C++ [Unix Process]

1. fork()

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

int main (int argc, char* argv[]) {
    int id = fork();
    Printf ("Hello world from id: %d \n", id);
    return 0;
}
```

→ -1 if error, 0 for new process and process id of the new process to the old process.

—x—

```
if (id == 0) {
    //Hello from child
} else {
    //Hello from parent
}
```

```
fork();
fork();
```

//4 execution line

```
fork();
fork();
fork();
fork();
```

//16 process

What if 3 process
Add if else.

2. Wait() function.

Req. 1 ... 10

1 ... 5 6 ... 10

Child Parent

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

int main (int argc, char* argv[]){
    int id = fork();
    int n;
    if (id==0) {
        n=1;
    } else{
    n=6;
    int i;
    for(i=n; i<n+5; i++){
        Printf("%d ",i); fflush(stdout);
    }
    Printf("\n");
    return 0;
}
```

Not expected Output

1 2 6 7 3 8 4 5 9 10

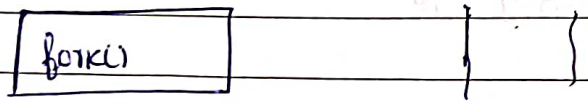
{} two execution line.

// Here is where wait function come into Action.

```
int main (int argc, char* argv[]) {
    int id= fork();
    int n;
    if (id==0) {
        n=1;
    } else {
        n=6;
    }
    if(id!=0){
        wait(NULL);
```

```
int i;
for(i=n; i<n+5; i++) {
    Printf("%d ", i);
    fflush(Stdout);
}
if(id!=0){
    printf("\n");
}
return 0;
}
```

3. Visualizing of fork() call



Both Execution line have different copy of memory it work with

Parent process memory	Child process Memory
argc=1 argv="main.exe"	argc=1 argv="main exe"
n=6 id=4300 i=11	n=1 id=0 i=6

4.

Process Ids

getPid() ← Reterives the process id

getppid() ← Retrives the parent process id.

```
#include <String.h>
#include <stdlib.h>
#include <unistd.h>
#include <Sys/wait.h>

int main (int argc, char* argv[]) {
    int id = forK();
    Printf("Current id: %d, Parent id: %d \n",
                getpid(), getppid());
    return 0;
}
```

Zombie Process :- Is a Concept, It refers to a child process that has terminated its execution but still have its entry in the process table.

Output:-

Current ID: 3784, parent ID: 3778
Current ID: 3778, Parent ID: 3772

In typical C or C++ program, main program PPID is usually processid of the shell or Program executed it.

✡✡

In Unix-like O.S, the process with PID 0 is typically init process. Its Ancestor of all other processes and Serve as root of process hierarchy. Its resp for initializing System and Spawning other processes. [If PPID is itself].

✓ if Child process dies without letting parent know (wait()) it becomes Zombie. (usually work this way as best practice.)

What if parent dies first?? (Special Case)

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

int main (int argc, char*argv[]) {
        int id = fork();
        if (id == 0) {
            sleep(1);   // for child
        }
        Printf ("Current ID: %d, parent ID: %d \n", getPid(),
                                                    getPPid());
        return 0;
}
```

1+2+3

Output :-

Current ID : 3843, parent ID : 3837

child. ← Current ID : 3848, parent ID : 1077

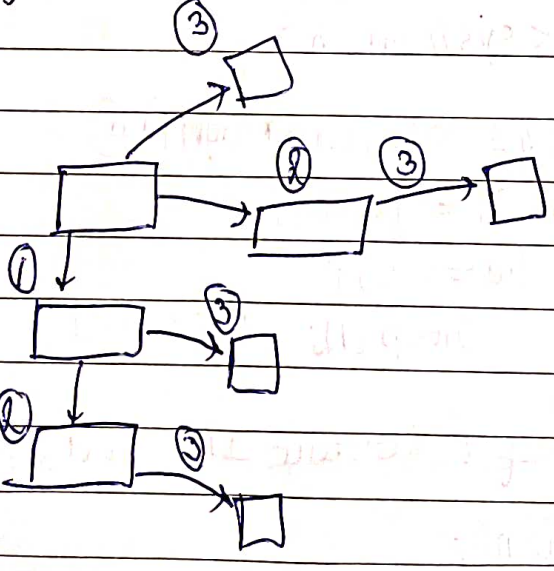
~~Be~~ It was 3843, but as parent died it get assigned different parent Process.

~~if~~

Actual Right way

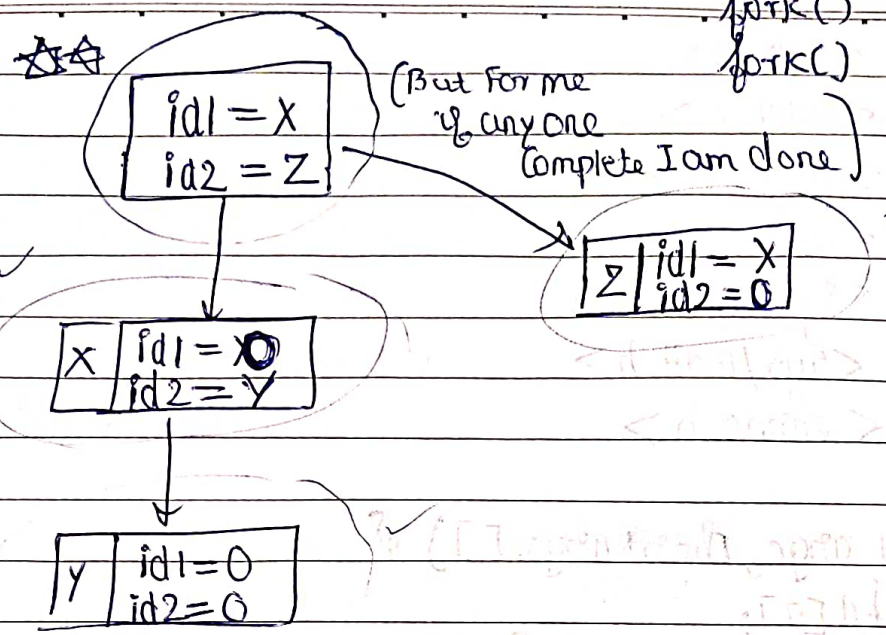
```
int main (int argc, char *argv[]){
        int id = fork();
        if (id == 0){
                sleep(1);
        }
        Printf("Current ID: %d, Parent ID: %d \n",
                                getpid(), getPPid());
        //no need of cond to check
        wait(NULL);          parent
        // can play with return value
}
```

// Multiple fork



fork() → 2
fork() → 4
fork() → 8

Issue With Wait for this ↓



#include < errno.h >

How to wait properly ??

```
// errno != ECHILD
While (wait (NULL) != -1 ~~if(errno != ECHILD);~~
{
    Printf ("waited for child to finish \n");
}
```

ECHILD (for wait) The calling process does not have any unwaited-for children.

—X—

3. Pipe()

Communicating between processes [using pipes] in C

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

int main (int argc, char* argv[]) {
    int fd[2];
    if (pipe (fd) == -1) {
        Printf ("An Error Occured While Opening the pipe\n");
        return 1;
    }
    int id = fork ();
    if (id == -1) {
        Printf ("An error Occured with fork \n");
        return 2;
    }
    if (id == 0) {
        //Child process
        close (fd[0]);
        int x;
        Printf ("Input a number : ");
        Scanf ("%d", &x);
        if (write (fd[1], &x, sizeof (int)) == -1) {
            Printf ("An error Occured with Writing to pipe \n"); return 3;
        close (fd[1]);
    } else {
        //Parent process
        close (fd[1]);
        int y;
        if (read (fd[0], &y, sizeof (int)) == -1) {
            Printf ("An error Occured with reading from pipe");
            return 4;
        }
        Printf ("Got from child process %d \n", y); y = y * 3;
        Printf ("Result is. %d.\n", y); Printf ("Result is %d", y);
        close (fd[1]);
    }
    return 0;
}
```

Sundaram

(Sum using multiple process)

Practical Use Case of pipe() and fork()

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

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

    int arr[] = {1,2,3,4,12,7,7};
    int arrSize = sizeof(arr)/sizeof(int);
    int start, end;
    int fd[2];
    if (pipe(fd) == -1) {
      return 1;
    }
    int id = fork();
    if (id == -1) {
      return 2;
    }
    if (id == 0) {
        start = 0;
        end = arrSize/2;
    } else {
        start = arrsize/2;
        end = arrSize;
    }
    int sum = 0;
    int i;
    for (i = start; i<end; i++) {
      sum += arr[i];
    }
    printf ("Calculated Partial Sum: %d, sum);
    if (id == 0) {
        close(fd[0]);
        if (write(fd[1], &sum, sizeof(int)) == -1) {
          return 3;
        }
        close(fd[1]);
```

```
else {
    int SumFromChild;
    close(fd[1]);
    if (read(fd[0], &SumFromChild, sizeof(int)) == -1) {
    return 4;
    }
    close(fd[0]);
    int totalSum = Sum + SumFromChild;
    printf("Total Sum is %d \n", totalSum);
    wait(NULL);

    return 0;

}.
```

H.W Have 3 Process do Sum.

↳ Code it

✡✡✡ Opening the read or write end of a FIFO blocks until the other end is also opened (by thread or a process)

Introduction to FIFOs (aka named pipes) in C

```
int main (int argc, char *argv[]) {
    int fd[2];
    if (pipe(fd) == -1) {        <-- Copy Shared across processes when forked.
    } return 1;
}
```

You can't have Pipe between processes which are on same Hirearchy? Can you.... no.

There is another concept similar to pipe in Unix System, which is FIFO (named Pipe)

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

int main (int argc, char *argv[]) {
    if (mkfifo("myfifo1", 0777) == -1) {
        if (errno != EEXIST) {
            printf ("Could not create fifo file\n"); return 1;
        }
    }
    printf ("Opening...\n");
    int fd = open ("myfifo1", O_WRONLY);   // Block here  Speciality of FIFO
    printf ("Opened\n");
    int x = 97;
    if (write (fd, &x, sizeof (x)) == -1) { return 2; }
    printf ("written"); close (fd); printf ("Closed");
    return 0;
}
```

Block

If you run alone → Stuck at Opening next line

If you run along with cat → ✓ runs as it read someone at other end

Opening...	cat myfifo1
Opened	"a"
~~Cld~~ Written	
Closed	

Work viceversa as well if someone read but no one to write.

cat myFifo1
← It blocks until control C

—X— ✓

It works perfectly fine if opened in O_RDWR ✓

You don't need C program, you can use mkfifo command

mkfifo myfifo2;

How to use FIFO files to communicate between processes in C

main1.c

```
int main (int argc, char *argv[]) {
    int arr[5];
    srand(time(NULL));
    int i;
    for(i=0; i<5; i++) {
        arr[i] = rand()%100;
    }
    printf("Generated %d\n", arr[i]);
```

main2.c

```
int main (int argc, char *argv[]) {
    int arr[5];
    int fd = open("sum", O_RDONLY);
    if(fd == -1) {
        return 1;
    }
    int i;
```

Research pipe() anonymous pipe (Related processes) (uni too, read multip
named pipe() any two process, unidirectional...

```
int fd = open("sum", O_WRONLY);
if (fd == -1)
    return 1;
if (write (fd, arr, sizeof (int)*5)
        == -1) {
    return 2;
}
close (fd);
return 0;
}
```

```
for (i=0; i<5; i++) {
    if (read (fd, &arr[i], sizeof(int))
        == -1) {
        return 2;
    } Printf ("Received %d \n", arr[i]);
}
close (fd);
int sum = 0;
for (i=0; i<5; i++) {
    sum += arr[i];
}
Printf ("Result is %d \n", sum);
return 0;
}
```

Two Way Communication between processes [using pipes] in C

```
int main (int argc, char* argv[]) {
    int P1 [2];
    int P2 [2];
    if (Pipe (P1) == -1) {
        return 1;
    }
    if (Pipe [P2] == -1) {
        return 2;
    }
    int Pid = fork();
    if (Pid == -1) {
        return 3;
    }
    if (Pid == 0) { //child process
        close (P1[1]); //P1 for read
        close (P2[0]); P2 for write
        int x;
        if (read [P1[0], &x, sizeof (int)) ==
            -1) {return 3;}
        Printf ("Received %d \n", x);
        x *= 4;
```

} else

```
if (write(P2[1], &x, sizeof(int)) == -1) {
        return 4;
}

Printf("Wrote %d \n", x);
close(P1[0]);
close(P2[1]);
} else { // Parent Process
        close(P1[0]);
        close(P2[1]);
        srand(time(NULL));
        int y = rand() % 10;
        if(write(P1[1], &y, sizeof(y)) == -1) {
                return 5;
        }
        Printf("Wrote %d\n", y);
        if(read(P2[0], &y, sizeof(y)) == -1) {
                return 6;
        }
        Printf("Result is %d \n", y);
}       close(P1[1]);
        close(P2[0]);
        wait(NULL);

return 0;
}
```

How to execute another program in C (using exec)

(check program)

Executing Commands in C

(Replacing Current process)
Comm Child replaces parent and Process

~~execlp fn~~

Example

```
int main (int argc, char* argv[]) {
    execlp ("ping", "ping", "-c", "3", "google.com", NULL);
    Printf ("Success!");
    return 0;
}
```

↓

3 packets transmitted
but Success was not printed.
(So what to do)

☆☆ The exec family of functions replaces the current process image with new process image.

```
int main (int argc, char * argv[]) {
    int Pid = fork();
    if (Pid == -1) {
        return 1;
    }.
    if (Pid == 0) {
        execlp("ping", "Ping", "-c", "3", "google.com", NULL);
        Printf ("This should not Print on the terminal \n");
    } else {
        wait(NULL);
        Printf ("Success \n");
    }  Printf ("Some post processing goes here\n");

    return 0;
}
```

Output:-

```
3 packet
Sucess!
Some post processing done here
```

Getting exit code in C

```
#include <stdio.h>
;
;
int main (int argc, char* argv[]) {
    int Pid = fork();
    if (Pid == -1) {
        return1;
    }
    if (Pid == 0) {
        int err = execlp ("ping", "ping", "-c", "3", "google.com", NULL);
// We can skip this →  if (err == -1) {
            Printf ("Could not find program to execut... c or other error occured");
            return 2;
        } else {
            int wstatus;
            wait (&wstatus);
Without signal by exit →  if (WIFEXITED (wstatus)) {
                int status code = WEXITSTATUS (wstatus);
                if (status code == 0) Printf ("Success");
                else printf ("Exit (Failure) with status code %d \n", status code);
            }
        }
    return 0;
}
```

☆☆ → 2 process can have file descriptor no. (3) but unique within a process

0 → Stdin
1 → Stdout
2 → Std-err
3 → PingResult

not change fd, of process id

Before we go to first line in main

dup (PingResult)

0 → Stdin
1 → Stdout
2 → Stderr
3 → PingResults.txt
4 → PingResults.txt

dup2 dup (PingResult.txt, Stdout)

Stdout is closed

0 → Stdin
1 → PingResults.txt
2 → Stderr
4 → PingResults.txt

Redirecting Standard Output in C

```
#include <stdio.h>
:
int main (int argc, char*argv[]) {
    int Pid = fork();
    if (Pid == -1) {
        return1;
    }
    if (Pid == 0) {
        int file = open ("PingResults.txt",
                         OWRONLY|O_CREAT,
                         0777);
        if (file == -1) {
            return 2;
        }
        Printf ("The fd to PingResults : %d \n",
                file);
        dup2 (file, STDOUT_FILENO);
        Close (file);
        int err = execlp ("ping", "ping", "-c",
                          "3", "google.com",
                          NULL);
        if (err == -1) {
            Printf ("Could not find program to execute
                     or other error occured \n");
            return 2;
        } else { int wstatus;
            wait (&wstatus);
            if (WIFEXITED (wstatus)) {
                int status code = WEXITSTATUS (wstatus);
                //0 success else failure with code —3
            }
        }
    }
```

dup / dup2

Return value

↳ On success, these system calls return new file descriptor on error -1 return with errno set

```
}
    return 0;
}
```

all this family execute Program and not Script

Executing Program vs executing Scripts in C

main.c

```
#include
;
```

In C++

exec runs a executable not command.

here rtt is not known on bash

```
which is ping
Ping /bin/Ping ...
>Ping -c1 google.com | grep "rtt"
```

It returns one line work fine It knows.

```
Ping -c1 google.com | grep "rtt"
```

Bash Smart enough to run both executable one after another.

Short introductions to Signals in C

```
int main (int argc, char * argv[]) {
    int Pid = fork();
    if (Pid == -1) {
        return 1;
    }
    if (Pid == 0) {
        while (1) {
            Printf ("Some text goes here \n");
            usleep (50000);
        }
    } else {
        Sleep (1);
        Kill (Pid, SIGKILL);
        wait (NULL);
    }
    return 0;
}
```

* Stopping and Continuing the execution of the process (using SIGCONT and SIGSTOP)

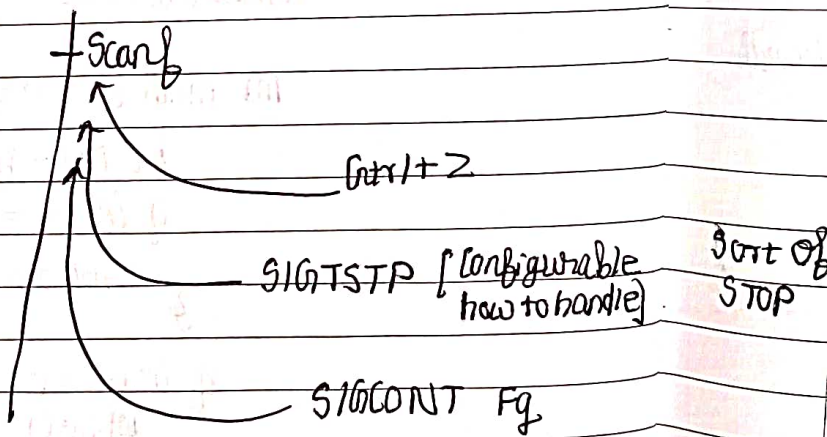
```
#include ...

int main (int argc, char* argv[]) {
    int Pid = fork();
    if (Pid == -1) {
        return 1;
    }
    if (Pid == 0) {
        while (1) {
            Printf ("Some Output");
            usleep (50000);
        }
    } else {
```

```
Kill (Pid, SIGSTOP);
Int t;
do {
  Printf ("Time in Seconds for execution: ");
  Scanf ("%d", &t);
  if (t>0) {
      Kill (Pid, SIGCONT);
      Sleep (t);
      Kill (Pid, SIGSTOP);
  }
} while (t>0);
Kill (Pid, SIGKILL);
Wait (NULL);
}
Return 0;
}
```

Background and foreground processes

STOP (Pause) — diff from → TERMINATED

Ctrl+Z



man → 1 User Command
2 System Command
3 Library Functions
8 System Admin Command

```
#include :

int main (int argc, char* argv[]) {
    int x;
    Printf ("Input Number: ");.   <-- Ctrl+Z (Stop)
    Scanf ("%d", &x);             <-- fg to bring Job to foreground.
    Printf ("Result %d * 5 = %d\n", x, x*5);.
    return 0;
}
```

Handling Signals

(1)

~~Void handle_Signal~~

```
Void handle_Sigcont (int Sig) {
    Printf ("Input Number:");
    fflush (stdout);
}

int main (int argc, char* argv[]) {
    Struct Sigaction. sa;
    Sa. sa_handler = &handle_Sigcont;
    Sa. sa_flags = SA_RESTART;
    Sigaction (SIGCONT, &sa, NULL);
    int x;
    Printf ("Input Number:");
    Scanf ("%d", &x);
    Printf ("Result %d * 5 = %d \n", x, x*5);
    return 0;
}
```

SA_RESTART ✓ (Documentation ✓). → ②

Communicating between processes

SIGUSR1 and SIGUSR2

↓ Only for app^{n} level
No special feature
default beh terminate the process.

```
int x=0;
Void handle_SigUSR(int Sig){
    if(x==0){
        Printf("(HINT) Remember multiplication is repeatitive addition!\n");
    }
}

int main(int argc, char* argv[]){
    int pid=fork();
    if(pid==-1){
        return;
    }
    if(pid==0){
        //child process
        sleep(5); kill(getppid(), SIGUSR1);
    }else{
        //Parrent process
        Sigaction sa; sa.sa_flags=SA_RESTART;
        sa.sa_handler=&handle_SigUSR;
        Sigaction(SIGUSR1,&sa,NULL);
        Printf("What is result of 3x5");
        Scanf("%d\n",&x);
        if(x==15) printf("Right"); else
        return 0; Printf("Wrong");
                  Wait(NULL);
    }
```

(Imp)
* Cannot send data with signal
* This handler should be atomic they should not actually modify, executed between function call

One more program to handle Signal

```
#include <Stdlib.h>
#include <Stdio.h>
#include <Signals.h>

Void handle_SigStp(int Sig){
    printf("Stop not allowed!!");
    fflush(stdout);
}

int main(int argc, char* argv[]){
    struct Sigaction sa;
    sa.sa_handler=&handle_SigStp;
    sa.sa_flags=SA_RESTART;
    Sigaction(SIGTSTP,&handle_SigStp);
```

✓ Preffered

long way

Another way

```
Signal(SIGTSTP,&handle_SigSTP);
```

Why to go from Signal away

Ans man page says it.

The behaviour of signal() varies across Unix version and varied historically across diff distribution of Linux use Sigaction instead.

Sundaram

Sundaram

SIGTSTP ← It didn't stop executed function
SIGCONT ← It execute function and continue execution (way it is implemented)

Some can't be handled.

Send an Array through the pipe

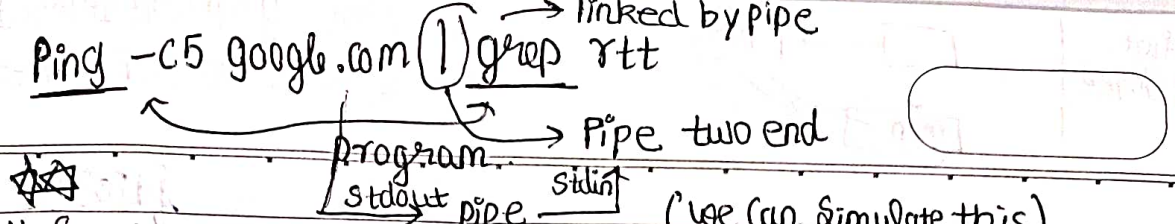
Main.c

```
int main (int argc, char * argv[]) {
    int fd[2];
    if (pipe(fd) == -1) {
        return 1;
    }
    int pid = fork();  if (pid == -1) { return 2; }
    if (pid == 0) {
        // child processes (writing array)
        close(fd[0]);
        int n, i;
        int arr[10];
        srand(time(NULL));
        n = rand()%10+1;
        printf("Generated");
        for (i=0; i<n; i++) {
            arr[i] = rand()%11;
            printf("%d", arr[i]);
        }
        printf("\n");

        if (write(fd[1], &n, sizeof(int)) < 0) {
            return 3;
        }
        printf("Sent n = %d\n", n);
        if (write(fd[1], arr, sizeof(int)*n) < 0) {
            return 4;
        }
        printf("Sent array\n");
        close(fd[1]);
    } else {
```

```
Close (fd[1]);
Int arr[10];
int n, i, sum=0;

if (read (fd[0], &n, sizeof(int))<0) {
    return 6;
}
Printf ("Received n=%d \n");
if ( read (fd[0], &arr, sizeof(int)*n)<0) {
    return 6;
}
Printf ("Received Array \n");
Close (fd[0]);
for(i=0; i<n; i++) {
    printf ("%d ", arr[i]);
    sum += arr[i];
}
printf ("Result of sum is %d \n", sum);
Wait(NULL);
}
return 0;
}
```

How to send a string through a pipe

On same line as above code. (take null if needed).



(Use can simulate this)

Simulating the pipe "|" operator in C

```
int main (int argc, char * argv[]){
    int fd[2];
    if (pipe(fd)==-1){
        return 1;
    }
    int pid1 = fork();
    if (pid1<0){
        return 2;
    }
    if (pid1==0){
        // child process 1 (Ping)
        dup2 (fd[1], STDOUT_FILENO);
        close (Fd[0]);
        close (Fd[1]); // referred by two Fd so dont close
        execlp("ping", "ping", "-c", "5", "google.com", NULL);
    }
    int pid2 = fork();
    if (Pid2<0){
        return 3;
    }
    if (Pid2 ==0){
        // child process 2 (grep)
        dup2(fd[0], STDINFILENO);
        close (fd[0]);
        close (fd[1]);
        execlp ("grep", "grep", "rtt", NULL); // close itself
    }
    close (fd[0]);
    close (fd[1]);
    WaitPid (Pid1, NULL, 0);
    WaitPid (Pid2, NULL, 0);
    return 0;
}
```

// close fd[1] when work over

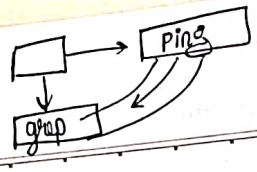
Importance

Hero ⟸ close (fd[0]); close (fd[1]);

Here also stuck there if fd not close

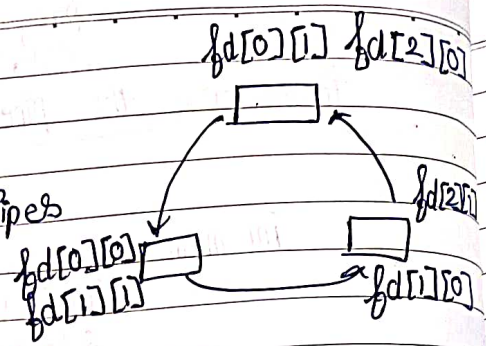
Sundaram

What is done?



Till when grep wait without knowing end if we remove line Hero

What is wait pid?



Working with multiple fork Pipes

```
#include ;
;

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

    int fd[3][2];
    int i;
    for (i=0;i<3;i++){
        if (pipe(fd[i])<0){
            return 1;
        }
    }

    int pid1=fork();
    if (Pid1==0){
            // child processes
        Close (fd[0][1]);
        Close (fd... . Close everything exept (0,0)(1,1)
        int x;
    if ( read (fd[0][0], &x, sizeof (int))<0) {return 3;}
        x+=5
    if (write(fd[1][1], &x, sizeof (int))<0) {return 4;}

    }
    int Pid2=fork();
    if (Pid2<0){
        return 5;
    }
```

6x3 = 18 Close

```
Close (fd[0][0]);
Close (fd[1][1]);
return 0;
```

Wait

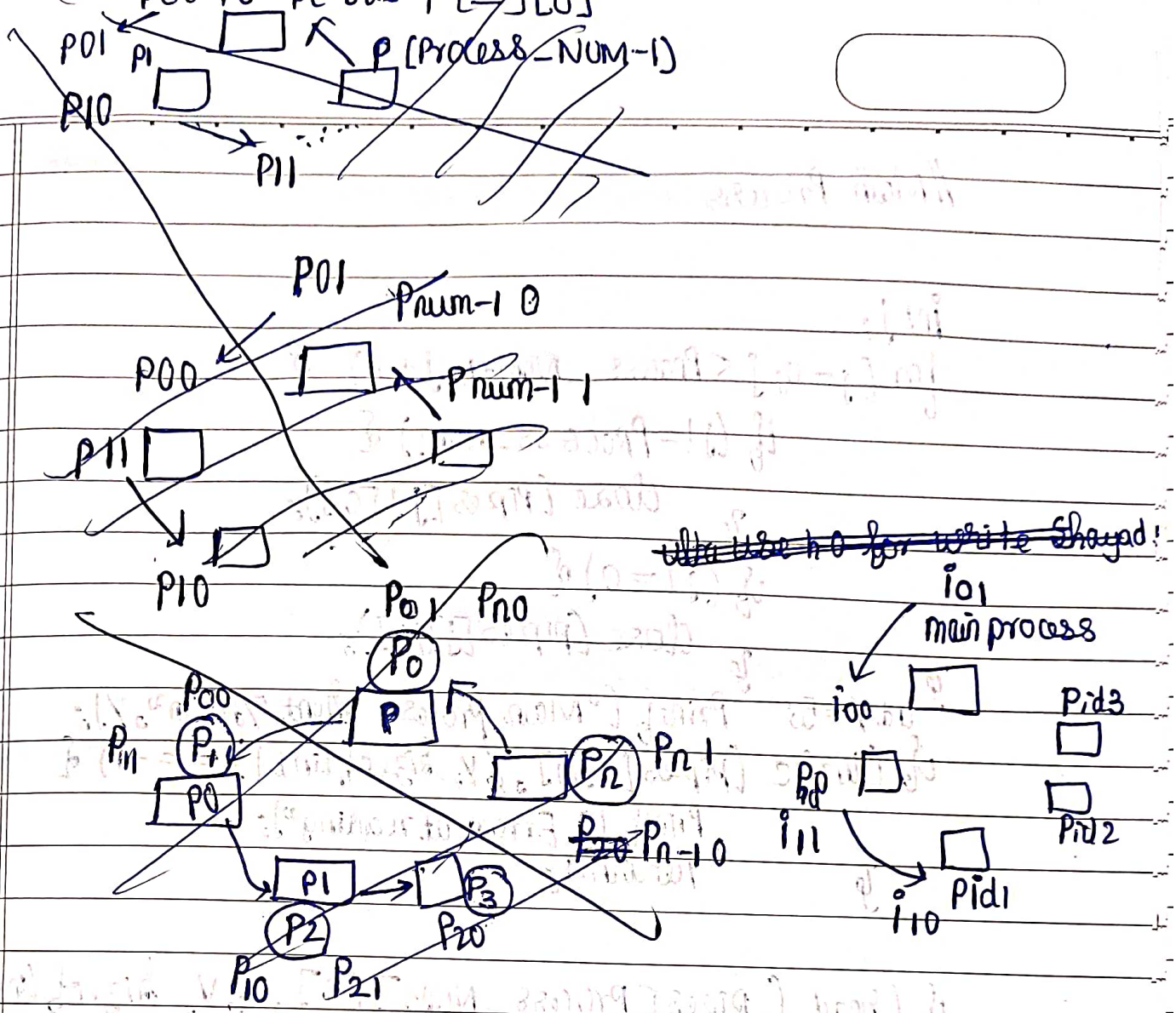
Sundaram

```
if(Pid2==0) {  //Child process 2
    Close all except fd[1][0] and fd[2][1]
int x;
x+=5 if(read[fd[1][0], &x, sizeof(int)]<0){return 8;}
} if(write[fd[2][1], &x, sizeof(int)]<0){return 7;}
  close(fd[1][0]);
  close(fd[2][1]);
  return 0;

  Close all except fd[0][1] and fd[2][0]
  int x=0;
  if(write[fd[0][1], &x, sizeof(int))<0){
    return 8;
  }
  if(read[fd[2][0], &x, sizeof(int))<0){
    return 9;
  }
  Printf("Result is %d \n", x);
  close(fd[0][1]);
  close(fd[2][0]);

  Wait pid(Pid1, NULL, 0);
  Waitpid(Pid2, NULL, 0);
  return 0;
}
```

Waitpid → waiting for a pid
option WNOHANG

Calling fork multiple times

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

#define Process_NUM 10

int main (int argc, char* argv[]) {
    int pids [ProcessNUM];
    int pipes [PROCESS_NUM+1] [2];
    int i;
// for 3 Process 4 pipe
    for (i=0; i< Process_NUM+1; i++) {
        if ( pipe [Pipes [i]) == -1) {
            Printf ("Error while creating pipe \n");
            return 1;
        }
    }

    for (i=0; i< processNum; i++) {
        Pids[i] = fork();
        if (Pids [i]==0) {
            //child processs
            int j;
            for (j=0; j< Process_NUM+1; j++) {
                if (i!=j) {
                    close (Pipes [j] [0]);
                }
                if (i+1 != j) {
                    close (Pipes [j] [1]);
                }
            }
        }
    }
```




```
int x;
if (read (Pipes[i][0], &x, sizeof (int)) == -1) {
    Printf ("Error in reading \n");
    return 3;
}
Printf ("(%d) Got %d \n", i, x);
x++;
if (write ( Pipes [i+1][1], &x, sizeof (int)) == -1) {
    Printf ("~~End of~~ Error at writing \n");
    return 4;
}
Printf ("(%d) Sent %d \n", i, x);
Close (Pipes [i][0]);
Close (Pipes [i+1][1]);
return 0;
}
}
```

```
//Main Process

int j;
for (j=0; j<PROCESS_NUM+1; j++) {
        if (j!=PROCESS_NUM) {
                close (Pipes[j][0]);
        }
        if (j!=0) {
                close (pipes[j][1]);
        }
} int y=5; Printf ("Main process sent %d \n", y);
if (Write (pipes[0][1], &y, sizeof(int)) == -1) {
        Printf ("Error at reading");
        return 4;
}

if (read (pipes[PROCESS_NUM][0], &y, sizeof(int)) == -1) {
        Printf ("Error at reading \n");
        return 3;
}

Printf ("The final result is %d \n", y);
close (Pipes[PROCESS_NUM][0]);
close (pipes[0][1]);
for (i=0; i<PROCESS_NUM; i++) {
        wait(NULL);
}
return 0;
}
```