CS 332/532 Systems Programming

Lecture 19

-Process Creation and Management-

Professor : Mahmut Unan – UAB CS


Experience UAB's largest annually held
Entrepreneurship, Tech and Engineering Mixer event
ACM
MIXER
WOMEN IN TECH
AT UAB
Tech x Engineering x Entrepreneurship
GOLD HALL UNDERGROUND
Thursday, October 17th @ 5:30 PM
Multiple Departments | Complimentary Catering
New Connections | Endless Possibilities
swe
Society of Women Engineers
University of Alabama at Birmingham
CE
The Collegiate Entrepreneurs' Organization
CREATING A WORLD OF OPPORTUNITIES
UNIVERSITY OF ALABAMA AT BIRMINGHAM
Business Casual Attire
Open to all students
For disability accommodations, please contact: bjpatel@uab.edu

Agenda

- execl
- execv
- execlp
- execvp
- execle
- execve

wait()

wait, waitpid, waitid - wait for process to change state

```
 pid_t wait(int *wstatus);
 pid_t waitpid(pid_t pid, int *wstatus, int
options);
 int waitid(idtype_t idtype, id_t id, siginfo_t
*infop, int options);
```

https://www.man7.org/linux/man-pages/man2/waitid.2.html

wait()

- The wait() call returns the PID of the child process that terminated when successful, otherwise, it returns -1.
- The wait() call also sets an integer value that is passed as an argument to the function which can be inspected with various macros provided in <sys/wait.h> to determine how the child process completed (e.g., terminated normally, terminated by a signal).

wait() waitpid()

- If the calling process created more than one child process, we can use the waitpid() system call to wait on a specific child process to change state.
- A state change could be any one of the following events: the child was terminated; the child was stopped by a signal; or the child was resumed by a signal. Similar to wait(), waitpid() returns the PID of the child process that changed state when successful, otherwise, it returns -1.

wait() waitpid()

- Here are the C APIs for
  the wait() and waitpid() system calls:

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int
options);
```

```c
int main() {
    int processId = fork();
    int count;
    fflush(stdout);

    if (processId==0){
        count=1;
    }else{
        count=6;
    }
    if (processId!=0){
        wait();
    }
   int i;
    for (i=count;i<count+5;i++){
        printf("%d",i);
        fflush( stdout );
    }

}
```

```
12345678910
Process finished with exit code 0
```

https://www.youtube.com/watch?v=tcYo6hipaSA&ab_channel=CodeVault

Example 1

- We will create a sample program to illustrate how to use fork() to create a child process, wait for the child process to terminate, and display the parent and child process ID in both processes.

fork.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();
    if (pid == 0) { /* this is child process */
        printf("This is the child process, my PID is %ld and my parent PID is %ld\n",
            (long)getpid(), (long)getppid());
    } else if (pid > 0) { /* this is the parent process */
        printf("This is the parent process, my PID is %ld and the child PID is %ld\n",
            (long)getpid(), (long)pid);

        printf("Wait for the child process to terminate\n");
```

fork.c

```
18
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("ERROR: Child process did not terminate normally!\n");
25              /* look at the man page for wait (man 2 wait) to
26                              determine how the child process was terminated */
27          }
28      } else { /* we have an error in process creation */
29          perror("fork");
30          exit(EXIT_FAILURE);
31      }
32
33      printf("[%ld]: Exiting program .....\n", (long)getpid());
34
35      return 0;
36  }
37
```

fork.c

```
(base) mahmutunan@MacBook-Pro lecture17 % ./exercise1
This is the parent process, my PID is 90695 and the child PID is 90696
Wait for the child process to terminate
This is the child process, my PID is 90696 and my parent PID is 90695
[90696]: Exiting program .....
Child process exited with status = 0
[90695]: Exiting program .....
(base) mahmutunan@MacBook-Pro lecture17 %
```

exec()

- execl, execlp, execle, execv, execvp, execvpe - execute a file
- The **exec**() family of functions replaces the current process image with a new process image.

https://man7.org/linux/man-pages/man3/exec.3.html

exec()

- Note that the child process is a copy of the parent process and control is split at the invocation of the fork() call between the parent and the child process.
- If we like the child process to execute a different program other than making a copy of the parent process, we can use the exec family of system calls to replace the current process image with a new one.

- 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[]);
```

- We will use the execl() to replace the child process created by fork().
- The execl() function takes as arguments the full pathname of the executable along with a pointer to an array of characters for each argument.
- Since we can have a variable number of arguments, the last argument is a null pointer.

p1.c

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
    printf("Hello from p1, process id= () %d\n", getpid());
    char *args[]={"Hello","CS","332",NULL};
    execv( path: "./p2",args);
    printf("we are not supposed to see this text");
    return 0;
}
```

p2

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

int main(int argc, char **argv) {
    printf("Hello from p2, process id= () %d\n", getpid());
    printf("The arguments are %s %s %s\n",argv[0],argv[1],argv[2]);
    printf("Now, the child process will terminate\n");
    return 0;
}
```

compile & run

```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p1.c -o p1
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p2.c -o p2
(base) mahmutunan@MacBook-Pro lecture18 % ./p1
Hello from p1, process id= () 30983
Hello from p2, process id= () 30983
The arguments are Hello CS 332
Now, the child process will terminate
(base) mahmutunan@MacBook-Pro lecture18 % _
```

p1.c

- Let's modify it a little bit and fork the process

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

int main(int argc, char **argv) {
    printf("Hello from p1, process id= () %d\n", getpid());
    int pid=fork();
    int status;
    if (pid == 0) {
    char *args[] = {"Hello", "CS", "332", NULL};
    execv( path: "./p2", args);
    printf("we are not supposed to see this text");
   }
    else if(pid>0){
        wait(&status);
    }
   return 0;
}
```

p2.c

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

int main(int argc, char **argv) {
    printf("Hello from p2, process id= () %d\n", getpid());
    printf("The arguments are %s %s %s\n",argv[0],argv[1],argv[2]);
    printf("Now, the child process will terminate\n");
    return 0;
}
```

compile&run

```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p1.c -o p1
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p2.c -o p2
(base) mahmutunan@MacBook-Pro lecture18 % ./p1
Hello from p1, process id= () 30922
Hello from p2, process id= () 30923
The arguments are Hello CS 332
Now, the child process will terminate
(base) mahmutunan@MacBook-Pro lecture18 %
```

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();
```

```c
    if (pid == 0) { /* this is child process */
        execl( path: "/usr/bin/uname",  arg0: "uname", "-a", (char *)NULL);
        printf("If you see this statement then execl failed ;-(\n");
    perror("execl");
    exit(-1);
    } else if (pid > 0) { /* this is the parent process */
        printf("Wait for the child process to terminate\n");
        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 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;
}

```

- 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();
```

```c
    if (pid == 0) { /* this is child process */
        execl( path: "/usr/bin/uname",  arg0: "uname", "-a", (char *)NULL);
        printf("If you see this statement then execl failed ;-(\n");
    perror("execl");
    exit(-1);
    } else if (pid > 0) { /* this is the parent process */
        printf("Wait for the child process to terminate\n");
        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 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 %
```