ii. wait ( )

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax: wait (NULL);

iii. exit ( )

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax: exit (0);

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the variables pid and i as integers.

Step 3: Get the child id value using the system call fork ().

Step 4: If child id value is less than zero then print "fork failed".

Step 5: Else if child id value is equal to zero, it is the id value of the child and then start the child process to execute and perform Steps 7 & 8.

Step 6: Else perform Step 9.

Step 7: Use a for loop for almost five child processes to be called.

Step 8: After execution of the for loop then print "child process ends".

Step 9: Execute the system call wait ( ) to make the parent to wait for the child process to get over.

Step 10: Once the child processes are terminated, the parent terminates and hence prints "Parent process ends".

Step 11: After both the parent and the child processes get terminated it execute the wait ( ) system call to permanently get deleted from the OS.

Step 12: Stop the program.

PROGRAM:

1.B.1) SOURCE CODE:

```
#include<stdio.h>
#include<unistd.h>
int main( )
{
        int i, pid;
        pid=fork( );
        if(pid== -1)
        {
                printf("fork failed");
                exit(0);
        }
        else if(pid==0)
        {
                printf("\n Child process starts");
                for(i=0; i<5; i++)
                {
                        printf("\n Child process %d is called", i);
                }
                printf("\n Child process ends");
        }
        else
        {
                wait(0);
                printf("\n Parent process ends");
        }
        exit(0);
}
```