Executable

execl, execv, execle, execve, execlp, execvp

&

fexecve

Executable

- `exec()` makes a process execute a given executable.
- Many variants exist of `exec()` system calls with different arguments.
- The exec family of functions shall replace the current process image with a new process image.
- The new image shall be constructed from a regular, executable file called the new process image file.
- There shall be no return from a successful `exec`, because the calling process image is overlaid by the new process image.
- When a C-language program is executed as a result of a call to one of the exec family of functions, it shall be entered as a C-language function call as follows:
- The arguments specified by a program with one of the exec functions shall be passed on to the new process image in the corresponding `main()` arguments.

Executable

- When a C-language program is executed as a result of a call to one of the exec family of functions, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

- where `argc` is the argument count and `argv` is an array of character pointers to the arguments themselves.

```
int main(int argc, char *argv[], char * environ[]);
```

- `extern char **environ` is initialized as a pointer to an array of character pointers to the environment strings.
- The `argv` and `environ` arrays are each terminated by a `null` pointer.
- The `null` pointer terminating the `argv` array is not counted in `argc`.

environ: user environment

• USER	The name of the logged-in user.
• HOME	A user's login directory.
• PATH	The sequence of directories, separated by colons.
• PWD	Absolute path to the current working directory.
• SHELL	The full pathname of the user’s login shell.
• EDITOR	Default editor name.

- Many more …

- A process can query, update, and delete these strings using the `getenv()`, `setenv()`, and `unsetenv()` functions, respectively.

Executable

```
int execl(const char *pathname, const char *arg0, ... /* (char *)0 */ );

int execv(const char *pathname, char *const argv[]);

int execle(const char *pathname, const char *arg0,.../* (char *)0, char *const
envp[] */ );

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

int execlp(const char *filename, const char *arg0, ... /* (char *)0 */ );

int execvp(const char *filename, char *const argv[]);

int fexecve(int fd, char *const argv[], char *const envp[]);
```

- **`l`** stands for list of arguments,
- **`v`** stands for vector (an argv[] vector),
- **`p`** stands for path and
- **`e`** stands for environment (an envp[] array).

➢ All seven return: −1 on error, no return on success.

Executable

- The first difference in these functions is that the first four (l,v,le,ve) take a pathname argument.
- The next two (lp,vp) take a filename argument, and
- The last one (`fexecve`) takes a file descriptor argument. When a filename argument is specified,
  - If filename contains a slash, it is taken as a pathname.
  - Otherwise, the executable file is searched for in the directories specified by the PATH environment variable.
- The PATH variable contains a list of directories, called path prefixes, that are separated by colons, eg.

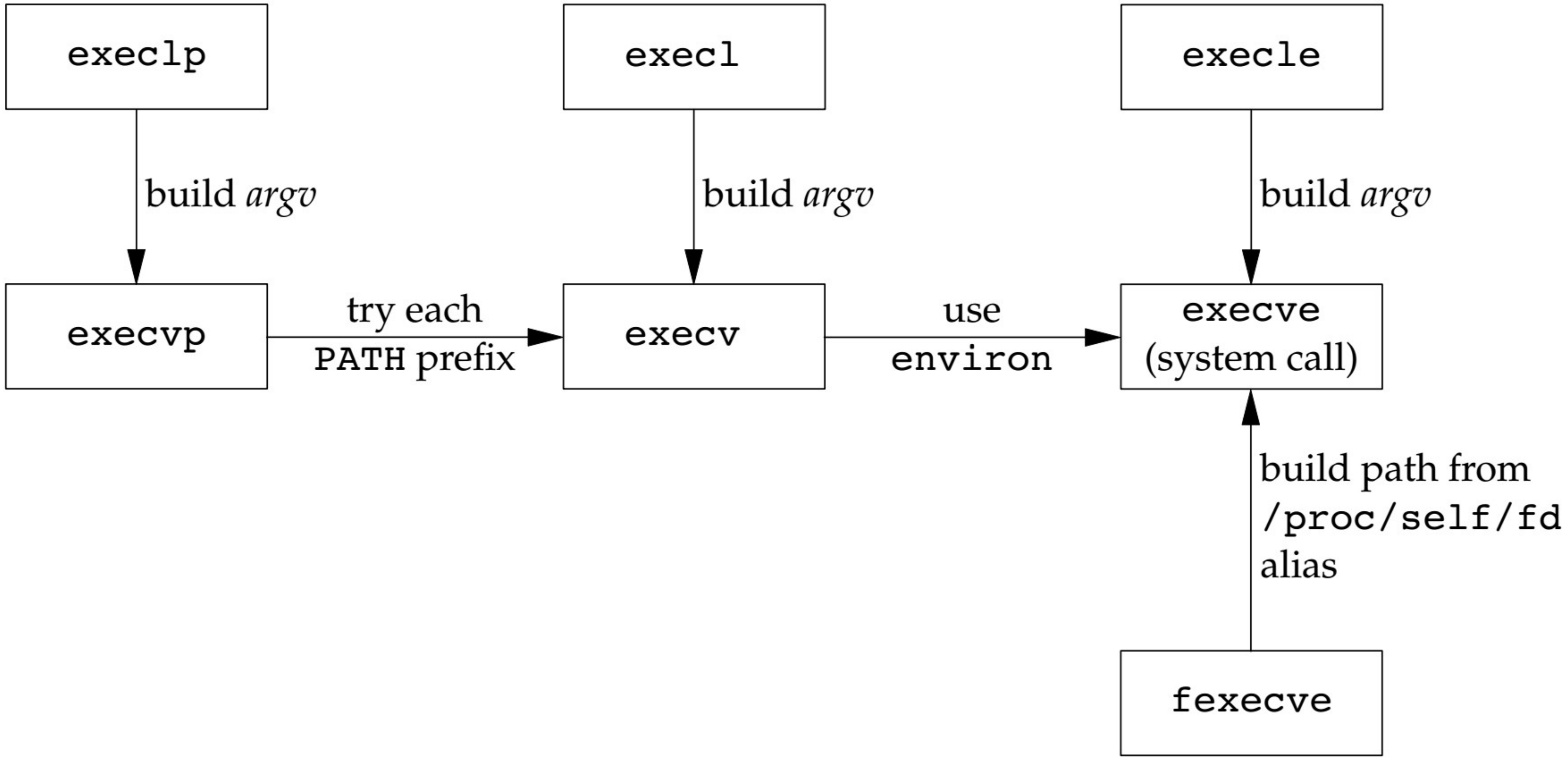
PATH=/bin:/usr/bin:/usr/local/bin/:.

Executable

- The functions `execl`, `execlp`, and `execle` require each of the command-line arguments to the new program to be specified as **separate arguments**.
- We mark the end of the arguments with a `null` pointer.
- For the other four functions (`execv`, `execvp`, `execve`, and `fexecve`), we have to build an **array of pointers** to the arguments.
- The three functions whose names end in an `e` (`execle`, `execve`, and `fexecve`) allow us to pass a pointer to an array of **pointers to the environment strings**.
- The other four functions, however, use the `environ` variable in the calling process to copy the existing environment for the new program.

Executable

- Relationship of the seven exec functions:


$$\begin{tikzcd}
\texttt{execlp} \arrow[d, "\text{build } argv"] & \texttt{execl} \arrow[d, "\text{build } argv"] & \texttt{execle} \arrow[d, "\text{build } argv"] \\
\texttt{execvp} \arrow[r, "\text{try each}", "\texttt{PATH}\text{ prefix}"'] & \texttt{execv} \arrow[r, "\text{use}", "\texttt{environ}"'] & \texttt{execve}\ \text{(system call)} \\
 & & \texttt{fexecve} \arrow[u, "\text{build path from } \texttt{/proc/self/fd} \text{ alias}"']
\end{tikzcd}$$

Example: prog.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
main(int argc, char *argv[])
{
   printf("hello (pid:%d)\n", (int) getpid());
   int id = fork();
   if (id< 0) {        // fork failed; exit
        printf("fork failed\n");
        exit(1);
   }
else if (id ==0) { // child (new process)
   printf("Child pid:%d \n", (int) getpid());
   char *myargs[3];
   myargs[0] = "prog_ other";
   myargs[1] = "prog_other";
   myargs[2] = NULL; //end of array
   execvp (myargs[0], myargs);
   printf("this shouldn't print out");
}
else { // parent goes down this path (main)
   int wc= wait (NULL);
   printf("Parent of %d (wc:%d), pid:%d\n",
id, wc, (int)getpid());
}
return 0;
}
```

Example- Executable: `echoall.c`

```
#include <sys/wait.h>
char *env[]={"USER=unknown", "PATH=/tmp",NULL};
int main(void)
{
pid_t pid;
if ((pid = fork()) < 0) {
        err_sys("fork error");
}
else if (pid == 0) {
if   (execle("../home/bin/echoall",   "echoall",
"myarg1", "MY ARG2", (char *)0, env) < 0)
        err_sys("execle error");
}
```

```
if (waitpid(pid, NULL, 0) < 0)
        err_sys("wait error");
if ((pid = fork()) < 0) {
        err_sys("fork error");
} else if (pid == 0) {
        if   (execlp("echoall",   "echoall",
"only one arg", (char *)0) < 0)
                err_sys("execlp error");
}
exit(0);
}
```

```
int
main(int argc, char *argv[])
{
    int         i;
    char        **ptr;
    extern char **environ;

    for (i = 0; i < argc; i++)      /* echo all command-line args */
        printf("argv[%d]: %s\n", i, argv[i]);

    for (ptr = environ; *ptr != 0; ptr++)   /* and all env strings */
        printf("%s\n", *ptr);

    exit(0);
}
```

Echo all command-line arguments and all environment strings

More about Executable

- http://www.it.uu.se/education/course/homepage/os/vt18/module-2/exec/
- https://www.baeldung.com/linux/exec-functions