Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-08
Processes in UNIX
Practical Programming with C (CSE 3544)

Publish on: 09-12-2024 **Submission on:** 16-12-2024
Course Outcome: CO_5 **Program Outcome:** PO_3 **Learning Level:** L_5

Problem Statement:

Experiment with programs, processes, memory allocation and manipulation for processes in UNIX.

Assignment Objectives:

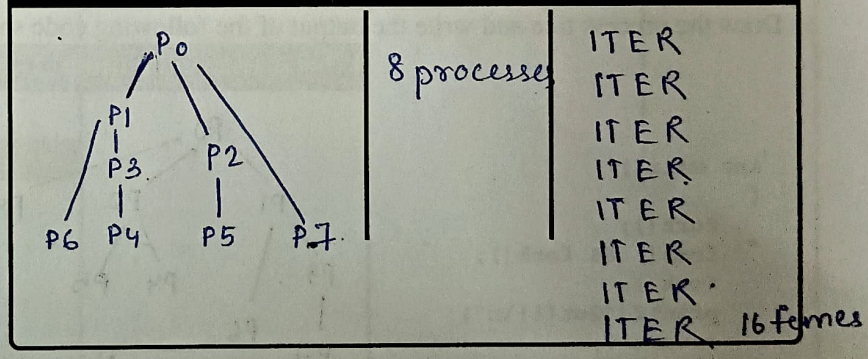
Students will be able to differentiate programs and processes, also able to learn how to create processes and able to explore the implication of process inheritance.

Programming/ Output Based Questions:

1. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int main(void) {
    fork();
    fork();
    fork();
    printf("ITER\n");
    printf("ITER\n");
    return 0;
}
/* Any formula can be
    devised for number
     of processes
    created here?
If so, state.*/
```

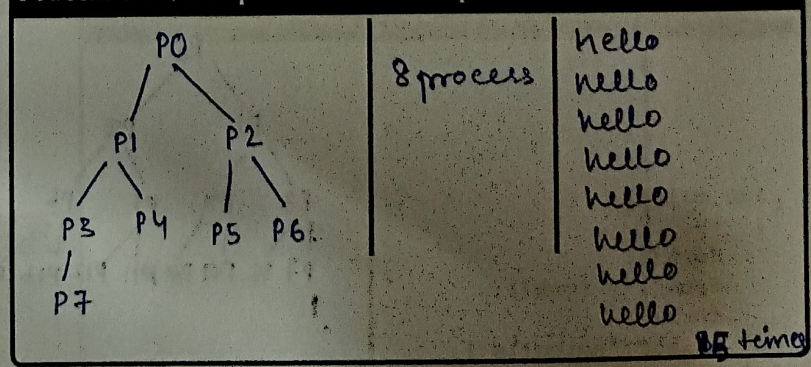
Process tree, # of processes and output



2. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int main(void){
  printf("hello\n");
  fork();
  printf("hello\n");
  fork();
  printf("hello\n");
  fork();
  printf("hello\n");
  return 0;
}
/* Any formula for
   nuber of outputs?
If so, state.*/
```

Process tree, # of processes and output



3. Run the following code on your machine and write the output. Suggest a way to avoid the mismatch of machine output w.r.t. dry run output.

```
#include<stdio.h>
#include<unistd.h>
int main(void)
{
   printf("A");
   fork();
   printf("P\n");
   return 0;
}
```

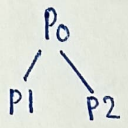
Output, Reason and Suggestion

AP
AP
	Deadpool buffer duplication occuring fork()	Explicitly flush the buffer before calling fork();
		printf("A");
		fflush(stdout);
		fork();
		printf("P\n");

4. Draw the process tree and write the output of the following code snippet.

```
int main()
{
   fork() && fork();
   printf("Able to\n");
   return 0;
}
```

Process tree and output

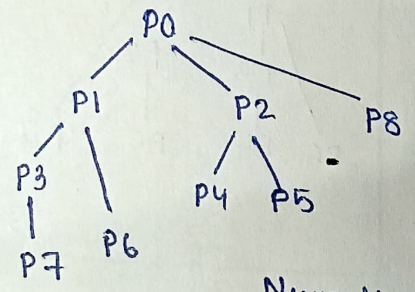


Able to
Able to
Able to

5. Draw the process tree and write the output of the following code snippet.

```
int main()
{
   fork();
   fork() && fork();
   fork();
   printf("Got!!!\n");
   return 0;
}
```

Process tree and output



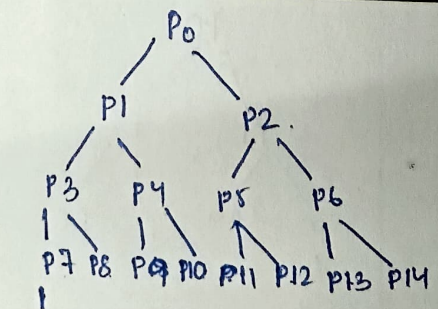
Got!!!
Got!!!
Got!!!
Got!!!
(10 times output)

Num of process = 9.

6. Draw the process tree and write the output of the following code snippet.

```
int main()
{
   fork();
   fork() + fork();
   fork();
   printf("doing!\n");
   return 0;
}
```

Process tree and output

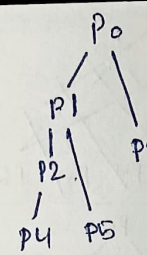


doing
doing
doing
doing
doing
doing
doing
doing
(16 lines).

7. Draw the process tree and write the output of the following code snippet.

```
int main(){
    fork();
    fork() || fork();
    fork();
    printf("Really!!!\n");
    return 0;
}
```

Process tree



Processes - 6.

Output

Really !!!
Really !!!
Really !!!
Really !!!
Really !!!
Really !!!

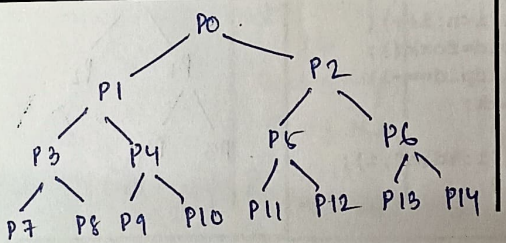
Remark

The fork() || fork() condition creates additional branches in the process tree, The logical OR (||) ensures only fork execute in some processes but the output remains the same as all process execute the printf

8. Draw the process tree and write the output of the following code snippet.

```
int main(){
 fork();
 fork()&& fork()||fork();
 fork();
 printf("guess\n");
 return 0;
}
```

Process tree



Output

guess
guess
guess
guess
guess
guess
guess
guess
guess
guess

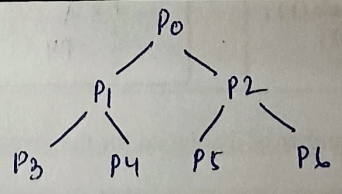
Remark

The combination of fork() creates additional branches leading to a higher no. of process All processes execute the final printf.

9. Draw the process tree and write the output of the following code snippet.

```
int main(){
 fork()&&fork();
 fork()||fork();
 printf("Hi\n");
 return 0;
}
```

Process tree



Output

Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi

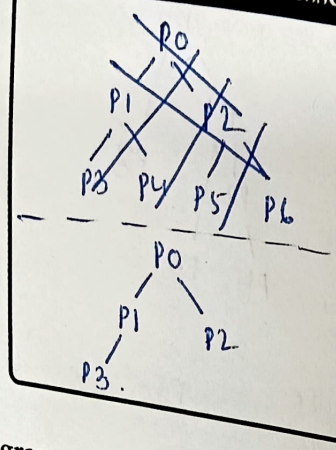
Remark

The && condition executes one fork in some branches while || ensures at least one fork is executed in other branches.

10. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int main(){
 int pid,pid2;
 pid=fork();
 if(pid){
   pid2=fork();
   printf("I\n");
 }
 else{
     printf("C\n");
     pid2=fork();
 }
 return 0;
}
```

Process tree, # of processes and output



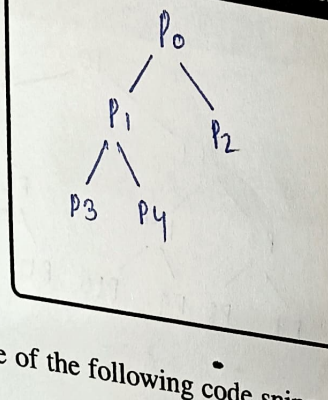
4 processes

I C I C

11. Construct the process tree diagram and also find the number of processes along with output of the following code snippet.

```
int
main(void){
 pid_t childpid;
 int i, n=3;
 for(i=1;i<n;i++){
  childpid=fork();
  if(childpid==-1)
     break;
 }
 printf("i:%d\n",i);
 return 0;
}
```

Process tree, # of processes and output



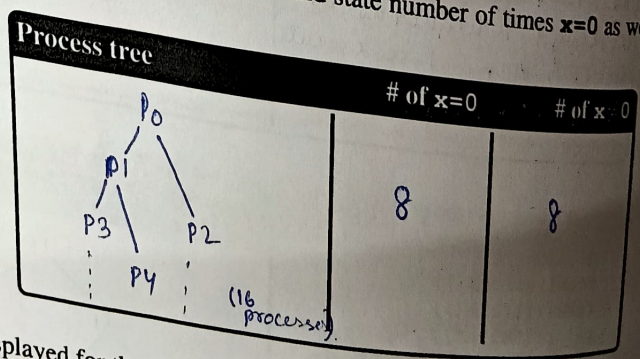
4 processes

i=1
i=2
i=3
i=3

12. Draw the process tree because of the following code snippet and state number of times **x=0** as well as **x≠0**will be displayed.

```
pid_t
add(pid_t a, pid_t b){
   return a+b;
}
int main(void){
  pid_t x=10;
  printf("%d\n",x);
  x=add(fork(),fork());
  printf("%d\n",x);
  return 0;}
```

Process tree



13. Determine the total number of displayed for the given code snippet.

```
int main(void){
 int x[]={10,20,fork(),fork()+fork()};
 int len=sizeof(x)/sizeof(int);
 for(int i=0;i<len;i++)
   fprintf(stderr," %d ",x[i]);
 printf("\n");
 return 0;
 }
```

# of display	# of Processes
No. of processes = 8	
total no. of display = 8 X 4 = 32	

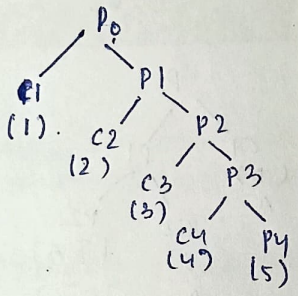
14. Determine the number of process(s) will be created when the below program becomes process and also write the output.

```
void show(){
  if(fork()==0)
   printf("1\n");
  if(fork()==0)
   printf("2\n");
 if(fork()==0)
   printf("3\n");
}
int main(void){
 show();
 return 0;
}
```

# of processes	Output
No. of processes = 8. (1 parent - 7 child).	o/p: 1, 2, 2, 3, 3 (order may vary) but 8 output.

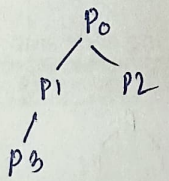
15. Draw the process tree of the following code snippet. Also give a count of processes and the output of the following code. Can the code segment generate fan of processes.

```
int main(void){
 if(fork()==0)
   printf("1\n");
 else if(fork()==0)
   printf("2\n");
 else if(fork()==0)
   printf("3\n");
 else if(fork()==0)
   printf("4\n");
 else
   printf("5\n");
 return 0;
 }
```

Process tree, # of processes and output		
	No. of processes 16 but only 5 of them produce output	1 2 3 4 5

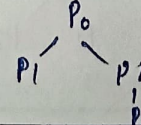
16. Find the output of the code segment showing the corresponding process tree.

```
int main(){
 pid_t p1,p2;
 p2=0;
 p1=fork();
 if (p1 == 0)
    p2 = fork();
 if (p2 > 0)
    fork();
 printf("done\n");
 return 0;
 }
```

Process tree	Output
	done done done done.

17. Find the number of direct children to the main process, the total number of processes and the output.

```
int main(){pid_t c1=1,c2=1;
 c1=fork();
 if(c1!=0)
    c2=fork();
 if(c2==0){
    fork();printf("1\n");
  }
 return 0;}
```

# of direct children to main process	Total processes	Output
2 direct children → P1, P2 	total process = 4	1 1

18. Find the output of the code segment.

```
int main(){
  struct stud s1={1,20};
  pid_t pid=fork();
  if(pid==0){
     struct stud s1={2,30};
     Printf("%d %d\n",s1.r,s1.m);
     return 0;
  }
  else{
    sleep(10);
    Printf("%d %d\n",s1.r,s1.m);
    return 0;
  }
}
```

```
struct stud{
  int r;
  int m;
};
```

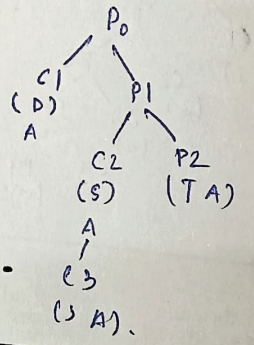
Output

2 30
1 20

19. Find the output of the code segment showing the corresponding process tree.

```
int main(){
  if(fork()){
    if(!fork()){
      fork();
      Printf("S ");
    }
    else{
      Printf("T ");
    }
  }
  else{
      Printf("D ");
  }
  Printf("A ");
  return 0;
}
```

Process tree



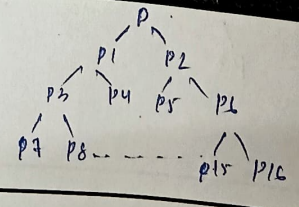
Output

DATASASA

20. Calculate the number of processes the following code snippet will generate.

```
int main(){int i;
  for(i=0;i<12;i++){
     if(i%3==0){
          fork();
     }
  }
  return 0;
}
```

Process tree



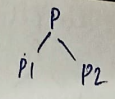
Output

None.

21. State the possible values of x for the given code snippet;

```
int x;
int a[2]={10,20};
x=5+a[fork() || fork()];
Printf("%d ",x);
```

Process tree



Output

15 25 15 25

22. Suppose four user-defined exit handlers X, Y, P, and Q are installed in the order X then Y then P then Q using atexit() function in a C program. Exit hadler X is designed to display 1, Y is designed to display 2, P is designed to display 3, and Q to display 4. State the order of their display, when the program is going to terminate after calling return 0/exit(0).

(A) 4, 3, 2 1 (C) 1, 2, 4, 3

(B) 1,2,3,4 (D) none

Choice: 4 3 2 1

23. You know that the **ps** utility in UNIX reports a snapshot of the current processes. Determine the state code of the given program, that became a process.

```
int main(void){
 fprintf(stderr,"PID=%ld\n",(long)getpid());
 while(1);
 return 0;
}
```

(A) R (C) T

(B) S (D) Z

Choice: R

24. Find the process state code of the given program, that became a process using the Unix utility **ps**. As you know **ps** displays information about a selection of the active processes.

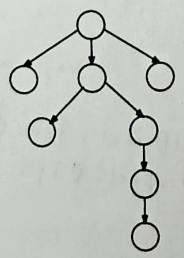
```
int main(void){
 fprintf(stderr,"PID=%ld\n",(long)getpid());
 while(1)
     sleep(1);
 return 0;
}
```

(A) R (C) T

(B) S (D) Z

Choice: S.

25. Develop a C code to create the following process tree. Display the process ID, parent ID and return value of `fork()` for each process.



OBSERVARIONS:

- Use ps utility to verify the is-a-parent relationship?
- Are ypu gettig any orphan process case?
- Are you getting any ZOMBIE case?

Figure 1: Process tree

Code here

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main(){
  pid_t pid1, pid2;
  printf("Parent Process: PID=%d, PPID=%d\n", getpid(), getppid());
// first fork
pid1 = fork();
if (pid1<0){
  perror("First fork failed");
  exit(1);
}
else if (pid1==0){
  printf("Child1: PID %d, PPID=%d, fork() return=%d\n", getpid(),
                                        getppid(), pid1);
  pid2 = fork();
  if (pid2<0){
    perror("second fork failed in child1");
    exit(1);
  }
  else if (pid2==0){
    printf("PID=%d, PPID=%d, fork return=%d\n", getpid(),
          getppid(), pid2);
  }
  else{
    wait(NULL);
  }
  exit(0);
  pid2 = fork();
  if(pid2<0){
    exit(1);
  }else if(pid2==0){
    printf("Child 2: PID=%d, PPID %d, fork() return=%d\n",
          getpid(), getppid(), pid2);
  }else{
    wait(NULL);
    wait(NULL);
  }
}
return 0;
}
```

26. Create two different user-defined functions to generate the following process hierarchy shown in **Figure-(a)** and **Figure-(b)**. Finally all the processes display their process ID and parent ID.

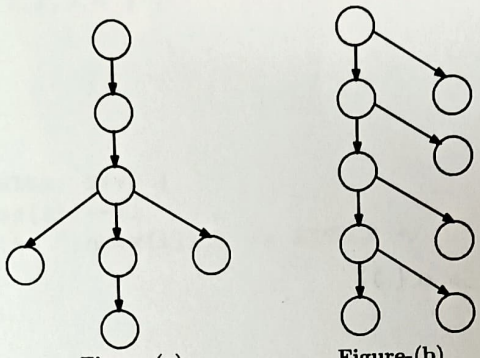


Figure 2: Process tree

Code here

(a)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
void create_fegure_a() {
    pid_t P1, P2, P3, P4;
    printf("PID = %d, PPID = %d \n", getpid(), getppid());
    P1 = fork();
    if (P1 == 0) {
        printf("PID = %d, PPID = %d \n", getpid(), getppid());
        exit(0);
    }
    else {
        P2 = fork();
        if (P2 == 0) {
            printf("PID = %d, PPID = %d \n", getpid(), getppid());
        } P3 = fork();
        if (P3 == 0) {
            printf("PID = %d, PPID = %d \n", getpid(), getppid());
        } exit(0);
        P4 = fork();
        if (P4 == 0) {
            printf("PID = %d, PPID = %d \n", getpid(), getppid());
        } exit(0);
```

Code here

```
    wait (NULL);
    wait (NULL);
    exit(0);
}
wait (NULL);
wait(NULL);
}}
int main (){
    create-figure-a();
    return 0;
}

void create-figure-b (){
    pid-t P1, P2, P3, P4;
    printf ("PID = %d, PPID = %d\n", getpid(), getppid());
    P1 = fork();
    if (p1 == 0){
        print ("PID = %d, PPID = %d \n", getpid(), getpid());
        P3 = fork().
    }if (p3 == 0){
        printf ("PID = %d, PPID = %d \n, getpid(), getppid();
        exit(0);
    }.
    wait (NULLL).
    exit(0)
} else {
    P2 = fork();
    if (p2 == 0) {
        printf ("PID = %d, "PPID = %d \n" getpid(), get ppid();
        exit(0).;
    }wait (NLL).
    wait (NULL);
}
}int main (){
    create figure-b();
    return 0;
}
```

27. What output will be at Line X and Line Y?

```
#define SIZE 5
int nums[SIZE] = { 0,1,2,3,4 } ;
int main(){
 int i;
 pid t pid;
 pid = fork();
 if(pid == 0){
    for (i = 0; i < SIZE; i++) {
       nums[i] *= nums[i] *-i;
       printf("CHILD:%d ",nums[i]);   /* LINE X */
    }
 }
 else if (pid > 0) {
    wait(NULL);
    for (i = 0; i < SIZE; i++)
       printf("PARENT: %d ",nums[i]);   /* LINE Y */
    }
  return 0;
}
```

Output

CHILD : 0 CHILD : 1
CHILD : 2 CHILD : 9
CHILD : 16
PARENT : 0 PARENT : 1
PARENT : 2 PARENT : 3
~~PARE~~ PARENT : 4

28. The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8,

Write a C program using the fork() system call that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child.

Code here

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
void generate_fibonacci(int n){
    int a=0, b=1, c;
    printf("Child process(PID:%d): Fibonacci sequence:\n", getpid());
    for(int i=0; i<n; i++){
        if(i==0)
            printf("%d ", a);
        else if(i==1)
            printf("%d ", b);
        else{
            c=a+b;
            printf("%d ", c);
            a=b;
            b=c;
        }
    }
    printf("\n");
}
int main(int argc, char *argv[]){
    if(argc!=2){
        fprintf(stderr, "Usage: %s \n", argv[0]);
        exit(1);
    }
    int n=atoi(argv[1]);
    if(n<=0){
        fprintf(stderr, "Please provide a +ve int\n");
        exit(1);
    }
    pid_t pid=fork();
    if(pid<0){
        perror("fork failed");
    }
    if(pid==0){
        generate_fibonacci(n);
    }
    else{
        wait(NULL);
        printf("Parent PID: %d. Child process completed\n", getpid());
    }
    return 0;
}
```

29. Write a **`MulThree.c`** program to multiply three numbers and display the output. Now write another C program **`PracticeExecl.c`**, which will fork a child process and the child process will execute the file **`MulThree.c`** and generate the output. The parent process will wait till the termination of the child and the parent process will print the process ID and exit status of the child.

Code here

MULThree.c

```
#include <stdio.h>
#include <stdlib.h>
int main (int arege , char *argv[]) {
   if (arege != 4){
      fprintf (stderr, "Usage : %s <num1> <num2> <num3> \n ", arg[0]);
   } return 1;
   int num1 = atoi(argv[1]);
   int num2 = atoi(argv[2]);
   int num3 = atoi(argv[3]);
   int r = num1 * num2 * num3;
   printf ("The result of %d %d %d is : %d\n", num1, num2, num3, r);
} return 0;
```

PracticeExecl.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
int main(){
  pid_t pid = fork()
   if (pid == 0){
      execl ("./MulThree.", "MulThree", "2", "3", "4", NULL);
      perror ("execl failed");
   } exit(0);
   else if (pid > 0){
      int status
      wait (&status);
      if(WIFEXITED(status)){
         printf("child process exited with status %d\n", WEXITSTATUS
                                                          (status))
   }
   } else{
      perror ("fork failed");
      return 1;
   }
   return 0;
}
```

30. You know the usages of the command **grep**. Implement the working of **grep -n pattern filename** in a child process forked from the parent process using **execl** system call. The parent process will wait till the termination of the child and the parent process will print the process ID and exit status of the child.

Code here

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[]){
  if (argc != 3){
    fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
    exit(1);
  }
  char *pattern = argv[1];
  char *filename = argv[2];
  pid_t pid = fork();
  if(pid == 0){
    printf ("child process : %d : Executing grep -n %s :..\n", getpid(),
                                        pattern, filename);
    execl ("/bin/grep", "grep", "-n", pattern, filename, NULL);
    perror ("execl failed");
    exit(1);
  }
  else{
    int status;
    waitpid (pid, &status, 0);
    if (WIFEXITED (status)) {
      printf ("Parent process : %d : Child process : %d exited with
                status : %d\n", getpid(), pid, WEXITSTATUS
                                                (status));
    }else {
      printf("Parent process : Child process donot
                terminate normally \n");
    }
  return 0;
}
```

31. Implement the above question using **execv**, **execlp**, **execvp**, **execle**, **execve** system calls.

Code here for execv

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
int main(int argc, char *argv[]){
  if (argc != 3) {
    fprintf(stderr, "Usage : %s <pattern> <filename> \n", argv[0]);
    exit(0);
  }
  char *pattern = argv[1];
  char *filename = argv[2];
  pid_t pid = fork();
  if (pid < 0){
    perror("Fork failed");
    exit(1);
  }
  if(pid == 0){
    char *args[] = {"grep", "-n", pattern, filename, NULL};
    printf("Child process : %d : Executing grep with exec \n",
                                        getpid());
    execv("/bin/grep", args);
  } perror("execv failed");
    exit(1);
  }
  else {
    int status;
    waitpid(pid, &status, 0);
    printf("Parent Process : %d, Child Process : %d,
        exited with status %d \n", getpid(), pid,
                                        WEXITSTATUS(status));
  }
  return 0;
}
```

Code here for execlp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[]){
    if (argc != 3)
     fprintf (stderr, "Usage : %s <pattern> <filename>\n", argv[0]);
     exit(1);
    }
    char *pattern = argv[1];
    char *filename= argv[2];
     pid_t pid = fork();
     if (pid<0){
        perror ("fork failed");
        exit(1);
      }
      if (pid==0){
        printf ("Child Process : %d : Executing grep with execlp\n",
                                                    getpid());
        execlp ("grep","grep","-n",pattern,filename,NULL);
        perror ("execlp failed");
        exit(1);
      }
      else {
        int status;
        waitpid (pid,&status,0);
        printf ("parent process %d : Child process %d
                exited with status %d \n", getpid(), pid,
                                        WEXITSTATUS(status));
      }
      return 0;
}
```

Code here for execvp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
        exit(1);
    }
    char *pattern = argv[1];
    char *filename = argv[2];
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork failed");
        exit(1);
    }
    if (pid == 0) {
        char *args[] = {"grep", "-n", pattern, filename, NULL};
        printf("Child Process (PID: %d): Execute grep with execvp\n",
                                                    getpid());
        execvp("grep", args);

        perror("execvp failed");

        exit(1);
    }
    else {
        int status;
        waitpid(pid, &status, 0);
        printf("Parent Process (PID: %d) exited with status: %d\n",
                                getpid(), pid, WEXITSTATUS(status));
    }
    return 0;
}
```

Code here for execve

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[]){
  if (argc != 3) {
    fprintf (stderr, "Usage: %s <pattern> <filename> \n", argv[0]);
    exit (1);
  }
  char *pattern = argv[1];
  char *filename = argv[2];
  pid_t pid = fork ();
  if (pid < 0) {
    perror ("fork failed");
    exit (1);
  }
  if (pid == 0) {
    char *env[] = {NULL}
    printf ("Child Process (PID: %d): with execle. \n", getpid ());
    execle ("/bin/grep", "grep", "-n", pattern, filename, NULL, env);
    perror ("execle failed");
    exit (0);
  }else {
      int status;
      waitpid (pid, &status, 0);
      printf ("Parent Process (PID: %d): Child process (PID: %d)
                  exited with status: %d \n", getpid (), pid,
                    WEXITSTATUS (status));
  }
  return 0;
}
```

Code here for execle

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[]){
    if (argc != 3){
        fprintf (stderr, "Usage : %s <pattern> <filename> \n", argv[0]);
        exit(1);
    }
    char *pattern = argv[1];
    char * filename = argv[2];
    pid_t pid = fork();
    if (pid < 0){
        perror("Fork failed");
        exit(1);
    }
    if (pid == 0){
        char *args[] = {"grep", "-n", pattern, filename, NULL};
        char *env[] = {NULL};
        printf ("Child Process (PID : %d): with execve \n", getpid());
        execve ("/bin/grep", args, env);
        perror ("execve failed");
        exit(1);
    }
    else {
        int status;
        wait(pid, &status, 0);
        printf ("Parent process (PID: %d): Child process (PID: %d)
                exited with status : %d \n", getpid(),
                pid, WEXITSTATUS (status));
    }
    return 0;
}
```