**Thread concept: clone, threads of java**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 4(d)**

**Title-** Write a program in Linux to use clone system call and show how it is 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:**

In Linux, both fork() and clone() are system calls that are used to create a new process, but there are important differences between the two in terms of functionality and flexibility.

* **fork()**: The fork() system call creates a new process by duplicating the calling process. The new process is a copy of the parent process, but the child process gets its own address space, file descriptors, and other resources. The fork() system call is the simplest way to create a new process in Linux. It is a "heavyweight" process creation, as the child process gets a full copy of the parent's state.
* **clone()**: The clone() system call is more flexible and fine-grained. It allows the calling process to specify which resources (such as address space, file descriptors, etc.) should be shared between the parent and child. This means that clone() can be used to create threads, containers, or processes with more customized behaviour compared to fork(). It allows the caller to control the behaviour of the child process more explicitly.

**Program:**

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sched.h>

#define STACK\_SIZE 1024\*1024  // 1MB stack size for cloned thread

// Function to be executed by the cloned thread

int child\_function(void \*arg) {

    printf("Child (clone) PID: %d, Parent PID: %d\n", getpid(), getppid());

    return 0;

}

int main() {

    printf("Parent PID: %d\n", getpid());

    // Using fork()

    pid\_t fork\_pid = fork();

    if (fork\_pid == -1) {

        perror("fork failed");

        exit(1);

    } else if (fork\_pid == 0) {

        // Child process (fork)

        printf("Child (fork) PID: %d, Parent PID: %d\n", getpid(), getppid());

        exit(0);

    }

    wait(NULL);  // Wait for forked process to finish

    // Using clone()

    void \*stack = malloc(STACK\_SIZE);

    if (!stack) {

        perror("malloc failed");

        exit(1);

    }

    pid\_t clone\_pid = clone(child\_function, stack + STACK\_SIZE, SIGCHLD, NULL);

    if (clone\_pid == -1) {

        perror("clone failed");

        free(stack);

        exit(1);

    }

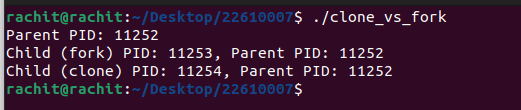
    wait(NULL);  // Wait for cloned process to finish

    free(stack);

    return 0;

}

**Output:**

****

**Conclusion:**

In conclusion, while both fork() and clone() can create new processes in Linux, clone() provides far more flexibility and control, allowing for the creation of processes or threads with customized resource sharing. fork() is easier to use and is suited for situations where a completely independent child process is required. clone(), on the other hand, is used for more complex process creation, such as threading and containerization.