perplexity

Write a program that uses the exec system call to run the “ls -l” command in the program. Running

your program should produce the same output as that produced by the “ls -l” command.
You must write three versions of your program using the following three different variants of the exec command: execl (takes arguments as a list), execlp (takes arguments as a list, and searches system PATH for the executable), and execvp (takes arguments as a vector, and searches
system PATH for the executable).
Hint: Example code to invoke these different variants of exec is given below.
2

```
execl("/bin/ls", "ls", "-l", (char ∗)NULL);
execlp("ls", "ls", "-l", (char ∗)NULL);
char *args[] = {"ls", "-l", NULL};
execvp("ls", args);
```

explanlation for this smalll one shot gunshot

Alright — here’s the **quick “one-shot” explanation** for your `exec` variants with `ls -l` so you can understand it in one go:

Goal

We want a program that **runs** `ls -l` exactly like typing it in the terminal — but from inside C code — using 3 different `exec` variants:

- `execl` → *pass full path, arguments as a variable list (arg0, arg1, ..., NULL)*
- `execlp` → *search PATH, arguments as a list*
- `execvp` → *search PATH, arguments as an array/vector*

Since `exec` **replaces** the current process, without `fork()` your program would never continue past the `exec`.
So, we **fork** each time and run one variant in the child, letting the parent continue.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
int main() {
    pid_t pid;

    // --- execl ---
    pid = fork();
    if (pid == 0) {
        printf("\n--- execl example ---\n");
        execl("/bin/ls", "ls", "-l", (char *)NULL);
        perror("execl failed");
        exit(1);
    }
    wait(NULL);

    // --- execlp ---
    pid = fork();
    if (pid == 0) {
        printf("\n--- execlp example ---\n");
        execlp("ls", "ls", "-l", (char *)NULL);
        perror("execlp failed");
        exit(1);
    }
    wait(NULL);

    // --- execvp ---
    pid = fork();
    if (pid == 0) {
        printf("\n--- execvp example ---\n");
        char *args[] = {"ls", "-l", NULL};
        execvp("ls", args);
        perror("execvp failed");
        exit(1);
    }
    wait(NULL);

    return 0;
}
```

One-shot conceptual explanation

- **All `exec*` functions replace** the current process image with a new program (in this case `ls`).
- **execl**: Give **full binary path** (`/bin/ls`) + name `"ls"` again as `argv`, followed by further args, ending with `(char *)NULL`.
- **execlp**: Give **only the program name** `"ls"`; `p` makes it search the `PATH` environment variable to find `/bin/ls`. Arguments are still passed individually.
- **execvp**: Give the program name as the first param, and a **vector/array of arguments** (`{"ls","-l",NULL}`); `p` again searches PATH.
- **Why fork()**: Because if you run an `exec` directly in `main`, the current program will be replaced and won't run later code. `fork` lets the parent continue.

- **Why wait()**: To make sure we finish one example before starting the next — avoids scrambled output.

 Shortcut to remember:

- `l` → **list** of args (arg0, arg1, arg2..., NULL)
- `v` → **vector** (array) of args
- `p` → search in **PATH**

If you want, I can also make a **Python equivalent** using `os.execl`, `os.execlp`, and `os.execvp` to see the exact same effect — would you like that?