Universidade Estadual de Campinas
Instituto de Computação
MC504 Sistemas Operacionais

T02

CPU Virtualization
The Process API

Arthur João Catto, PhD

2º semestre de 2018

Essential Questions

- Here we discuss process management in Unix systems, in an attempt at answering two questions
  - What interfaces should the OS present for process creation and control?
  - How should these interfaces be designed to enable ease of use as well as utility?

Process creation

The `fork()` system call (`p1.c`)

```c
int main(int argc, char *argv[]) {
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    printf("this is process %d my rc is %d\n", (int)getpid(), rc);
    if (rc < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {
        // parent goes down this path (original process)
        printf("hello, I am parent of %d (pid:%d)\n",
            rc, (int) getpid());
    }
    return 0;
}
```

The `fork()` system call (`p1.c`)

- Read the docs to find out how `fork` works.
- Compile and run `p1.c`. The snapshot below is the result I got.

```
hello world (pid:1395)
this is process 1395 my rc is 1396
hello, I am parent of 1396 (pid:1395)
this is process 1396 my rc is 0
hello, I am child (pid:1396)
```

- Did you get something similar?
- Does `fork` seem to be working as you expected?
- Can you write down a trace of `p1`'s execution?
- Is that the only output that `p1` might have produced? Could it be longer? Could the lines appear in another sequence?

Waiting
for process termination

The `wait()` system call (`p2.c`)

```
6   int main(int argc, char *argv[]) {
7       printf("hello world (pid:%d)\n", (int) getpid());
8       int rc = fork();
9       if (rc < 0) {
10          // fork failed; exit
11          fprintf(stderr, "fork failed\n");
12          exit(1);
13      } else if (rc == 0) {
14          // child (new process)
15          printf("hello, I am child (pid:%d)\n", (int) getpid());
16          sleep(3);
17      } else {
18          // parent goes down this path (original process)
19          int wc = wait(NULL);
20          printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
21              rc, wc, (int) getpid());
22      }
23      return 0;
24  }
```

The `wait()` system call (`p2.c`)

- Read the docs to learn about `sleep` and `wait`.
- Compile and run `p2.c`. The snapshot below is what I got as a result.

```
hello world (pid:1474)
hello, I am child (pid:1475)
hello, I am parent of 1475 (wc:1475) (pid:1474)
```

- Did you get something similar?
- Do `sleep` and `wait` seem to be working as expected?
- Can you write down a trace of `p2`'s execution?
- Is that the only output that `p2` might have produced? Could it be longer? Could the lines appear in another sequence?

Running
an external program

The execvp() system call (p3.c)

```c
int main(int argc, char *argv[]) {
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {
        // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");   // program: "wc" (word count)
        myargs[1] = strdup("p3.c"); // argument: file to count
        myargs[2] = NULL;           // marks end of array
        execvp(myargs[0], myargs);  // runs word count
        printf("this shouldn't print out");
    } else {
        // parent goes down this path (original process)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

The `wait()` system call (`p3.c`)

- Read the docs to learn about `execvp`.
- Compile and run `p3.c`. The snapshot below is what I got as a result.

```
hello world (pid:1697)
hello, I am child (pid:1698)
      30     123     966 p3.c
hello, I am parent of 1698 (wc:1698) (pid:1697)
```

- Did you get something similar?
- Does `execvp` seem to be working as expected?
- Can you write down a trace of `p3`'s execution?
- Is that the only output that `p3` might have produced? Could it be longer? Could the lines appear in another sequence?

Redirecting
the output of a process

The `close()` and `open()` system calls (`p4.c`)

```
 8  int main(int argc, char *argv[]) {
 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: redirect standard output to a file
16      close(STDOUT_FILENO);
17      open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
18
19      // now exec "wc"...
20          char *myargs[3];
21          myargs[0] = strdup("wc");   // program: "wc" (word count)
22          myargs[1] = strdup("p4.c"); // argument: file to count
23          myargs[2] = NULL;           // marks end of array
24          execvp(myargs[0], myargs);  // runs word count
25      } else {
26          // parent goes down this path (original process)
27          int wc = wait(NULL);
28      }
29      return 0;
30  }
```

Redirecting program output (p4.c)

- Read the docs to learn about `close`, `open` and `cat`.
- Compile and run `p4.c`. The snapshot below is what I got as a result.

```
arthur.catto$ ./p4
arthur.catto$
```

- Did you get something similar? Where has `p4`'s output gone?
- Does `p4` seem to be working as expected?
- Now type `cat p4.output` at the command prompt. This is what I got...

```
SUP080:nb180808 arthur.catto$ ./p4
SUP080:nb180808 arthur.catto$ cat p4.output
      30     109     845 p4.c
```

- Is that what you would expect? Is that the only possible output?