Implementing clone() system call

System Call on Guix OS

A **system call** is a mechanism that allows user-level applications to request services from the kernel of the operating system. Since user-space programs cannot directly access hardware or perform privileged operations, they use system calls to interact with resources such as files, memory, and processes. When a system call is invoked, the CPU switches from user mode to kernel mode, allowing the requested operation to be executed securely. Once completed, control is returned to the user program.

Now I am going to demonstrate how to implement and use the clone() system call on Guix OS. The clone() system call is used in Unix-like operating systems to create a new process. It is similar to fork(), but clone() allows more fine-grained control over what is shared between the parent and the child processes, such as memory space, file descriptors, and signal handlers.

To implement and test the clone() system call on Guix OS (which is based on the GNU operating system), follow the steps below:

Install Required Development Tools

First, I will open a Terminal and install essential C development tools.

```
guix install gcc make glibc
```

This command ensures the presence of gcc, make, and the C library headers, which are necessary for compiling and linking C programs.

Create the Source File

Next, I am going to write a C source file to test the clone() system call. I will Use a text editor called nano.

nano clone_test.c

The clone() system call is used to create a new process or thread in Linux.

```
#define _GNU_SOURCE

#include <sched.h>
```

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

int childFunc(void *arg) {
    printf("Hello from child process\n");
    return 0;
}

int main() {
    const int STACK_SIZE = 1024*1024;
    char *stack = malloc(STACK_SIZE);
    if (!stack) {
        perror("malloc");
        exit(1);
    }

    int pid = clone(childFunc, stack + STACK_SIZE, SIGCHLD, NULL);
    if (pid == -1) {
        perror("clone");
        exit(1);
    }
```

```
    printf("Hello from parent process\n");
    wait(NULL);
    return 0;
}
```

Finally I Compile the code using the following command

```
gcc -o clone_example clone_example.c -Wall
```

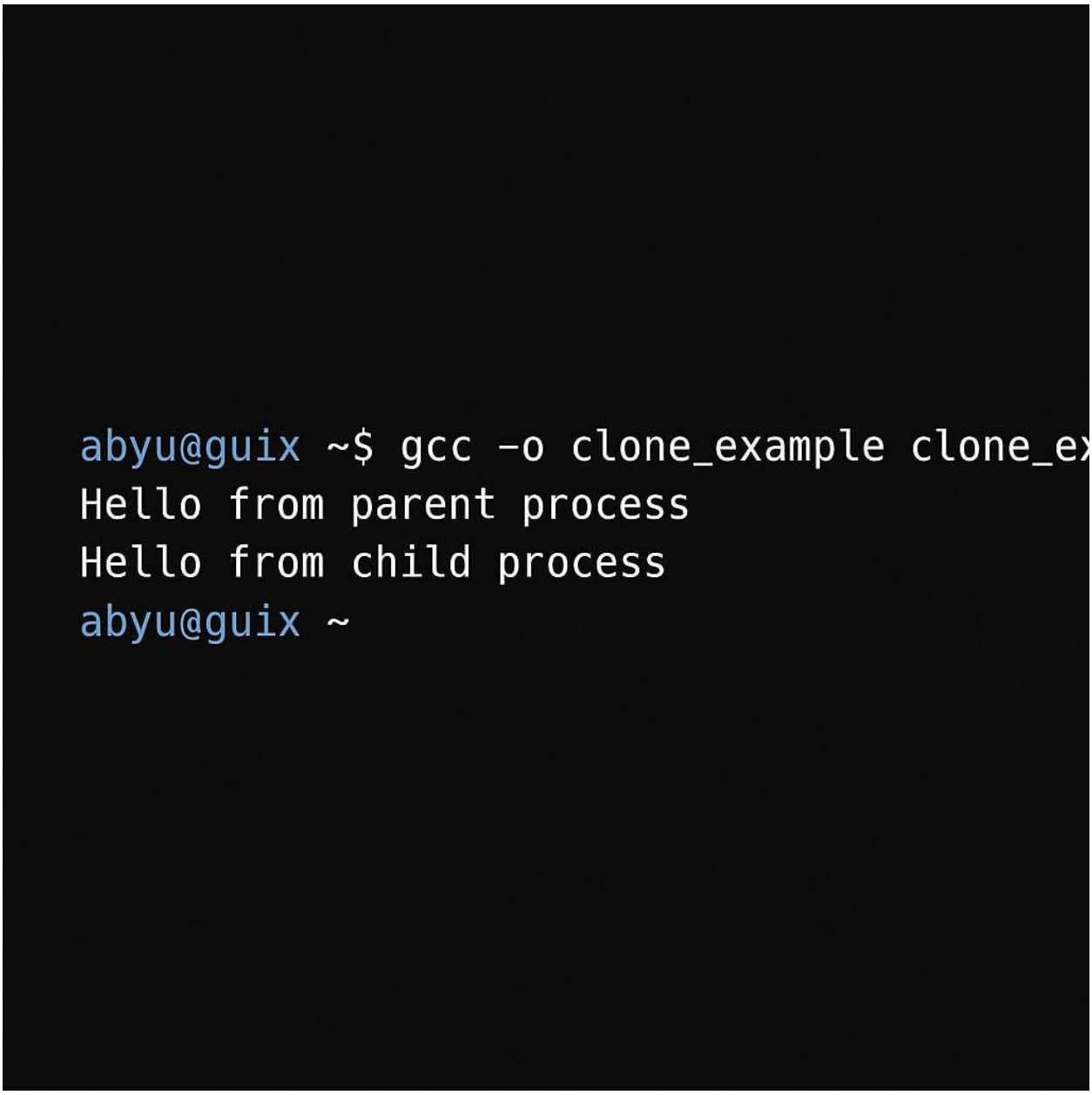
I will use the command `./clone_test` to run the code.

The result will be:

Hello from parent process

Hello from Child process

This the expected out put of the above c code.



What This Output Means

1. **Parent process message**:
    - This line is printed **after the clone() call** returns in the parent process.
    - getpid() gives the parent's PID.
    - The return value of clone() is the child's PID (if successful), so the parent knows the child's identity.
2. **Child process message**:
    - This line is printed from inside the child_function() passed to clone().

- It runs in the newly created process.
- getpid() here gives the child's PID.