //The output of the execl command will be directed towards the write end of the file
        close(fd[0]);// when we have to use execl we have to close both ends so something cannot be entered in pipe
        close(fd[1]);
        execl("/usr/bin/ls","ls", NULL);
        }
        dup2(fd[0],STDIN_FILENO);// The input for the grep command will taken from the read end
        close(fd[1]);
        close(fd[0]);
        execl("/usr/bin/grep","grep","e", NULL);
        return 0;
}
```

Lets redirect a code output to a file

Please note the header files I wrote all of them using man command.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
int main(int argc, char *argv[]){
```

```
int fd[2];

pid_t child;

if(argc!=2){

perror("With <file.exe>< text file name>");

return 1;

}

if(pipe(fd)==-1){

perror("Unable to create");

}

int file=creat(argv[1],0644);//to open a file we can use this function the difference between
//open and creat is that we have to specify more in open like

//Currently it is allowing owner to read and create file but it is allowing group and others to
only read the file

//int fd = open("myfile.txt", O_CREAT | O_WRONLY, 0644);

dup2(file, STDOUT_FILENO); what ever the function which will run it's output will be directed
to the file

close(fd[1]);

close(fd[0]);

for (int i=0;i<5;i++){

printf("%d",i);

}

return 0;

}
```



Another redirect a code output to a file from array.

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

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

int fd[2];

pid_t child;

if(argc!=2){

perror("With <file.exe>< text file name>");

return 1;
```

```
}
if(pipe(fd)==-1){
perror("Unable to create");
}
int file=creat(argv[1],0644);
dup2(file, STDOUT_FILENO);
close(fd[1]);
close(fd[0]);
char array[4][10]={{"mikaal"},{"Hamza"},{"Afaq"},{"Omair"}};
for (int i=0;i<4;i++){
printf("%s\n",array[i]);
}
return 0;
}
```