Exec Family — Quick Notes (Linux/Unix)

The **exec family** of functions replaces the current running process image with a *new* program. After a successful exec call:

- The **current process ID remains the same**
- Previous code, data, stack, heap are **discarded**
- The new program starts execution from its `main()`
- `exec*()` **does not return** on success (only on failure)

Typical use-case: child process created using `fork()` calls `exec()` to run another program.

✔️Core Idea

```
fork()  → creates a new child process
exec()  → loads and runs a new program *inside* that process
```

🔛Variants of exec Family

Function	Args Passed As	Uses PATH?	Environment Customizable?
`execl(path, ...)`	List	❌No	Inherits
`execlp(file, ...)`	List	✅Yes	Inherits
`execle(path, ..., envp)`	List	❌No	✅Yes
`execv(path, argv[])`	Array	❌No	Inherits
`execvp(file, argv[])`	Array	✅Yes	Inherits
`execvpe(file, argv[], envp)`	Array	✅Yes	✅Yes

Legend: - **List** → arguments passed as variable arguments - **Array** → arguments passed as `char *argv[]` - **Uses PATH** → searches executable in `PATH`

♌Function Prototypes

```c
int execl (const char *path, const char *arg0, ..., (char *)0);
int execlp(const char *file, const char *arg0, ..., (char *)0);
int execle(const char *path, const char *arg0, ..., (char *)0, char *const
```

```
envp[]);

int execv (const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

The last argument in list-style calls must be `NULL`.

On failure: returns `-1` and sets `errno`.

⚠️Important Notes

- `exec()` **does not create a new process** → it replaces the current one
- File descriptors remain open unless marked `FD_CLOEXEC`
- Memory, variables, stack are lost after success
- Use `perror()` to debug failures

✅Example 1 — Using `execvp()` (Common & Recommended)

Runs `ls -l /` using PATH search.

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

int main() {
    char *args[] = {"ls", "-l", "/", NULL};

    printf("Before execvp\n");

    if (execvp("ls", args) == -1) {
        perror("execvp failed");
        exit(EXIT_FAILURE);
    }

    // This will run ONLY if exec fails
    printf("This will not print if exec succeeds\n");
    return 0;
}
```

Output (if success): program is replaced by `ls` output.

✅Example 2 — Using `execl()` (Full Path Required)

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

int main() {
    printf("Running /bin/date using execl...\n");

    if (execl("/bin/date", "date", "+%F %T", (char *)NULL) == -1) {
        perror("execl failed");
        exit(EXIT_FAILURE);
    }

    return 0; // executes only if execl fails
}
```

✅Example 3 — `fork() + execvp()` Pattern

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

int main() {
    pid_t pid = fork();

    if (pid == 0) { // Child
        char *args[] = {"ping", "-c", "2", "google.com", NULL};
        execvp("ping", args);
        perror("execvp failed");
        exit(EXIT_FAILURE);
    }
    else if (pid > 0) {
        printf("Parent: waiting for child...\n");
        wait(NULL);
        printf("Parent: child finished\n");
    }
```

```
    else {
        perror("fork failed");
    }
}
```

🛂 Environment-Passing Example — `execvpe()`

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

int main() {
    char *args[] = {"env", NULL};
    char *envp[] = {
        "MYVAR=Shabaz",
        "PATH=/usr/bin:/bin",
        NULL
    };

    execvpe("env", args, envp);
    perror("execvpe failed");
    return 1;
}
```

🔯 When to Use Which

- Use `execvp` → most common, uses PATH, array args
- Use `execlp` → PATH + list arguments
- Use `execv` / `execl` → when you know full path
- Use `execle` / `execvpe` → when you want custom environment

🆔 Common Errors & Fixes

Issue	Cause	Fix
`ENOENT`	File not found	Check path / PATH
`EACCES`	Permission denied	`chmod +x program`
`EINVAL`	Bad args / NULL missing	Ensure last arg is NULL
Returns to code	exec failed	Always check + `perror()`

⬜ Quick Checklist

- Ensure program exists & executable
- Last argument must be `NULL`
- After successful exec — no code runs after it
- Combine with `fork()` when needed

If you want, I can also create **PDF / DOCX / PPT / TXT** export versions of these notes for download.