// fork() system call examples

//Example ………………………………………………………………………………………………………………………………

#include<stdio.h>

#include<unistd.h>

int main(){

fork();

printf("Hello\n");

return 0;

}

//Result : Hello will be output two times ( by parent process and child process)

/\*Note: fork() system call creates a child process. Child process is a copy of the parent process by default. Child process goes live after the successful fork() system call. Hence all statements after a fork system call will be executed twice i.e by the child and parent processes.

\*/

//Example ………………………………………………………………………………………………………………………………

#include<stdio.h>

#include<unistd.h>

int main(){

fork(); //Will be executed by parent process leading to child 1 creation

fork(); // Will be executed by parent process leading to child 2 creation and also executed by

//child 1 leading to child11 process

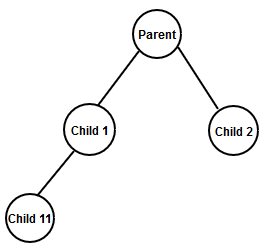
printf("Hello\n");

return 0;

}

/\* Result : Hello will be output four times ( by parent, child1 (created out of first fork), child2 (created out of second fork) and child11 (created out of second fork that is executed by child1). Plz refer the diagram below.

\*/



//Example ………………………………………………………………………………………………………………………………

#include<stdio.h>

#include<unistd.h>

int main(){

fork();

fork();

fork();

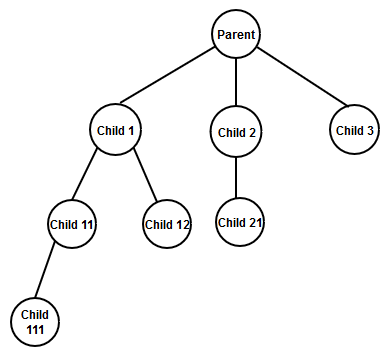
printf("Hello\n");

return 0;

}

/\* Result : Hello will be output eight times as there are eight processes running as given below in the diagram. In general if there are ‘n’ calls to fork then there will be 2n processes created and running.

\*/



//Example ………………………………………………………………………………………………………………………………

#include<stdio.h>

#include<unistd.h>

int main(){

printf("Hello 1\n"); //Will be executed by parent process only

fork();

printf("Hello 2\n"); //Will be executed by parent and child processes

return 0;

}

//Example ………………………………………………………………………………………………………………………………

//General flow in assigning task to child process

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(){

pid\_t pid;

// Statements here will be executed by parent only …………………………………….

pid = fork(); //Creates a child process

if ( pid < 0 ) {

// This block is to check if fork has failed (unsuccessful) if so we return or exit immediately

perror("fork"); //Prints the error message behind the failure of fork

exit( -1);

}

if ( pid == 0 ) {

/\* These statements will be executed by child. Because when child executes this “if” condition it becomes true since fork returns 0 to the child process

\*/

}

else {

/\* These statements will be executed by parent. Because when parent executes this “if” condition it becomes false since fork returns a non-zero positive value (which is the ID of the child process) upon successful child process creation

\*/

}

// Statements here will be executed by parent and child …………………………………….

return 0;

}

//Example ………………………………………………………………………………………………………………………………

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(){

pid\_t pid;

pid = fork();

if ( pid < 0 ) {

perror("fork"); //Prints the error message behind the failure of fork

exit( -1);

}

if ( pid == 0 ) {

printf("Child Process is executing\n");

}

else {

printf("Parent Process is executing\n");

}

return 0;

}

//Example ………………………………………………………………………………………………………………………………

#include <sysltypes.h>

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t pid;

pid =fork();

if (pid < 0) {

fprintf(stderr, "Fork Failed");

return 1;

}

else if (pid == 0) {

execlp("/bin/ls","ls",NULL); // Child process is executing ls command

}

else {

wait (NULL) ; //Parent waits for child to complete its task

printf("Child Completed");

}

return 0;

}

//Example ………………………………………………………………………………………………………………………………

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

pid\_t pid;

printf("Before fork system call");

fflush(stdout); //flushes out the contents of output buffer to standard output device

pid = fork();

if( pid < 0 ) {

fprintf(stderr, "Fork Failed");

return 1;

}

printf("Hello ...\n");

return 0;

}

//Example ………………………………………………………………………………………………………………………………

//Check the behaviour of the program as given firstly

// Secondly by uncommenting fflush(stdout)

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

pid\_t pid;

printf("Before fork system call");

// fflush(stdout);

pid = fork();

if( pid < 0 ) {

fprintf(stderr, "Fork Failed");

return 1;

}

printf("Hello ...\n");

return 0;

}

//Example ………………………………………………………………………………………………………………………………

//Execution context switching between Parent and Child Processes

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

pid\_t pid;

printf("Parent Process is executing before fork()\n");

pid = fork();

if (pid < 0) {

fprintf(stderr, "Fork Failed");

return 1;

}

else if (pid == 0) {

printf("Child Process is executing..BEFORE SLEEP\n");

sleep(1);

printf("Child Process is executing..AFTER SLEEP\n\n\n");

}

else {

printf("Parent Process is executing\n");

}

return 0;

}

//Creating an Orphan Process

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

pid\_t pid;int status;

pid = fork();

if (pid < 0) {

fprintf(stderr, "Fork Failed");

return 1;

}

else if (pid == 0) {

sleep(2);

printf("\nChild Process is executing\n");

}

else

{

return 0;

printf("\nParent Process is executing\n");

}

return 0;

}