Bahir Dar University

Faculty of Computing

Department of Software Engineering

OSSP Individual Assignment

System Call Implementation - `execve()`

Full Name: Elias Zeleke
Student ID: BDU1601353
Section: B
System Call: execve ( )
Submitted To: Lecturer Wendimu Baye
Submission Date: 16/08/2017 E.C

System Call Implementation

❖ execve() in User Space

The execve ( ) system call in C allows a program to execute a new process by replacing the current one. In user space, this system call is accessed via a glibc wrapper, which internally uses the syscall ( ) function to request the kernel's implementation of execve.

This wrapper handles communication with the kernel and passes the necessary arguments: the path to the executable, the arguments for the new program, and the environment variables.

User-Space Code:

```
#include <unistd.h>

#include <sys/syscall.h>

#include <errno.h>

int execve(const char *pathname, char *const argv[], char *const envp[]) {

return syscall(SYS_execve, pathname, argv, envp);

}
```

this code defines how the high-level execve() call in C is translated into a low-level system call using syscall(). The constant SYS_execve is defined in system headers and maps to the correct syscall number for your platform.

❖ Kernel Space: Actual System Call Logic

In the Linux kernel, the execve() system call is defined inside the fs/exec.c file. The syscall is implemented and registered using the SYSCALL_DEFINE3 macro, which specifies that execve() takes three arguments.

Kernel-Level Code:

```
SYSCALL_DEFINE3(execve,
   const char __user *, filename,
   const char __user *const __user *, argv,
   const char __user *const __user *, envp)
{
   return do_execve(getname(filename), argv, envp);
}
```

Key Steps in Kernel Logic:

- getname(filename): Validates and retrieves the binary's filename from user space.
- do_execve ( ) : Responsible for loading the new binary, setting up memory, clearing the current process image, and initializing the new one.
- If successful, the calling process image is destroyed and replaced by the new program.
- If an error occurs, it returns an error code.

This syscall does not return on success — the process context is changed, and the new program takes control.

❖ How execve() Works

The execve() system call is unique because it replaces the currently running program. Unlike fork(), which creates a new child process, execve() completely overwrites the current process image with a new executable.

Key Behavior:

- It does not return if the new program loads successfully.
- If execution fails (e.g., due to a missing binary or permission error), execve() returns -1 and sets the errno variable.
- All previous memory content, file descriptors (unless marked O_CLOEXEC), and signal handlers are removed.

This makes execve() critical for creating shells, script runners, or OS-level tools that manage programs.

❖ execve() Example Code and Output

execdemo.c - A Program That Executes ls -l /home

This example demonstrates the use of execve() to run the ls command on the /home directory. It replaces the current process with a new one pointing to /bin/ls.

```
#include <unistd.h>

#include <stdio.h>

int main() {

   char *program = "/bin/ls";

   char *args[] = { "ls", "-l", "/home", NULL };

   execve(program, args, NULL);

   // If we reach here, execve has failed

   perror("execve failed");

   return 1;

}
```

Compile and Run:

```
gcc execdemo.c -o execdemo
./execdemo
```

Expected Output:

If /home contains files or folders, the output will look like:

```
/home:
total 8
drwxr-xr-x 2 root root 4096 Apr 24 09:59 user1
-rw-r--r-- 1 root root   10 Apr 24 10:00 notes.txt
```

If the binary or path is invalid, the error handler prints:

```
execve failed: No such file or directory
```

❖ hello.c – A Basic Output Program for Testing GCC

This is a simple program used to verify that your compiler and execution environment (e.g., Tiny Core Linux) are working correctly. It prints a static message to the terminal.

```
#include <stdio.h>

int main() {

printf("This is Operating assignment at Bahir Dar University.\n");

return 0;
}
```

Compile and Run:

```
gcc hello.c -o hello

./hello
```

Output:

```
This is Operating assignment at Bahir Dar University.
```

❖Under the Hood: Execution Layers

Layer	Component	**Role**
User Space	`execve()` (glibc)	Wrapper function that calls syscall()
Syscall Entry	`execve()` (glibc)	Registers syscall and passes args to kernel logic
Kernel Logic	do_execve()	Loads new executable, maps memory, replaces process image
File System	search_binary_handler()	Locates and validates the binary (e.g., ELF or script)

This layered architecture ensures that the request to run a new program safely flows from user space into the kernel, all the way to the file esystem.

❖ Summary

execve() is a low-level Linux system call used to load and run new programs.

Unlike fork(), which creates a child process, execve() replaces the current process entirely.

It is essential for building shells, scripting environments, and boot-time process initialization.

This assignment demonstrated both the code-level implementation of execve() and its underlying kernel logic using simple, real-world examples.

Tiny Core Linux was used as the platform due to its lightweight, modular nature — making it ideal for system-level programming exploration.

...Thank you...