# 5. Interlude: Process API

**Operating System: Three Easy Pieces**

# The fork() System Call

□ Create a new process

 ◆ The newly-created process has its own copy of the **address space**,
 **registers**, and **PC**.

**p1.c**

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

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());
    } else {                 // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

# Calling fork() example (Cont.)

**Result (Not deterministic)**

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

# The wait() System Call

❑ This system call won't return until the child has run and exited.

**p2.c**

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

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());
    } else {                  // parent goes down this path (main)
        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 (Cont.)

**Result (Deterministic)**

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

# The exec() System Call

- Run a program that is different from the calling program

**p3.c**

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

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
        …
```

**p3.c (Cont.)**

```
…
        execvp(myargs[0], myargs); // runs word count
        printf("this shouldn't print out");
    } else {                            // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

**Result**

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

# All of the above with redirection

**p4.c**

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

int
main(int argc, char *argv[]){
    int rc = fork();
    if (rc < 0) {           // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
        …
```

**p4.c**

```c
        …
        // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc");           // program: "wc" (word count)
        myargs[1] = strdup("p4.c");         // argument: file to count
        myargs[2] = NULL;                   // marks end of array
        execvp(myargs[0], myargs);          // runs word count
    } else {                                // parent goes down this path (main)
        int wc = wait(NULL);
    }
    return 0;
}
```

**Result**

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```

- Disclaimer: This lecture slide set is used in AOS course in University of Cantabria. Was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book  written by Remzi and Andrea Arpaci-Dusseau (at University of Wisconsin)