Write a program in Linux to use clone system call and show how itis different from fork system call.

**Objectives:**

1. To learn about threading in Linux/Unix and Java and difference between them
2. Use of system call/library to write effective programs

**Theory:**

Syntax:

#define \_GNU\_SOURCE

#include <sched.h>

int clone(int (\**fn*)(void \*), void \**child\_stack*,

int *flags*, void \**arg*, ...

/\* pid\_t \**ptid*, void \**newtls*, pid\_t \**ctid* \*/ );

/\* For the prototype of the raw system call, see NOTES \*/

Description:

clone() creates a new process, in a manner similar to [fork(2).](http://man7.org/linux/man-pages/man2/fork.2.html)

This page describes both the glibc clone() wrapper function and the underlying system call on which it is based. The main text describes the wrapper function; the differences for the raw system call are described toward the end of this page.

Difference between fork and clone:

Unlike [fork(2),](http://man7.org/linux/man-pages/man2/fork.2.html) clone() allows the child process to share parts of its execution context with the calling process, such as the virtual address space, the table of file descriptors, and the table of signal handlers. (Note that on this manual page, "calling process" normally corresponds to "parent process". But see the description of

CLONE\_PARENT below.)

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
|  |  |  |  |
| 1 | fn | Int (function) | Run code under child process. |
|  |  |  |  |
| 2 | pchild\_stack | void\* | Allocating memory. |
|  |  |  |  |
| 3 | pid | int | ID of cloned process. |
|  |  |  |  |

**Program:**

#include <stdio.h>

#include <sched.h>

#include <stdlib.h>

#include <sys/wait.h>

int fn()

{

printf("\nThis code is running under child process.\n\n");

int i = 0;

int n = 7;

for ( i = 1 ; i <= 8 ; i++ )

{

printf("%d \* %d = %d\n", n, i, (n\*i));

}

return 0;

}

void main(int argc, char \*argv[])

{

printf("We are in Parent\n\n");

void \*pchild\_stack = malloc(1024 \* 1024);

if ( pchild\_stack == NULL )

{

printf("ERROR: Unable to allocate memory.\n");

exit(EXIT\_FAILURE);

}

printf("Creating Child\n");

int pid = clone(fn, pchild\_stack + (1024 \* 1024), SIGCHLD, argv[1]);

if ( pid < 0 )

{

printf("ERROR: Unable to create the child process.\n");

exit(EXIT\_FAILURE);

}

wait(NULL);

free(pchild\_stack);

printf("\nChild process terminated.\n");

}

**Output:**

it@it-OptiPlex-3020:~/Downloads$ gcc D4.c

it@it-OptiPlex-3020:~/Downloads$ ./a.out

We are in Parent

Creating Child

This code is running under child process.

7 \* 1 = 7

7 \* 2 = 14

7 \* 3 = 21

7 \* 4 = 28

7 \* 5 = 35

7 \* 6 = 42

7 \* 7 = 49

7 \* 8 = 56

Child process terminated.

**Conclusion:**

clone() can be used to produce a child which could share some information of its parent and can be further modified to have its own form.

**References:** https://www.geeksforgeeks.org/clone-method-in-java-2/