CS 332/532 Systems Programming

Lecture 17

-Process Creation and Management-

Professor : Mahmut Unan – UAB CS

Agenda

- exec

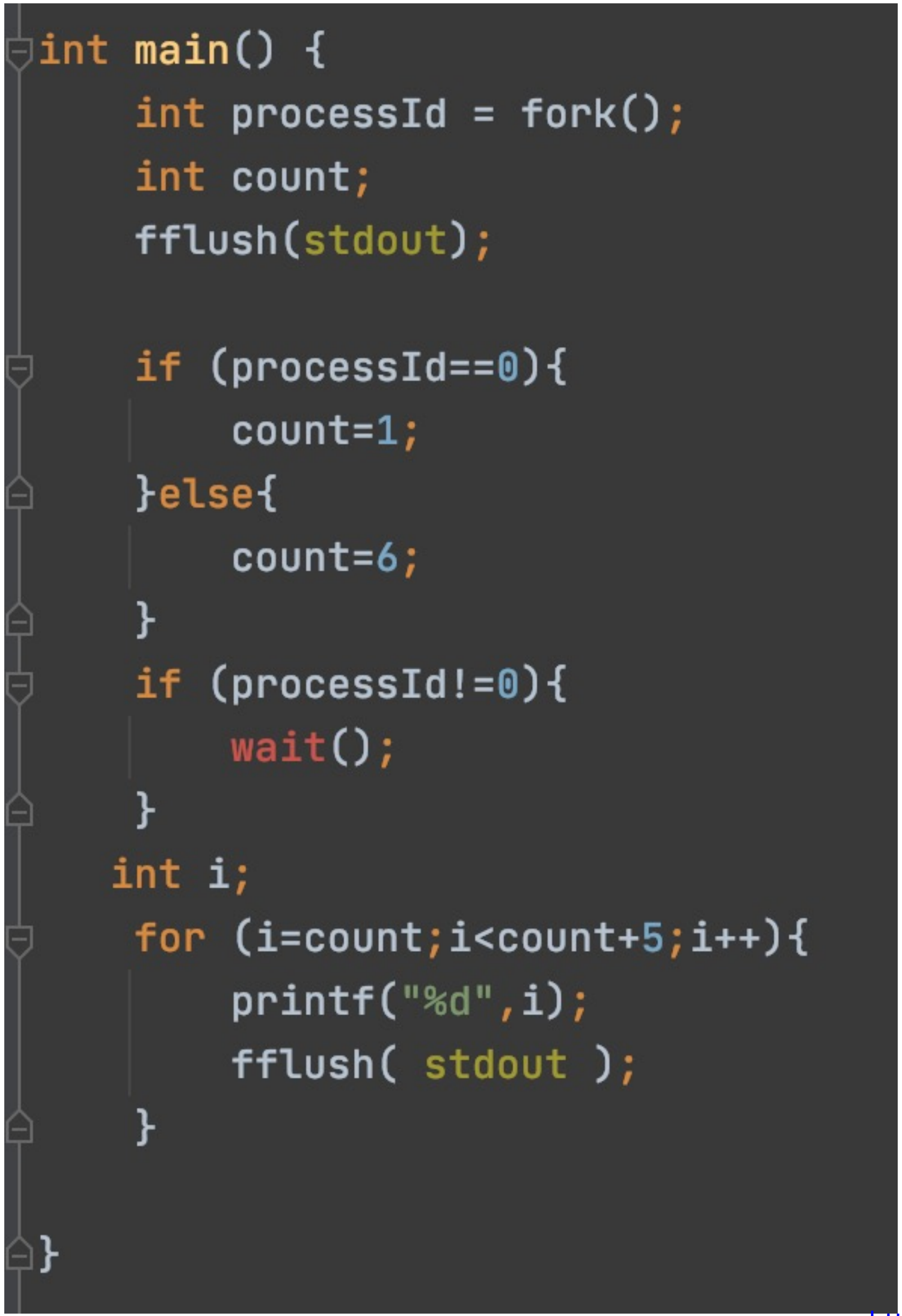
Exam 1

- February 28th, 2022
  - Monday
- Friday
  - Review

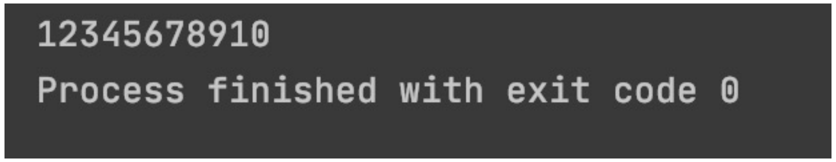
wait() waitpid()

- 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 );
    }

}
```



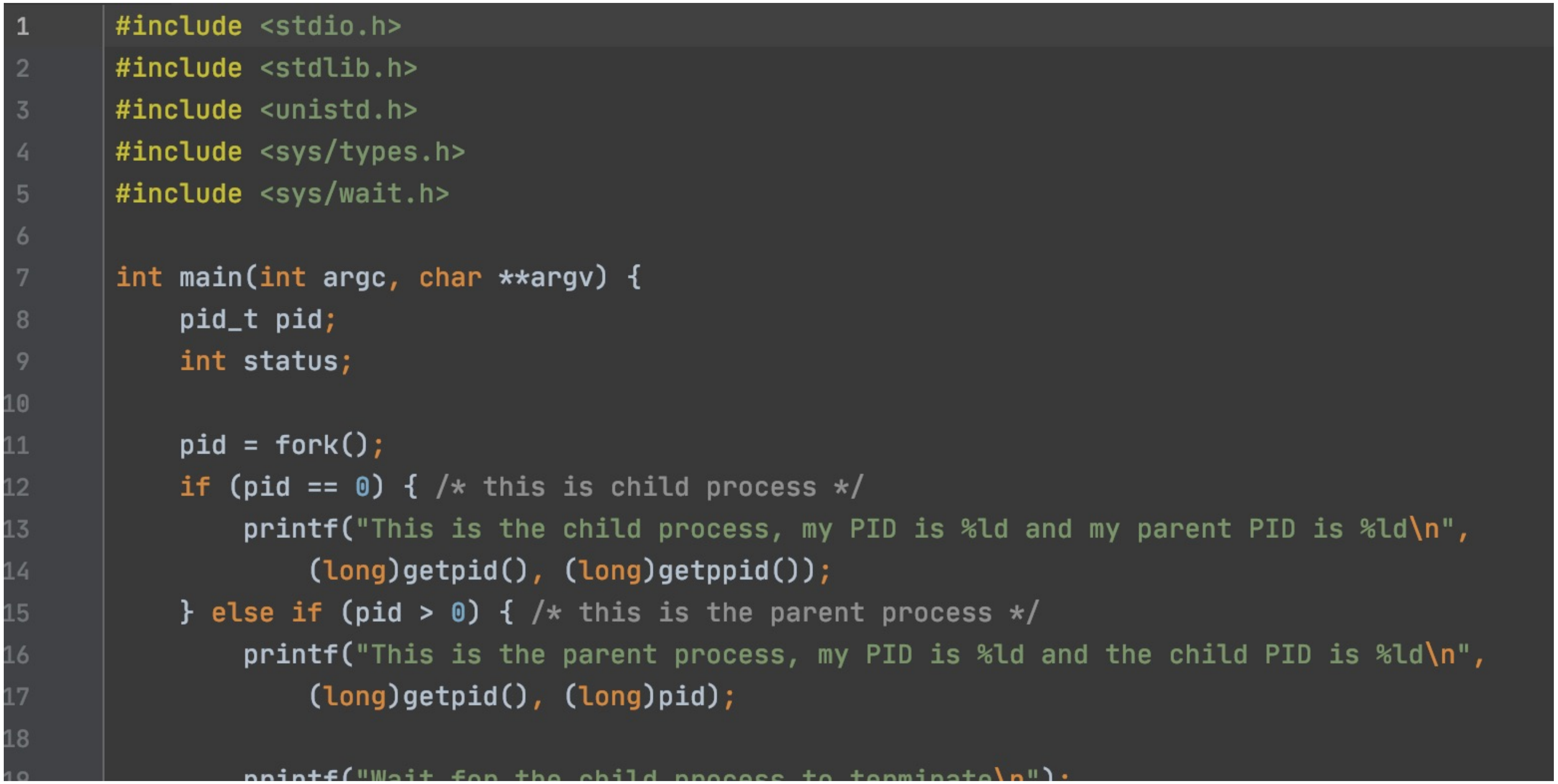
```
12345678910
Process finished with exit code 0
```

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

Example 1

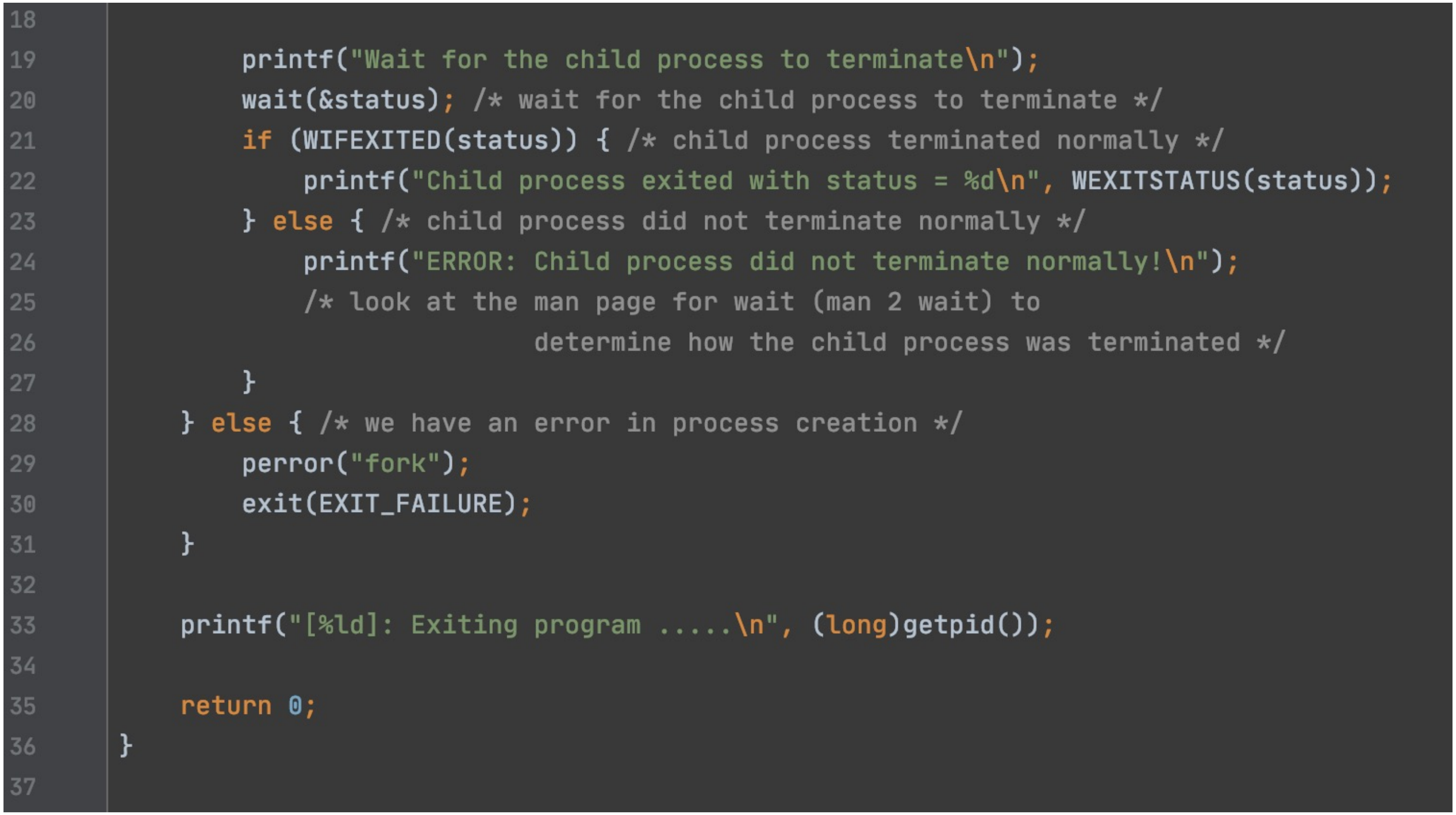
- We will create a sample program to illustrate how to use fork() to create a child process, wait for the child process to terminate, and display the parent and child process ID in both processes.

fork.c



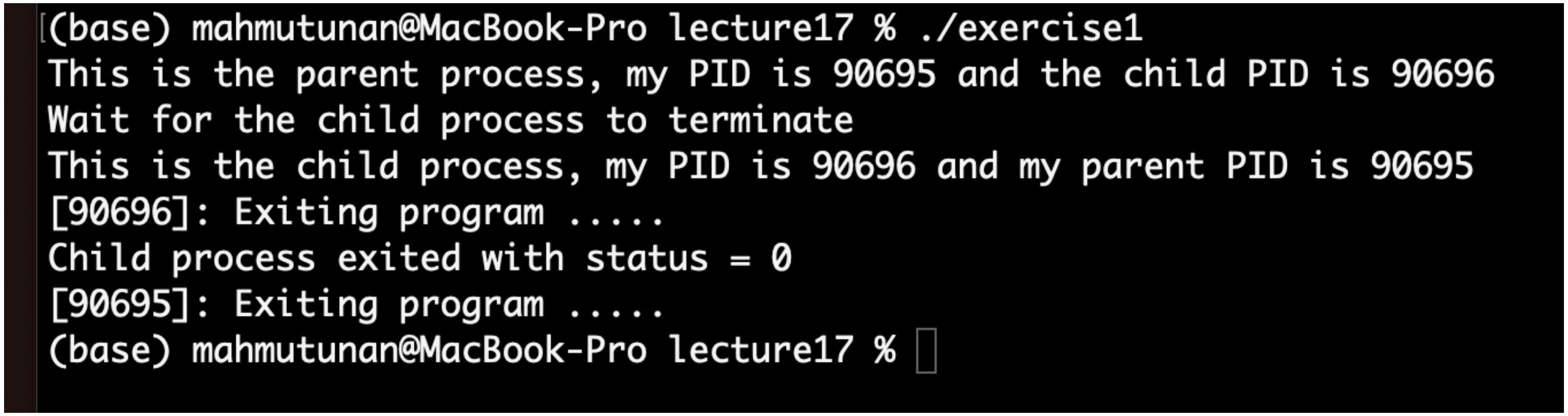
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

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



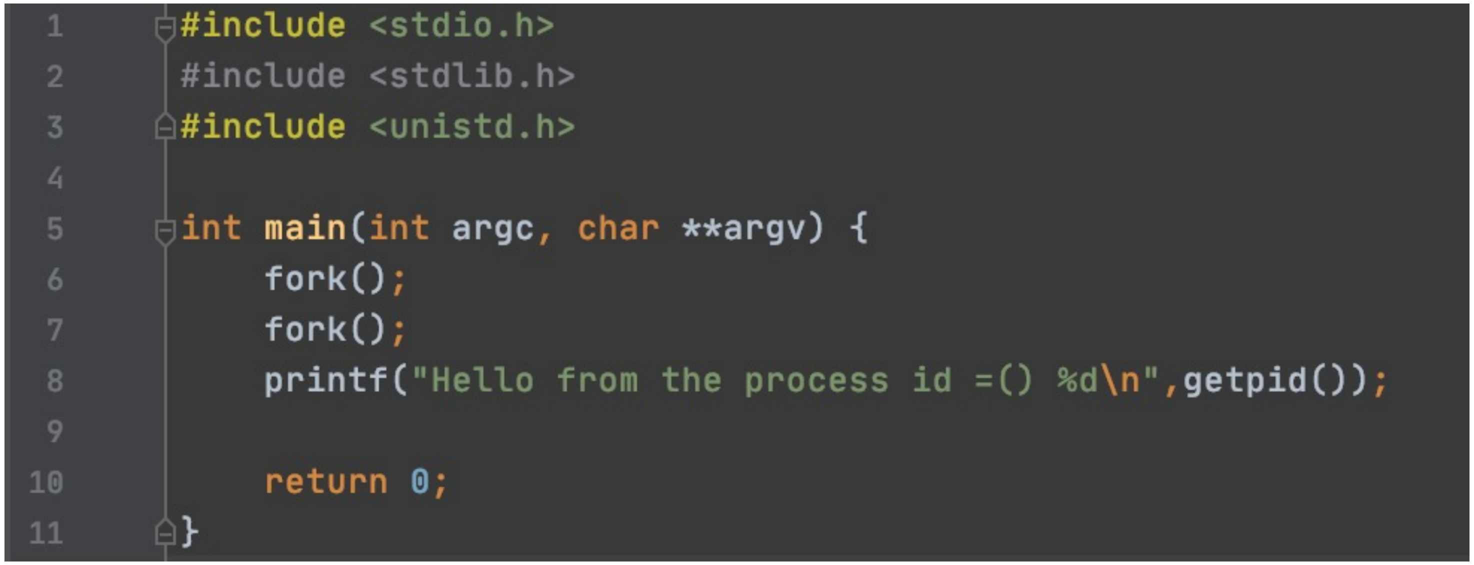
```c

        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("ERROR: 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 in process creation */
        perror("fork");
        exit(EXIT_FAILURE);
    }

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

    return 0;
}

```

fork.c

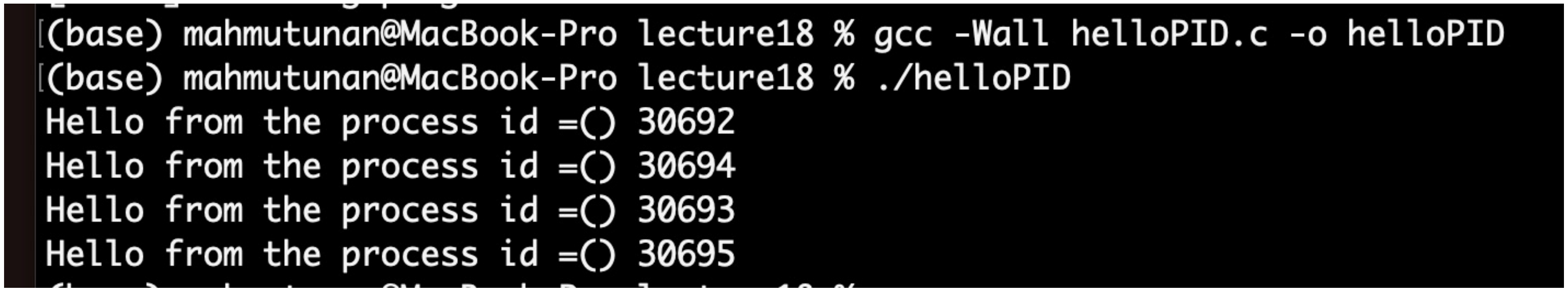


```
(base) mahmutunan@MacBook-Pro lecture17 % ./exercise1
This is the parent process, my PID is 90695 and the child PID is 90696
Wait for the child process to terminate
This is the child process, my PID is 90696 and my parent PID is 90695
[90696]: Exiting program .....
Child process exited with status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

fork() -recall



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

int main(int argc, char **argv) {
    fork();
    fork();
    printf("Hello from the process id =() %d\n",getpid());

    return 0;
}
```



```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall helloPID.c -o helloPID
(base) mahmutunan@MacBook-Pro lecture18 % ./helloPID
Hello from the process id =() 30692
Hello from the process id =() 30694
Hello from the process id =() 30693
Hello from the process id =() 30695
```

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.

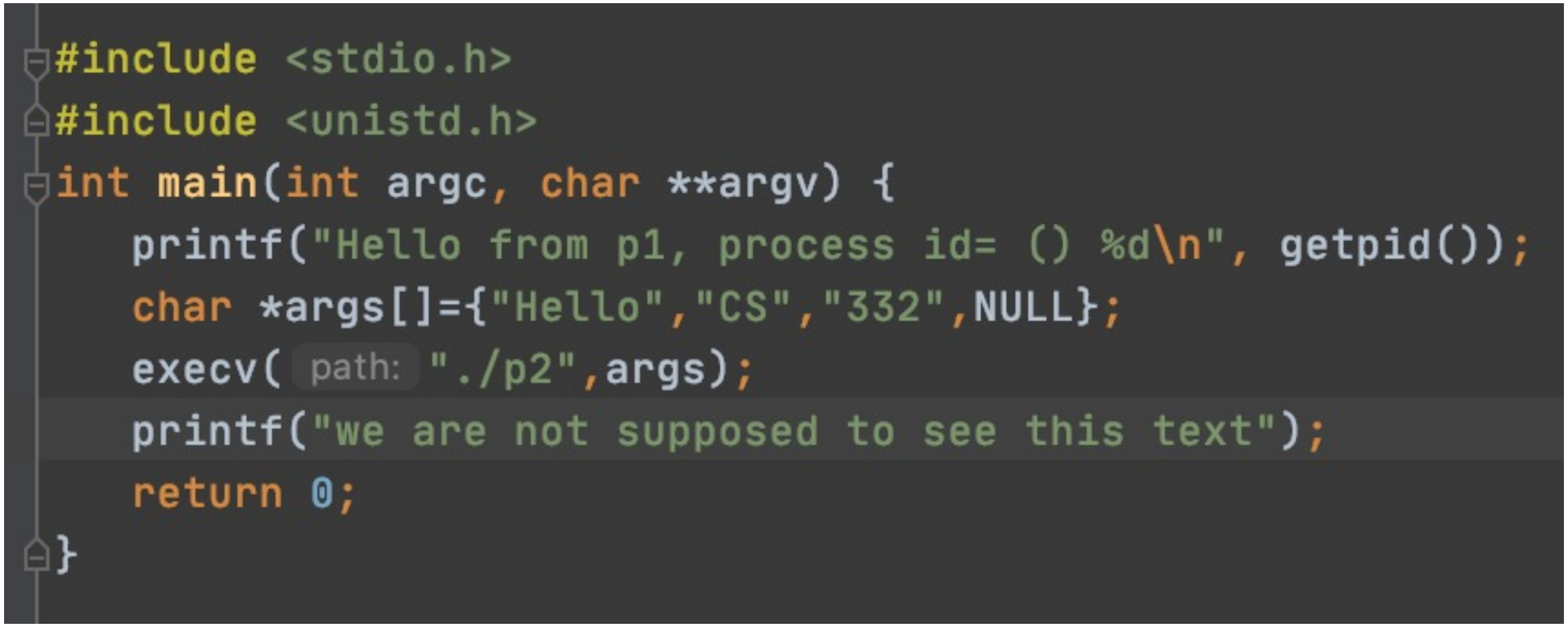
- Here is the C APIs for the exec family of system calls:

#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[]);
```

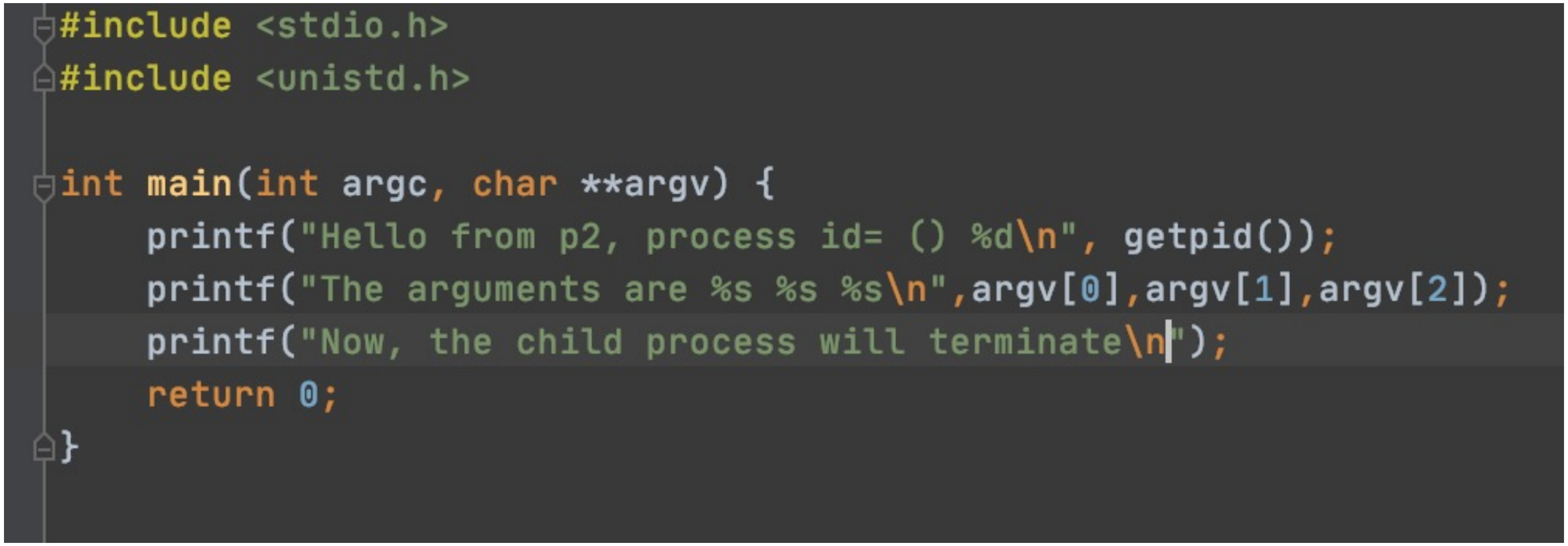
- We will use the execl() to replace the child process created by fork().
- The execl() function takes as arguments the full pathname of the executable along with a pointer to an array of characters for each argument.
- Since we can have a variable number of arguments, the last argument is a null pointer.

p1.c



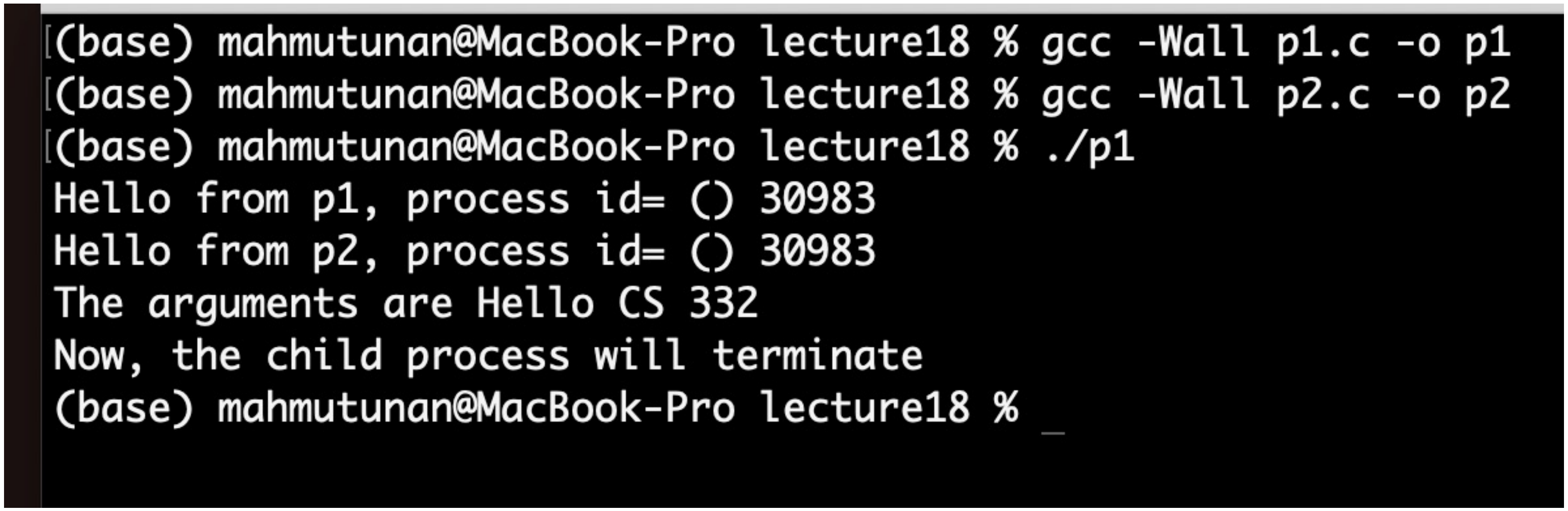
```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
    printf("Hello from p1, process id= () %d\n", getpid());
    char *args[]={"Hello","CS","332",NULL};
    execv( path: "./p2",args);
    printf("we are not supposed to see this text");
    return 0;
}
```

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;
}
```

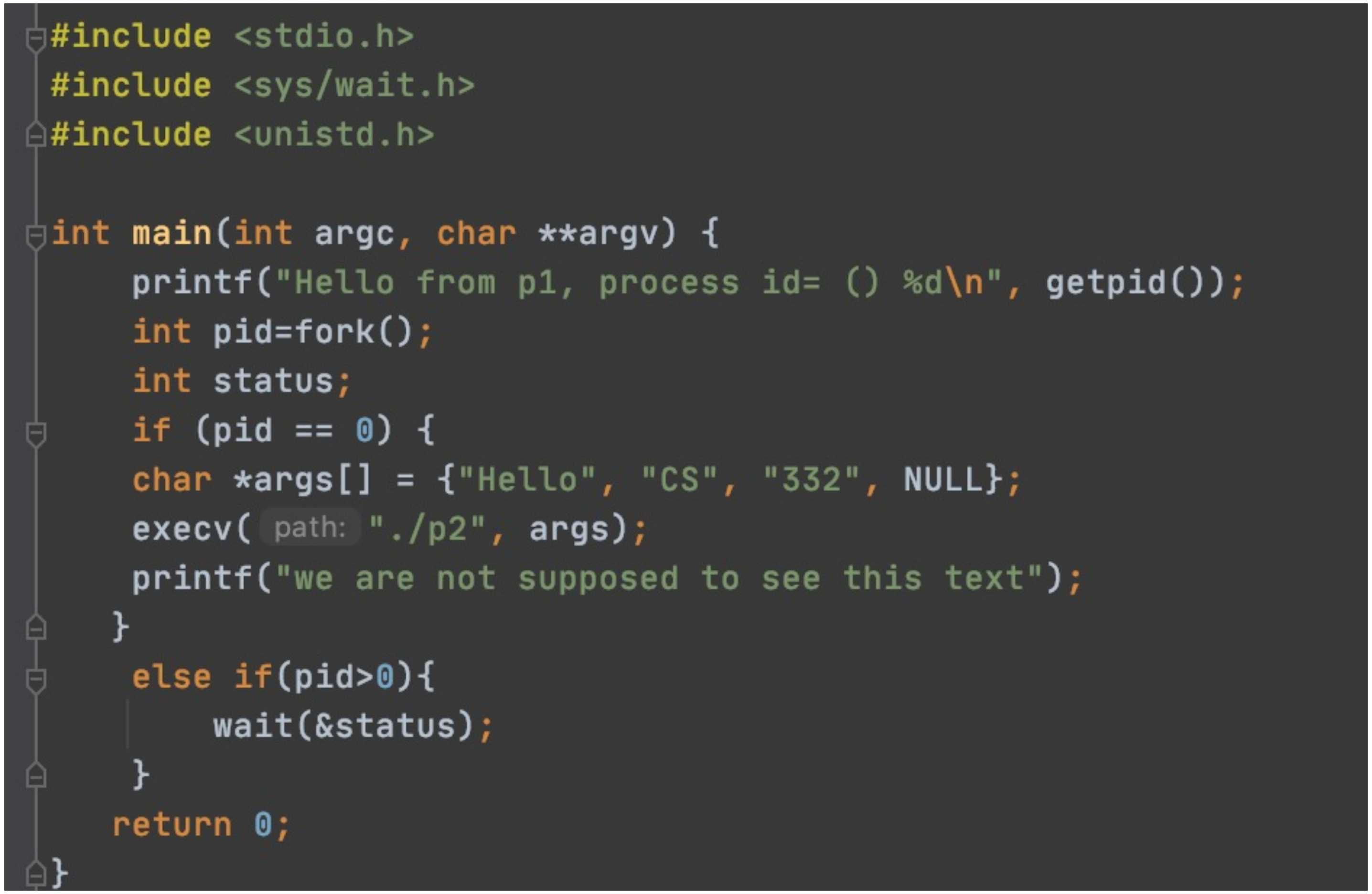
compile & run



```
(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 % _
```

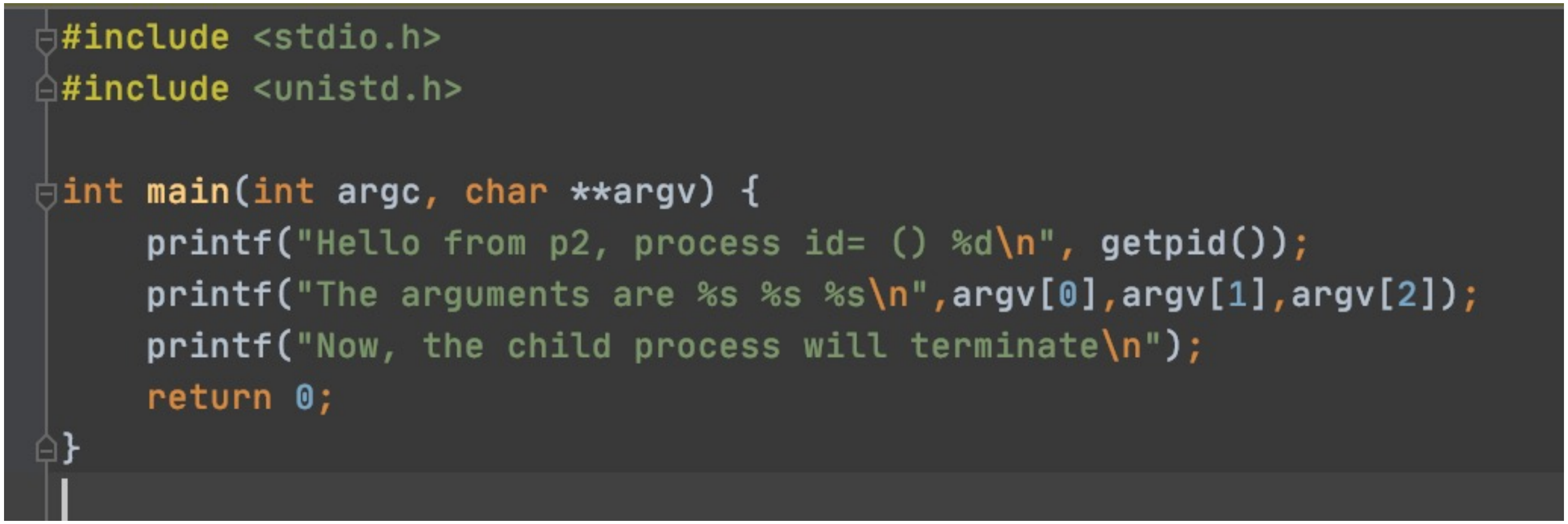
p1.c

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



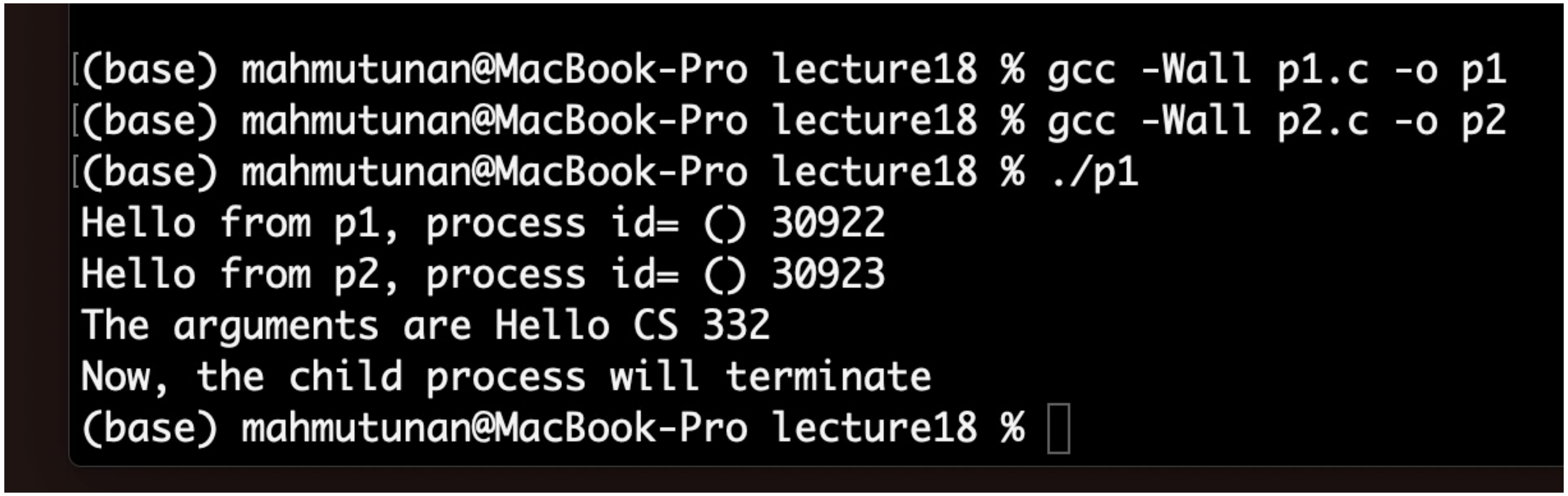
```c
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

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



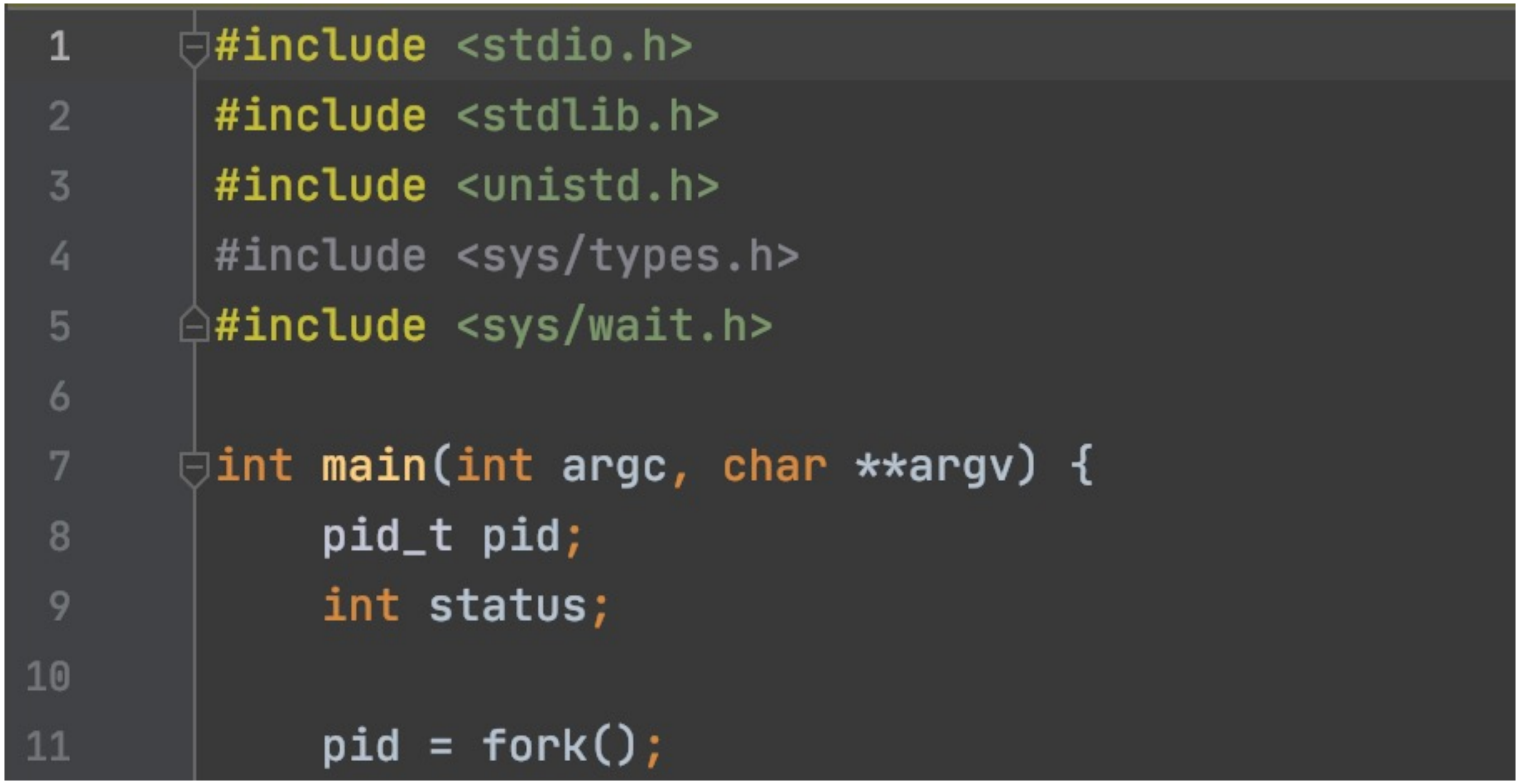
```c
#include <stdio.h>
#include <unistd.h>

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

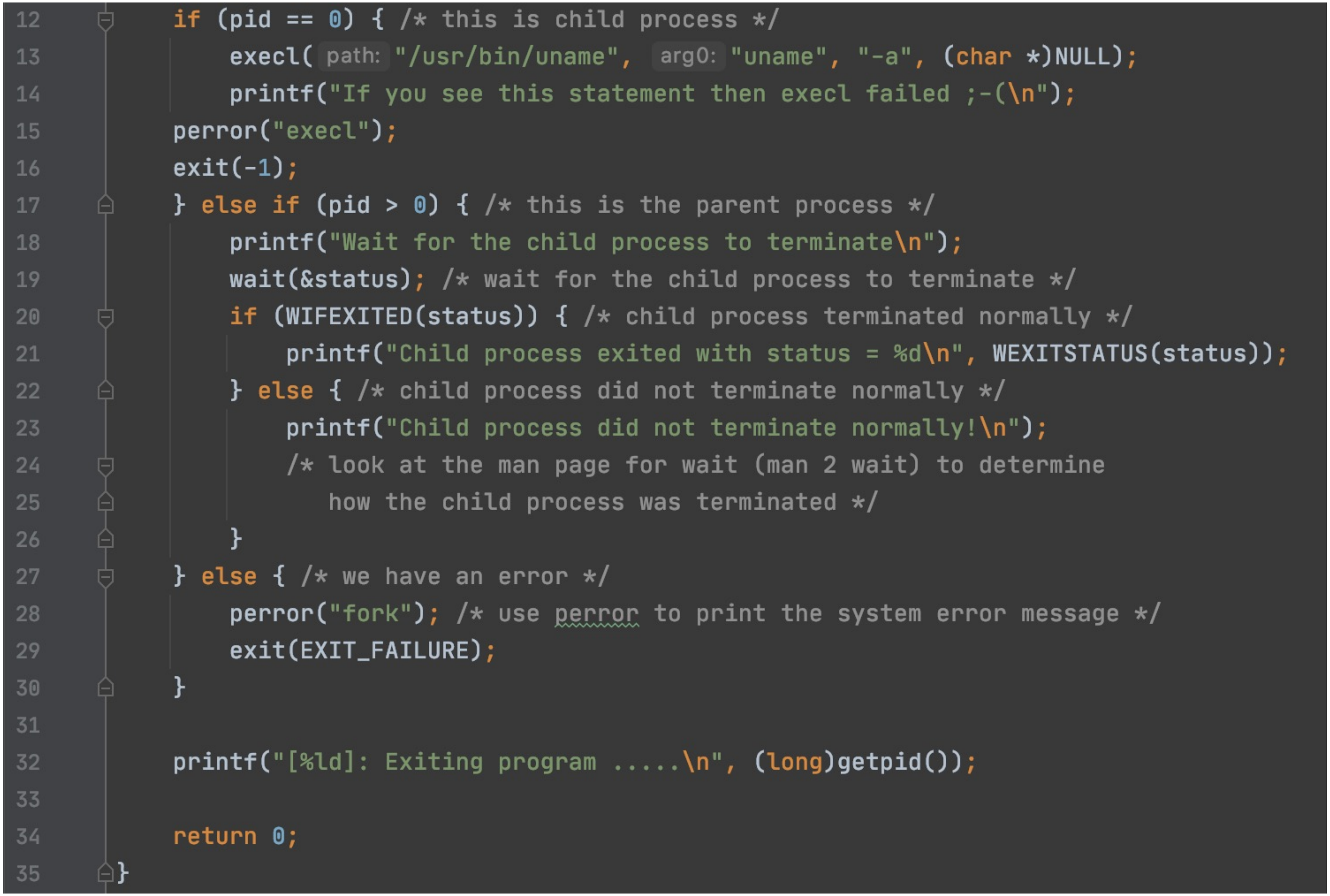


```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p1.c -o p1
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall p2.c -o p2
(base) mahmutunan@MacBook-Pro lecture18 % ./p1
Hello from p1, process id= () 30922
Hello from p2, process id= () 30923
The arguments are Hello CS 332
Now, the child process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

forkexecl.c

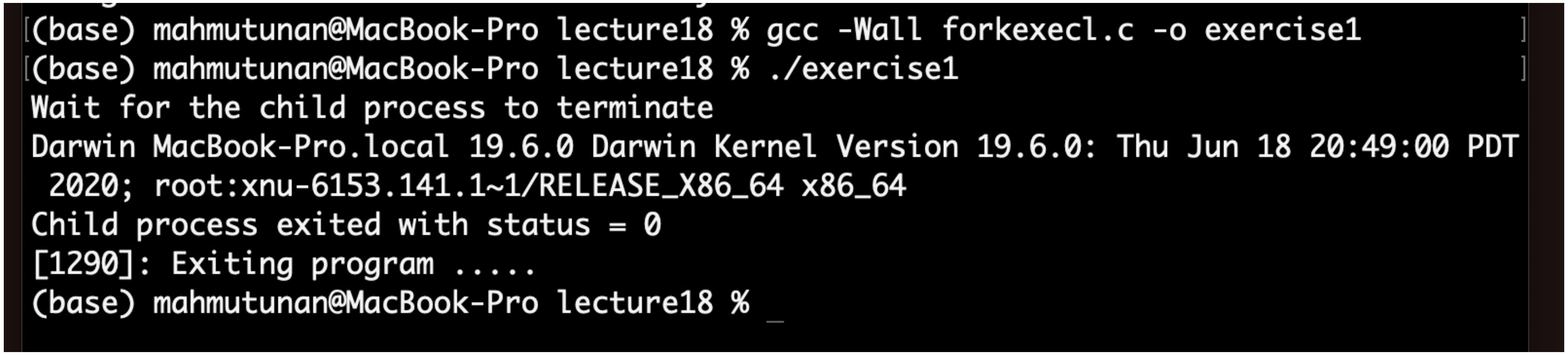


```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

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

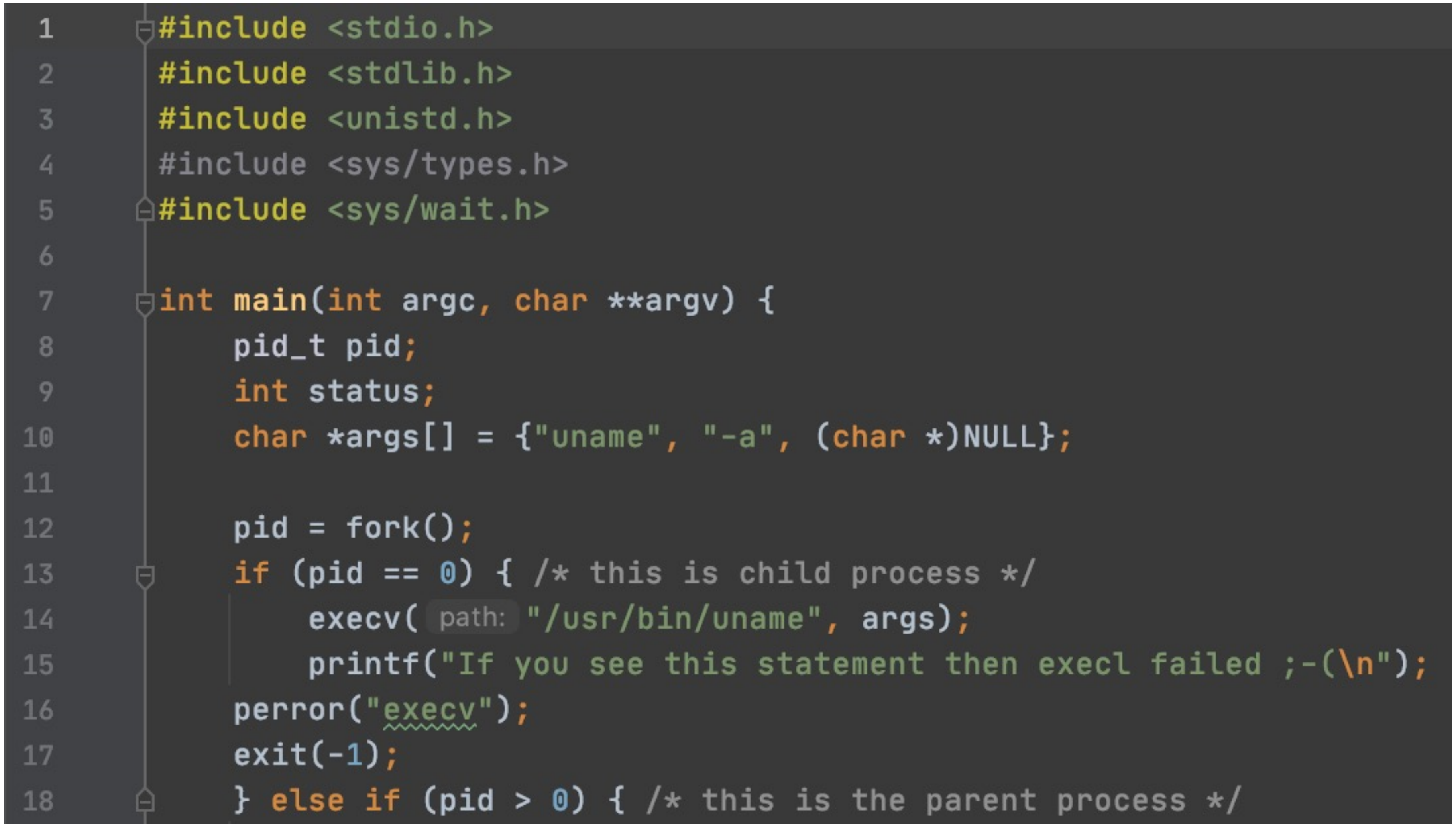


```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall forkexecl.c -o exercise1
(base) mahmutunan@MacBook-Pro lecture18 % ./exercise1
Wait for the child process to terminate
Darwin MacBook-Pro.local 19.6.0 Darwin Kernel Version 19.6.0: Thu Jun 18 20:49:00 PDT
 2020; root:xnu-6153.141.1~1/RELEASE_X86_64 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)

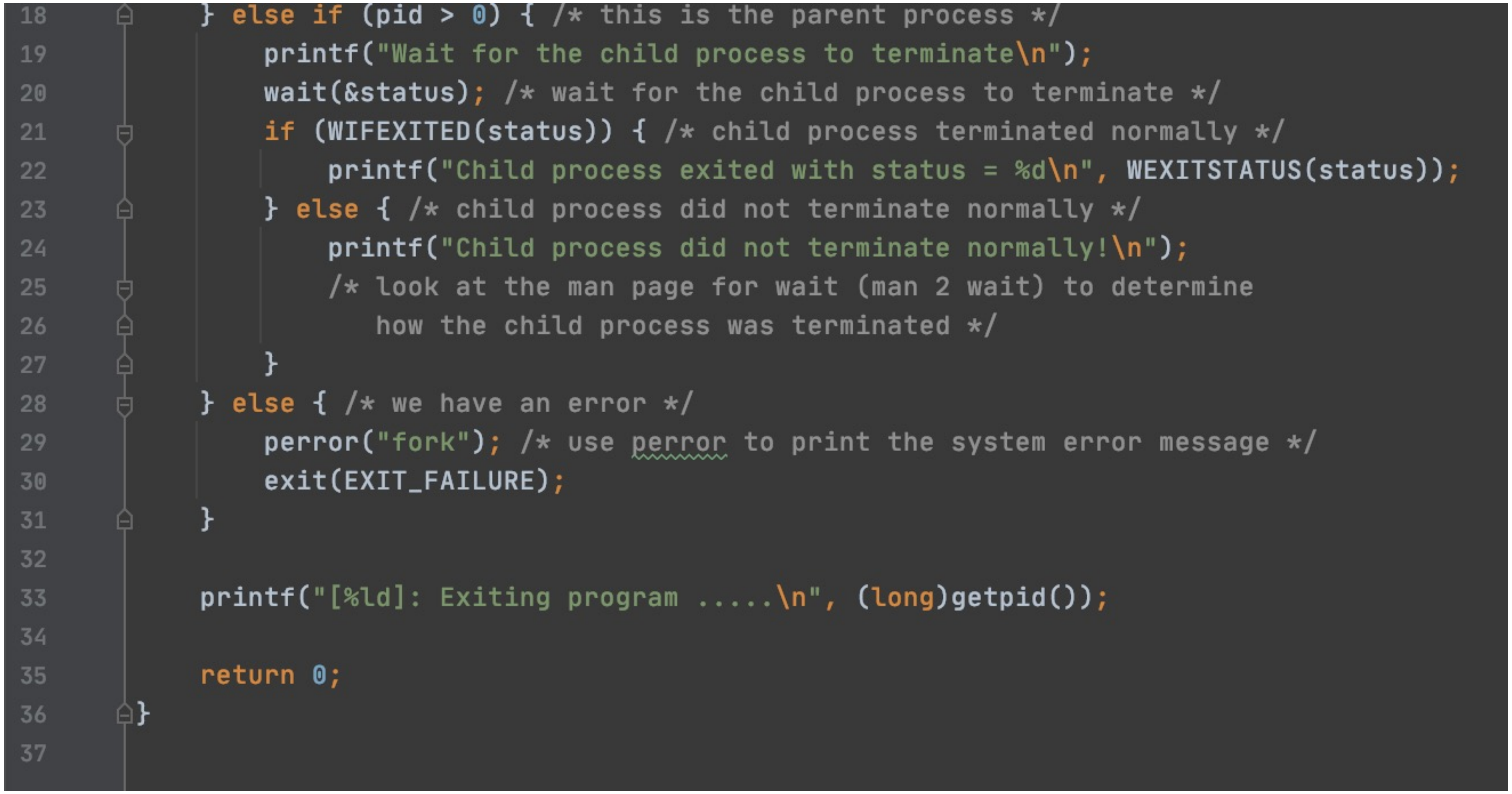
- Functions that have *p* in the name use *filename* as the first argument while functions without *p* use the *pathname* as the first argument. If the filename contains a slash character (/), it is considered as a pathname, otherwise, all directories specified by the PATH environment variable are searched 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



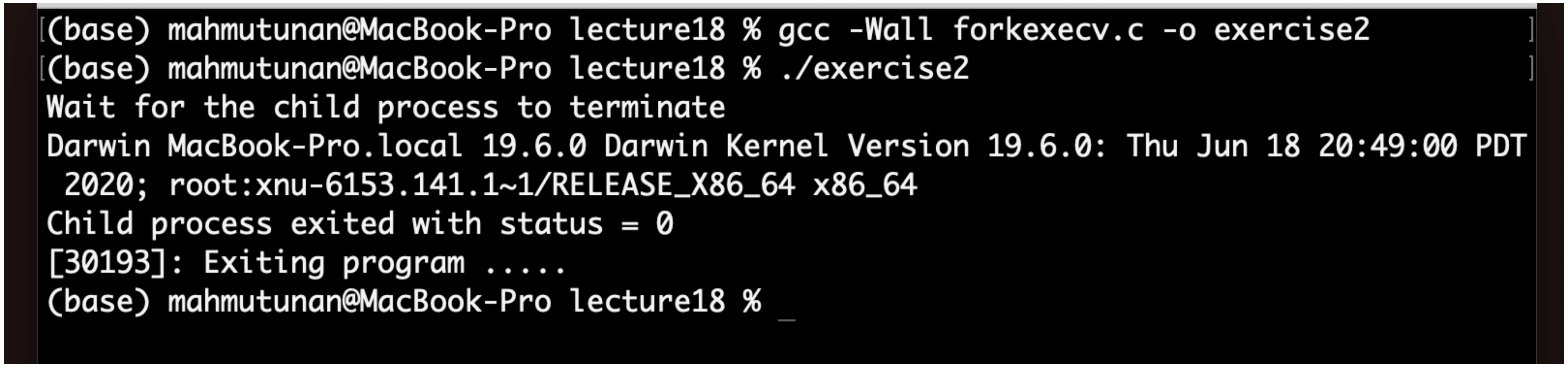
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

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

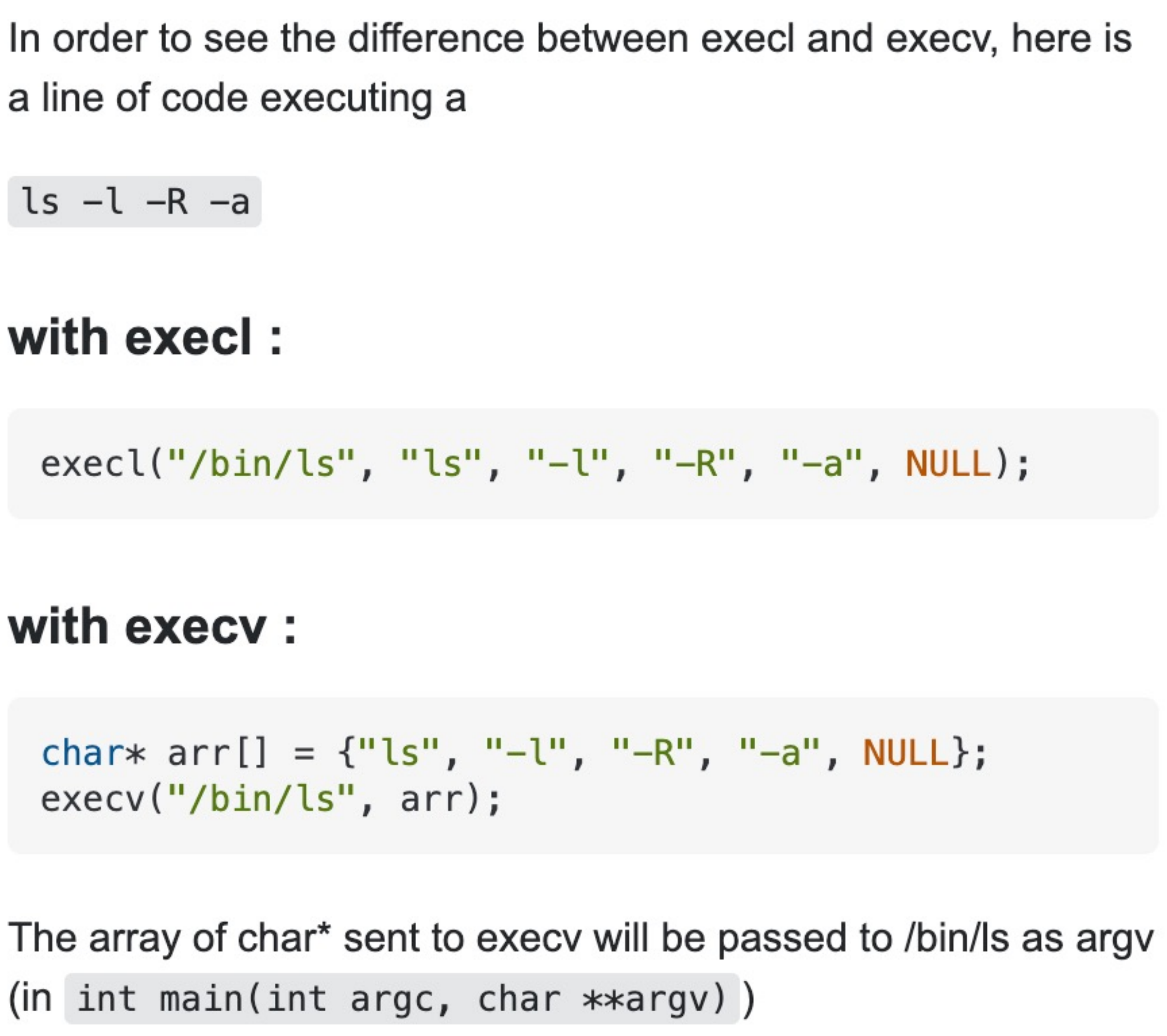


```
    } else if (pid > 0) { /* this is the parent process */
        printf("Wait for the child process to terminate\n");
        wait(&status); /* wait for the child process to terminate */
        if (WIFEXITED(status)) { /* child process terminated normally */
            printf("Child process exited 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 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 :

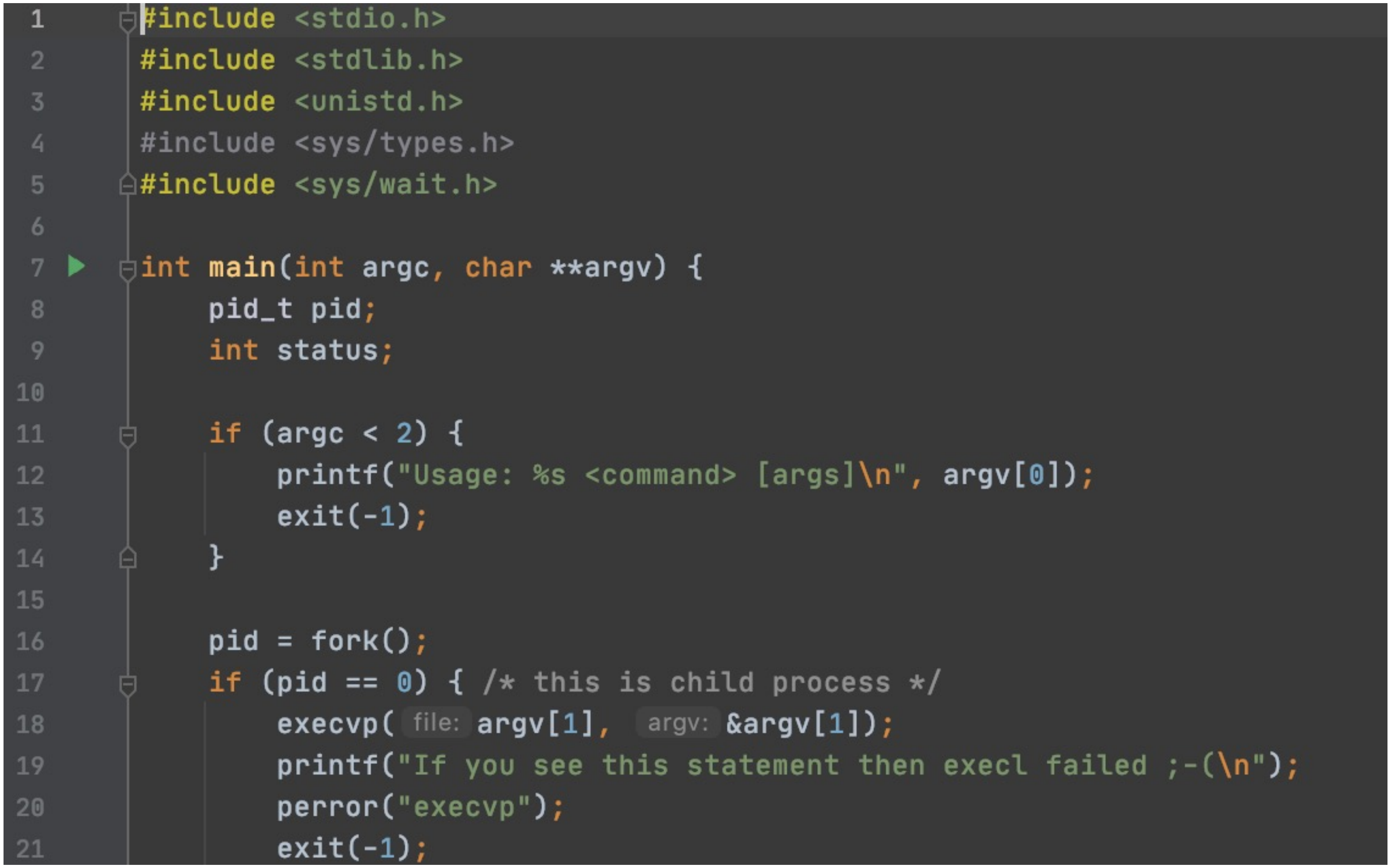
```
execl("/bin/ls", "ls", "-l", "-R", "-a", NULL);
```

with execv :

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

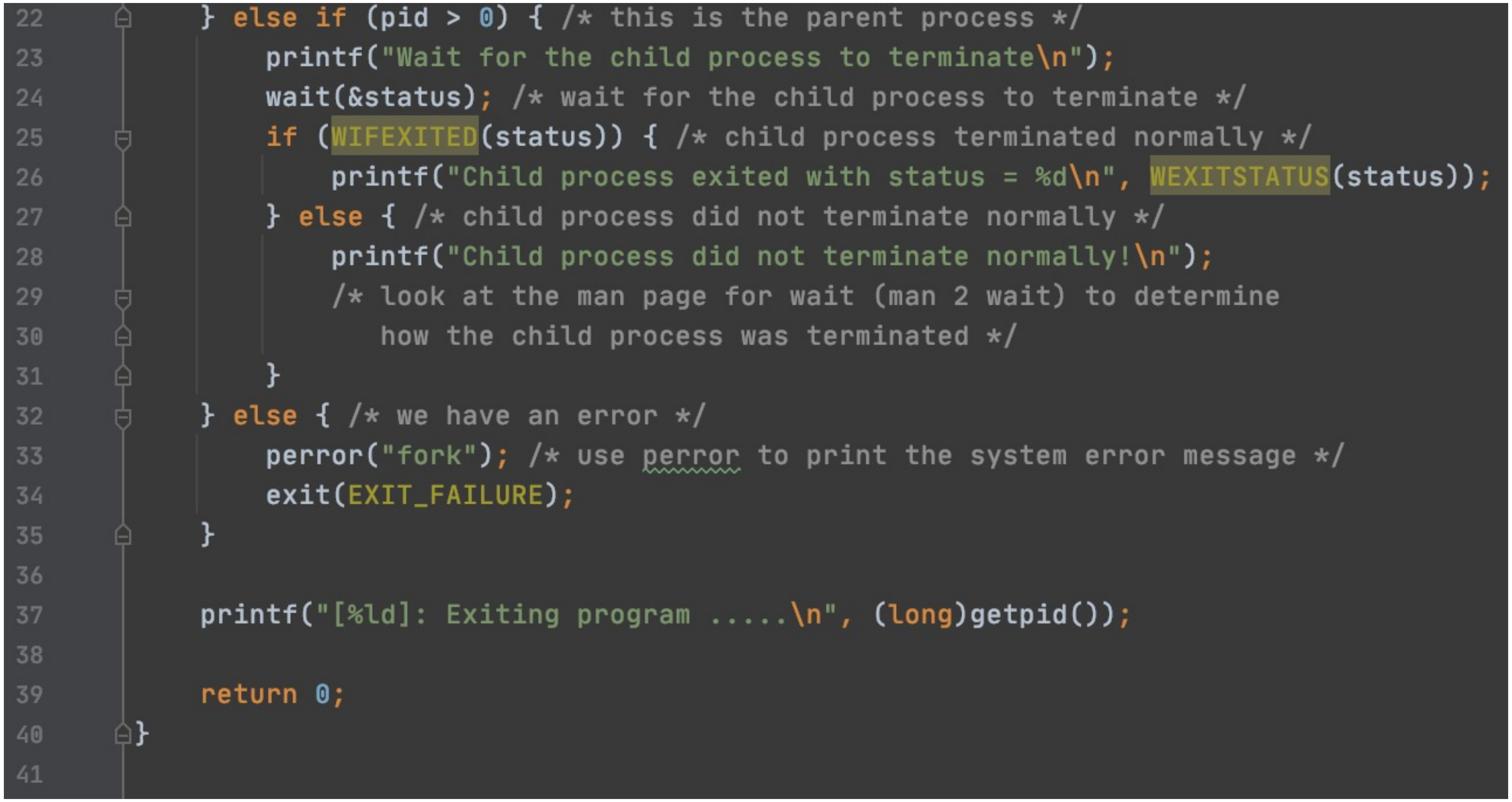
The array of char* sent to execv will be passed to /bin/ls as argv (in `int main(int argc, char **argv)`)

forkexecvp.c



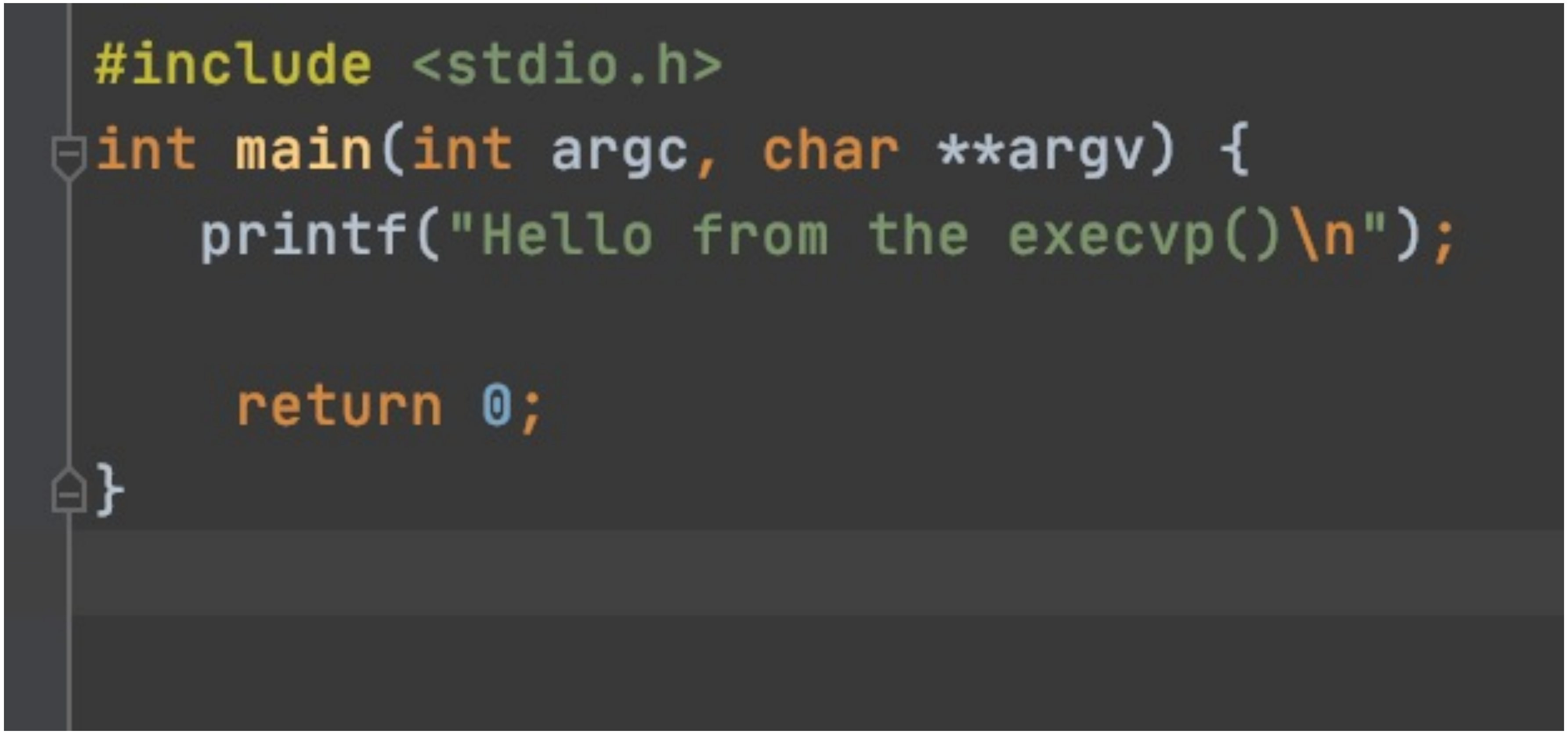
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

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



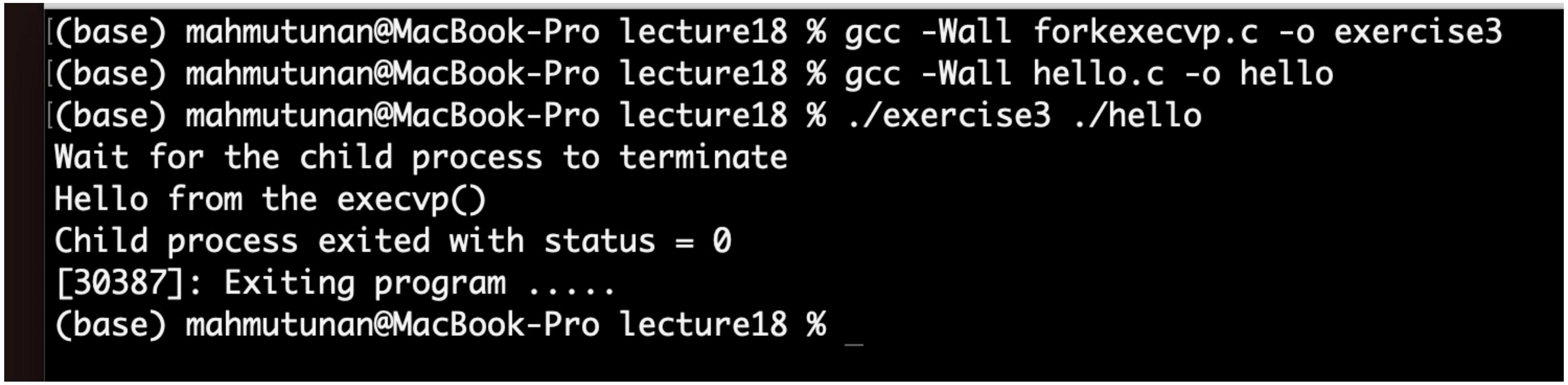
```c
    } else if (pid > 0) { /* this is the parent process */
        printf("Wait for the child process to terminate\n");
        wait(&status); /* wait for the child process to terminate */
        if (WIFEXITED(status)) { /* child process terminated normally */
            printf("Child process 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;
}
```

hello.c



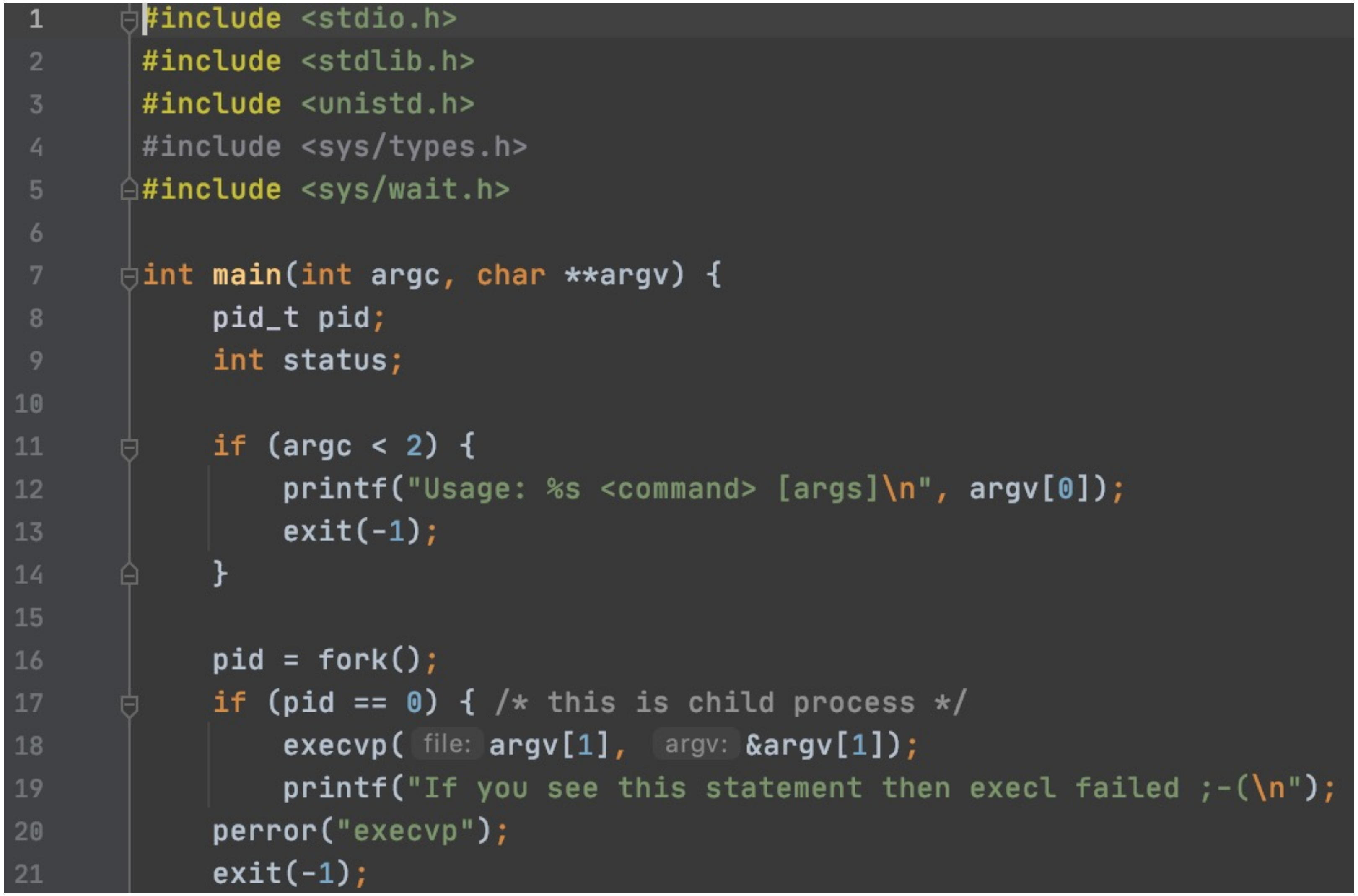
```c
#include <stdio.h>
int main(int argc, char **argv) {
    printf("Hello from the execvp()\n");

     return 0;
}
```

compile & run

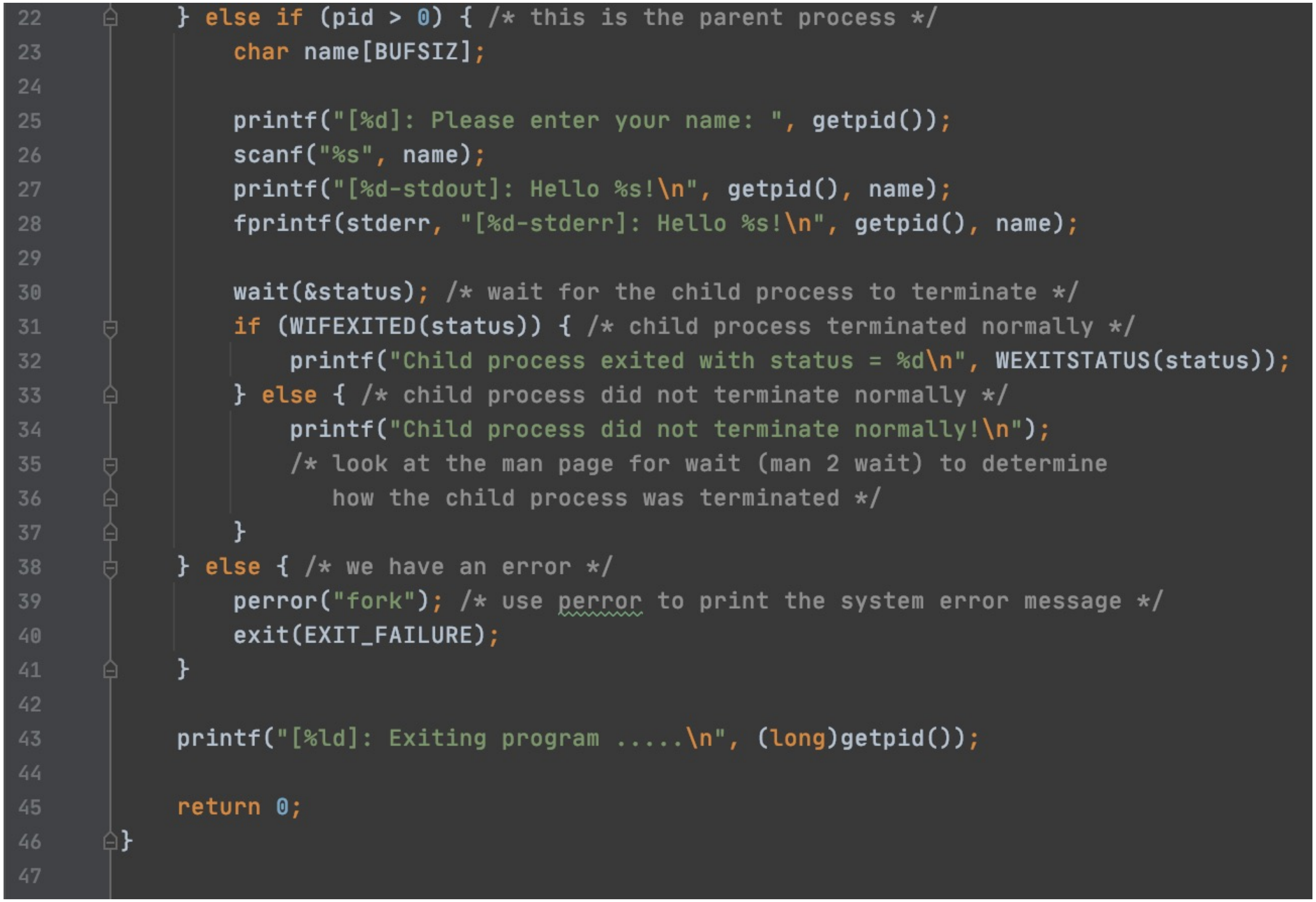


```
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall forkexecvp.c -o exercise3
(base) mahmutunan@MacBook-Pro lecture18 % gcc -Wall hello.c -o hello
(base) mahmutunan@MacBook-Pro lecture18 % ./exercise3 ./hello
Wait for the child process to terminate
Hello from the execvp()
Child process exited with 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;
}

```