CS 332/532 Systems Programming

Lecture 20

Exec family – Linux Processes

Professor : Mahmut Unan – UAB CS

Agenda

- exec family ~quick review
- Linux Processes

- Here is the C APIs for the exec family of system calls:

#include <unistd.h>

```
int execl(const char *pathname, const char
*arg, ...);
int execlp(const char *filename, const char
*arg, ...);
int execle(const char *pathname, const char
*arg, ..., char * const envp[]);
int execv(const char *pathname, char *const
argv[]);
int execvp(const char *filename, char *const
argv[]);
int execvpe(const char *filename, char *const
argv[], char *const envp[]);
```

ls -lh

```
(base) mahmutunan@MacBook-Pro lecture19 % ls -lh
total 240
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   341B Oct  9 13:13 execle.c
-rw-r--r--@ 1 mahmutunan  staff   221B Oct  9 12:58 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   194B Oct  9 12:59 execv.c
-rw-r--r--@ 1 mahmutunan  staff   326B Oct  9 13:19 execve.c
-rw-r--r--@ 1 mahmutunan  staff   198B Oct  9 13:01 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:58 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:00 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:02 exercise4
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:13 exercise5
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:19 exercise6
```

execl

```c
int main(void) {
    char *binaryPath = "/bin/ls";
    char *arg0="ls";
    char *arg1 = "-lh";
    char *arg2 = "/Users/mahmutunan/Desktop/lecture19";

    execl(binaryPath, arg0, arg1, arg2, NULL);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execl.c -o exercise1
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise1
total 160
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   220B Oct  9 12:33 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   192B Oct  9 12:29 execv.c
-rw-r--r--@ 1 mahmutunan  staff   196B Oct  9 12:37 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:33 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:34 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:37 exercise4
(base) mahmutunan@MacBook-Pro lecture19 %
```

execlp

```c
#include <unistd.h>

int main(void) {
    char *commandName = "ls";
    char *arg1 = "-lh";
    char *arg2 = "/Users/mahmutunan/Desktop/lecture19";

    execlp(commandName,commandName, arg1, arg2, NULL);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execlp.c -o exercise2
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise2
total 160
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   221B Oct  9 12:58 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   192B Oct  9 12:29 execv.c
-rw-r--r--@ 1 mahmutunan  staff   196B Oct  9 12:37 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:58 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:34 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:37 exercise4
(base) mahmutunan@MacBook-Pro lecture19 %
```

execv

```c
#include <unistd.h>

int main(void) {
    char *binaryPath = "/bin/ls";
    char *args[]= {"ls","-lh","/Users/mahmutunan/Desktop/lecture19",NULL};

    execv(binaryPath,args);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execv.c -o exercise3
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise3
total 160
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   221B Oct  9 12:58 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   194B Oct  9 12:59 execv.c
-rw-r--r--@ 1 mahmutunan  staff   196B Oct  9 12:37 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:58 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:00 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:37 exercise4
(base) mahmutunan@MacBook-Pro lecture19 % _
```

execvp

```c
#include <unistd.h>

int main(void) {
    char *commandName= "ls";
    char *args[]= {commandName,"-lh","/Users/mahmutunan/Desktop/lecture19",NULL};

    execvp(commandName,args);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execvp.c -o exercise4
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise4
total 160
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   221B Oct  9 12:58 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   194B Oct  9 12:59 execv.c
-rw-r--r--@ 1 mahmutunan  staff   198B Oct  9 13:01 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:58 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:00 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:02 exercise4
(base) mahmutunan@MacBook-Pro lecture19 %
```

Environment / Environment Variable

- An important Unix concept is the **environment**, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.
- A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.
- When you log in to the system, the shell undergoes a phase called **initialization** to set up the environment. Environment variables allow you to customize how the system works and the behavior of the applications on the system.

https://www.tutorialspoint.com/unix/unix-environment.htm

Environment / Environment Variable

Sr.No.	Variable & Description
1	**DISPLAY** Contains the identifier for the display that **X11** programs should use by default.
2	**HOME** Indicates the home directory of the current user: the default argument for the cd **built-in** command.
3	**IFS** Indicates the **Internal Field Separator** that is used by the parser for word splitting after expansion.
4	**LANG** LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is **pt_BR**, then the language is set to (Brazilian) Portuguese and the locale to Brazil.
5	**LD_LIBRARY_PATH** A Unix system with a dynamic linker, contains a colonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
6	**PATH** Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
7	**PWD** Indicates the current working directory as set by the cd command.
8	**RANDOM** Generates a random integer between 0 and 32,767 each time it is referenced.

https://www.tutorialspoint.com/unix/unix-environment.htm

execle

```c
#include <unistd.h>

int main(void) {
    char *binaryPath= "/bin/bash";
    char *arg1 = "-c";
    char *arg2 = "echo \"Visit $HOSTNAME:$PORT from your browser.\"";
    char *const env[] = {"HOSTNAME=https://www.uab.edu/cas/computerscience/", "PORT=8080", NULL};

    execle(binaryPath, binaryPath,arg1, arg2, NULL, env);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execle.c -o exercise5
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise5
Visit https://www.uab.edu/cas/computerscience/:8080 from your browser.
(base) mahmutunan@MacBook-Pro lecture19 % _
```

execve

```c
#include <unistd.h>

int main(void) {
    char *binaryPath= "/bin/bash";
    char *const args[] = {binaryPath,"-c","echo \"Visit $HOSTNAME:$PORT from your browser.\"",NULL};
    char *const env[] = {"HOSTNAME=https://www.uab.edu/cas/computerscience/", "PORT=8080", NULL};

    execve(binaryPath, args, env);

    return 0;
}
```

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc execve.c -o exercise6
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise6
Visit https://www.uab.edu/cas/computerscience/:8080 from your browser.
(base) mahmutunan@MacBook-Pro lecture19 %
```

Ref : https://linuxhint.com/exec_linux_system_call_c/

forkexecl.c

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

int main(int argc, char **argv) {
    pid_t pid;
    int status;

    pid = fork();
```

```
12      if (pid == 0) { /* this is child process */
13          execl( path: "/usr/bin/uname",  arg0: "uname", "-a", (char *)NULL);
14          printf("If you see this statement then execl failed ;-(\n");
15      perror("execl");
16      exit(-1);
17      } else if (pid > 0) { /* this is the parent process */
18          printf("Wait for the child process to terminate\n");
19          wait(&status); /* wait for the child process to terminate */
20          if (WIFEXITED(status)) { /* child process terminated normally */
21              printf("Child process exited with status = %d\n", WEXITSTATUS(status));
22          } else { /* child process did not terminate normally */
23              printf("Child process did not terminate normally!\n");
24              /* look at the man page for wait (man 2 wait) to determine
25                 how the child process was terminated */
26          }
27      } else { /* we have an error */
28          perror("fork"); /* use perror to print the system error message */
29          exit(EXIT_FAILURE);
30      }
31
32      printf("[%ld]: Exiting program .....\n", (long)getpid());
33
34      return 0;
35  }
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall forkexecl.c -o exercise1
(base) mahmutunan@MacBook-Pro lecture18 % ./exercise1
Wait for the child process to terminate
Darwin MacBook-Pro.local 19.6.0 Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT
 2020; root:xnu-6153.141.1~1/RELEASE_X86_64 x86_64
Child process exited with status = 0
[1290]: Exiting program .....
(base) mahmutunan@MacBook-Pro lecture18 %
```

- Let us look at the different versions of the exec functions. There are two classes of exec functions based on whether the argument is a list of separate values (l versions) or the argument is a vector (v versions):
- functions that take a variable number of command-lines arguments each as an array of characters terminated with a null character and the last argument is a null pointer – *(char *)NULL (execl, execlp, and execle)*
- functions that take the command-line arguments as a pointer to an array of pointers to the arguments, similar to argv parameter used by the main method (execv, execvp, and execvpe)

- Functions that have *p* in the name use *filename* as the first argument while functions without *p* use the *pathname* as the first argument. If the filename contains a slash character (*/*), it is considered as a pathname, otherwise, all directories specified by the PATH environment variable are searched for the executable.
- Functions that end in *e* have an additional argument – a pointer to an array of pointers to the environment strings.

forkexecv.c

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

int main(int argc, char **argv) {
    pid_t pid;
    int status;
    char *args[] = {"uname", "-a", (char *)NULL};

    pid = fork();
    if (pid == 0) { /* this is child process */
        execv( path: "/usr/bin/uname", args);
        printf("If you see this statement then execl failed ;-(\n");
    perror("execv");
    exit(-1);
    } else if (pid > 0) { /* this is the parent process */
```

forkexecv.c

```
18      } else if (pid > 0) { /* this is the parent process */
19          printf("Wait for the child process to terminate\n");
20          wait(&status); /* wait for the child process to terminate */
21          if (WIFEXITED(status)) { /* child process terminated normally */
22              printf("Child process exited with status = %d\n", WEXITSTATUS(status));
23          } else { /* child process did not terminate normally */
24              printf("Child process did not terminate normally!\n");
25              /* look at the man page for wait (man 2 wait) to determine
26                 how the child process was terminated */
27          }
28      } else { /* we have an error */
29          perror("fork"); /* use perror to print the system error message */
30          exit(EXIT_FAILURE);
31      }
32
33      printf("[%ld]: Exiting program .....\n", (long)getpid());
34
35      return 0;
36  }
37
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall forkexecv.c -o exercise2
(base) mahmutunan@MacBook-Pro lecture18 % ./exercise2
Wait for the child process to terminate
Darwin MacBook-Pro.local 19.6.0 Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT
 2020; root:xnu-6153.141.1~1/RELEASE_X86_64 x86_64
Child process exited with status = 0
[30193]: Exiting program .....
(base) mahmutunan@MacBook-Pro lecture18 % _
```

In order to see the difference between execl and execv, here is a line of code executing a

`ls -l -R -a`

with execl :

```
execl("/bin/ls", "ls", "-l", "-R", "-a", NULL);
```

with execv :

```
char* arr[] = {"ls", "-l", "-R", "-a", NULL};
execv("/bin/ls", arr);
```

The array of char* sent to execv will be passed to /bin/ls as argv (in `int main(int argc, char **argv)`)

https://stackoverflow.com/questions/32142164/how-to-use-execv-system-call-in-linux

forkexecvp.c

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

int main(int argc, char **argv) {
    pid_t pid;
    int status;

    if (argc < 2) {
        printf("Usage: %s <command> [args]\n", argv[0]);
        exit(-1);
    }

    pid = fork();
    if (pid == 0) { /* this is child process */
        execvp( file: argv[1],  argv: &argv[1]);
        printf("If you see this statement then execl failed ;-(\n");
        perror("execvp");
        exit(-1);
```

forkexecvp.c

```
22      } else if (pid > 0) { /* this is the parent process */
23          printf("Wait for the child process to terminate\n");
24          wait(&status); /* wait for the child process to terminate */
25          if (WIFEXITED(status)) { /* child process terminated normally */
26              printf("Child process exited with status = %d\n", WEXITSTATUS(status));
27          } else { /* child process did not terminate normally */
28              printf("Child process did not terminate normally!\n");
29              /* look at the man page for wait (man 2 wait) to determine
30                 how the child process was terminated */
31          }
32      } else { /* we have an error */
33          perror("fork"); /* use perror to print the system error message */
34          exit(EXIT_FAILURE);
35      }
36
37      printf("[%ld]: Exiting program .....\n", (long)getpid());
38
39      return 0;
40  }
41
```

hello.c

```c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("Hello from the execvp()\n");

     return 0;
}
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall forkexecvp.c -o exercise3
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall hello.c -o hello
(base) mahmutunan@MacBook-Pro lecture18 % ./exercise3 ./hello
Wait for the child process to terminate
Hello from the execvp()
Child process exited with status = 0
[30387]: Exiting program .....
(base) mahmutunan@MacBook-Pro lecture18 %
```

forkexecvp2.c

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

int main(int argc, char **argv) {
    pid_t pid;
    int status;

    if (argc < 2) {
        printf("Usage: %s <command> [args]\n", argv[0]);
        exit(-1);
    }

    pid = fork();
    if (pid == 0) { /* this is child process */
        execvp( file: argv[1], argv: &argv[1]);
        printf("If you see this statement then execl failed ;-(\n");
    perror("execvp");
    exit(-1);
```

```c
    } else if (pid > 0) { /* this is the parent process */
        char name[BUFSIZ];

        printf("[%d]: Please enter your name: ", getpid());
        scanf("%s", name);
        printf("[%d-stdout]: Hello %s!\n", getpid(), name);
        fprintf(stderr, "[%d-stderr]: Hello %s!\n", getpid(), name);

        wait(&status); /* wait for the child process to terminate */
        if (WIFEXITED(status)) { /* child process terminated normally */
            printf("Child process exited with status = %d\n", WEXITSTATUS(status));
        } else { /* child process did not terminate normally */
            printf("Child process did not terminate normally!\n");
            /* look at the man page for wait (man 2 wait) to determine
               how the child process was terminated */
        }
    } else { /* we have an error */
        perror("fork"); /* use perror to print the system error message */
        exit(EXIT_FAILURE);
    }

    printf("[%ld]: Exiting program .....\n", (long)getpid());

    return 0;
}

```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture19 % gcc -Wall forkexecvp2.c -o exercise7
(base) mahmutunan@MacBook-Pro lecture19 % ./exercise7 ls -lh
[65577]: Please enter your name: total 280
-rw-r--r--@ 1 mahmutunan  staff   239B Oct  9 12:56 execl.c
-rw-r--r--@ 1 mahmutunan  staff   341B Oct  9 13:13 execle.c
-rw-r--r--@ 1 mahmutunan  staff   221B Oct  9 12:58 execlp.c
-rw-r--r--@ 1 mahmutunan  staff   194B Oct  9 12:59 execv.c
-rw-r--r--@ 1 mahmutunan  staff   326B Oct  9 13:19 execve.c
-rw-r--r--@ 1 mahmutunan  staff   198B Oct  9 13:01 execvp.c
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:56 exercise1
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 12:58 exercise2
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:00 exercise3
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:02 exercise4
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:13 exercise5
-rwxr-xr-x  1 mahmutunan  staff    12K Oct  9 13:19 exercise6
-rwxr-xr-x  1 mahmutunan  staff    13K Oct  9 13:51 exercise7
-rw-r--r--@ 1 mahmutunan  staff   1.8K Oct  5 19:48 forkexecvp2.c
mahmut
[65577-stdout]: Hello mahmut!
[65577-stderr]: Hello mahmut!
Child process exited with status = 0
[65577]: Exiting program .....
(base) mahmutunan@MacBook-Pro lecture19 % _
```

Processes in a Linux environment

- User processes in a Linux environment could be in one of the following three states: `foreground, background`, or `suspended.`
- Most interactive applications that take input from the keyboard or command-line argument and display output in a terminal are considered as foreground processes. Till now, we have been executed all our programs as foreground processes by typing the name of the command in the bash shell.

- Non-interactive processes that are typically not connected to a terminal and execute in the background are considered as background processes.
- You can execute a program in the background by typing the program name followed by the symbol `&` at the end.
- You will notice that the shell returns the command-prompt with the background process number and the corresponding process identifier (PID) of the process that was created.
- For example:

```
$ nano myprog.c &
[1] 16946
$
```

- You can display various jobs that are currently running in the background using the *jobs* command. It will show the job number, the current state of the job, and the job name. For example:

```
$ jobs
[1]+  Stopped                 nano myprog.c
$
```

- If you like to list additional information such as the PID of the job you can use the -l option. For example:

```
$ jobs -l
[1]+ 16946 Stopped (tty output)  nano myprog.c
$
```

In case of the above example, we have invoked an editor and it has been stopped since it requires terminal to display the output and continue. If we had created a non-interactive job in the background, then it would be in the running state. For example:

```
$ sleep 20 &
[2] 17513
$ jobs
[1]+  Stopped                 nano myprog.c
[2]-  Running                 sleep 20 &
$
```

- We can bring a job that is running in the background to foreground by using the *fg* command. For example, to switch the *sleep* process to foreground, we specify the job number after the % symbol:

```
$ fg %2
sleep 20
$ jobs
[1]+  Stopped                 nano myprog.c
$
```

- You will notice that the sleep process will return the command prompt in the terminal when it completes execution and when you type *jobs* again, it will only show one process. You can use `fg` command to switch to the editor using: `fg %1`.
- You can continue with your edits, save the file, and exit the editor.
- After you exit, if you type `jobs` again, you will notice that it does not list any processes since the *nano* process is no longer executing.
- If you like to suspend a foreground process then type *Control-Z* when the program is executing and that process will be suspended

- For example, if we started the sleep process in foreground and would like to suspend it, then type *Control-Z* and you will see a message similar to what you saw when you started a process in background:

```
$
$ jobs
$ sleep 100
^Z
[1]+  Stopped                 sleep 100
$ jobs
[1]+  Stopped                 sleep 100
$
```

- However, notice that the sleep process is stopped (it is not running) unlike the previous case when it was running in the background. If you like the sleep process to continue then you have to using the *bg* command as follows:

```
$ bg %1
[1]+ sleep 100 &
$ jobs
[1]+  Running                 sleep 100 &
$
[1]+  Done                    sleep 100
$ jobs
```

- Now you notice that the sleep process is running and when it is done you will see the message *Done* in your terminal.
- There are special background processes that are started at system startup and they continue to run till the system is shutdown.
- These special background processes are called ***daemons***.
- These processes typically end in “d” and some examples are: systemd, crond, ntpd, nfsd, sshd, httpd, named.
- If you like to terminate a process that is executing in the foreground, you use Control-C to kill it.
- If you like to terminate a process in the background, you could bring it to foreground and then use Control-C or use the *kill* command to terminate the background process directly

- For example:
- ```
  $ jobs
  $ sleep 100 &
  [1] 1519
  $ jobs
  [1]+  Running                 sleep 100 &
  $ kill %1
  [1]+  Terminated              sleep 100
  $
  $ jobs
  ```
  $ You can also provide the PID as the argument to *kill* command to terminate a process.
  ```