1)

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

void forkexample()
{
	int x = 1;

	if (fork() == 0)
		printf("Child has x = %d\n", ++x);
	else
		printf("Parent has x = %d\n", --x);
}
int main()
{
	forkexample();
	return 0;
}
```

2)

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


int main()
{
	pid_t pid;
	pid=fork();		/* fork a child process*/

if(pid<0)		/*error occurred*/
{
		fprintf(stderr, “fork failed”)
		exit(-1);
}


else if(pid==0)		/* child process*/
{
	execlp(“/bin/ls”, “ls”, NULL)
}
else
{			/*parent process*/
	wait(NULL)	/* parent will wait for the child to complete*/
	printf(“child completed”);
exit(0)	;
}
}
```

3)

```
#include <unistd.h>
#include  <stdio.h>
#include  <sys/types.h>
#define   MAX_COUNT  5
void  ChildProcess(void);              /* child process prototype  */
void  ParentProcess(void);             /* parent process prototype */

int  main(void)
{
    pid_t  pid;

    pid = fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}
void  ChildProcess(void)
{
    int   i;
    for (i = 1; i <= MAX_COUNT; i++)
    printf("   This line is from child, value = %d\n", i);
    printf("   *** Child process is done ***\n");
}
void  ParentProcess(void)
{
      int   i;
      for (i = 1; i <= MAX_COUNT; i++)
        printf("This line is from parent, value = %d\n", i);
        printf("*** Parent is done ***\n");
}
```

4)

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

void main(int argc,char *arg[])
{
int pid; pid=fork(); if(pid<0)
{
printf("fork failed"); exit(1);
}
else if(pid==0)
{
execlp("whoami","ls",NULL); exit(0);
}
else
{
printf("\n Process id is -%d\n",getpid()); wait(NULL);
exit(0);
}
}
}
```