OSTEP Chapter 5

ECE 3600, Fall 2022

Table of Contents

1. p1.c fork() example
2. p2.c fork() and wait() example
3. p3.c fork() and execvp() example
4. p4.c fork() and I/O redirect

1. p1.c fork() example

after fork(), parent or child could be scheduled next

```
$ make
gcc -o p1 p1.c -Wall
gcc -o p2 p2.c -Wall
gcc -o p3 p3.c -Wall
gcc -o p4 p4.c -Wall
$ cat -n ostep/code/cpu-api/p1.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4
     5  int
     6  main(int argc, char *argv[])
     7  {
     8      printf("hello world (pid:%d)\n", (int) getpid());
     9      int rc = fork();
    10      if (rc < 0) {
    11          // fork failed; exit
    12          fprintf(stderr, "fork failed\n");
    13          exit(1);
    14      } else if (rc == 0) {
    15          // child (new process)
    16          printf("hello, I am child (pid:%d)\n", (int) getpid());
    17      } else {
    18          // parent goes down this path (original process)
    19          printf("hello, I am parent of %d (pid:%d)\n",
    20                 rc, (int) getpid());
    21      }
    22      return 0;
    23  }
$ ./p1
hello world (pid:21918)
hello, I am parent of 21919 (pid:21918)
hello, I am child (pid:21919)
$ ./p1
hello world (pid:21920)
hello, I am child (pid:21921)
hello, I am parent of 21921 (pid:21920)
$
```

2. p2.c fork() and wait() example

parent waits for child to finish first

```
$ cat -n ostep/code/cpu-api/p2.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <sys/wait.h>
     5
     6  int
     7  main(int argc, char *argv[])
     8  {
     9      printf("hello world (pid:%d)\n", (int) getpid());
    10      int rc = fork();
    11      if (rc < 0) {
    12          // fork failed; exit
    13          fprintf(stderr, "fork failed\n");
    14          exit(1);
    15      } else if (rc == 0) {
    16          // child (new process)
    17          printf("hello, I am child (pid:%d)\n", (int) getpid());
    18          sleep(1);
    19      } else {
    20          // parent goes down this path (original process)
    21          int wc = wait(NULL);
    22          printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
    23                 rc, wc, (int) getpid());
    24      }
    25      return 0;
    26  }
$ ./p2
hello world (pid:21937)
hello, I am child (pid:21938)
hello, I am parent of 21938 (wc:21938) (pid:21937)
$ ./p2
hello world (pid:21939)
hello, I am child (pid:21940)
hello, I am parent of 21940 (wc:21940) (pid:21939)
$
```

--> try checking the child exit status

3. p3.c fork() and execvp() example

child runs wc

```
$ cat -n ostep/code/cpu-api/p3.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <string.h>
     5  #include <sys/wait.h>
     6
     7  int
     8  main(int argc, char *argv[])
     9  {
    10      printf("hello world (pid:%d)\n", (int) getpid());
    11      int rc = fork();
    12      if (rc < 0) {
    13          // fork failed; exit
    14          fprintf(stderr, "fork failed\n");
    15          exit(1);
    16      } else if (rc == 0) {
    17          // child (new process)
    18          printf("hello, I am child (pid:%d)\n", (int) getpid());
    19          char *myargs[3];
    20          myargs[0] = strdup("wc");   // program: "wc" (word count)
    21          myargs[1] = strdup("p3.c"); // argument: file to count
    22          myargs[2] = NULL;           // marks end of array
    23          execvp(myargs[0], myargs);  // runs word count
    24          printf("this shouldn't print out");
    25      } else {
    26          // parent goes down this path (original process)
    27          int wc = wait(NULL);
    28          printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
    29                 rc, wc, (int) getpid());
    30      }
    31      return 0;
    32  }
$ ./p3
hello world (pid:22206)
hello, I am child (pid:22207)
 32 123 966 p3.c
hello, I am parent of 22207 (wc:22207) (pid:22206)
$
```

4. p4.c fork() and I/O redirect

child redirects stdout to file p4.output then runs wc

```
$ cat -n ostep/code/cpu-api/p4.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <string.h>
     5  #include <fcntl.h>
     6  #include <assert.h>
     7  #include <sys/wait.h>
     8
     9  int
    10  main(int argc, char *argv[])
    11  {
    12      int rc = fork();
    13      if (rc < 0) {
    14          // fork failed; exit
    15          fprintf(stderr, "fork failed\n");
    16          exit(1);
    17      } else if (rc == 0) {
    18          // child: redirect standard output to a file
    19          close(STDOUT_FILENO);
    20          open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
    21
    22          // now exec "wc"...
    23          char *myargs[3];
    24          myargs[0] = strdup("wc");   // program: "wc" (word count)
    25          myargs[1] = strdup("p4.c"); // argument: file to count
    26          myargs[2] = NULL;           // marks end of array
    27          execvp(myargs[0], myargs);  // runs word count
    28      } else {
    29          // parent goes down this path (original process)
    30          int wc = wait(NULL);
    31          assert(wc >= 0);
    32      }
    33      return 0;
    34  }
$ ./p4
$ cat p4.output
 34 114 884 p4.c
$
```