

CS 3305A

Process

Lecture 4

Sept 20 2023

Copyright Notice:

1. Lecture slides are being provided with permission from the copyright for in-class (CS3305A) use only. The slides must not be reproduced or provided to anyone outside of the class.
2. All download copies of the slides and/or lecture recordings are for personal use only.
3. Students must destroy these copies within 30 days after receipt of final course evaluations

Topics to be discussed

- ❑ Fork and Files
 - ❑ Process File Descriptor Table

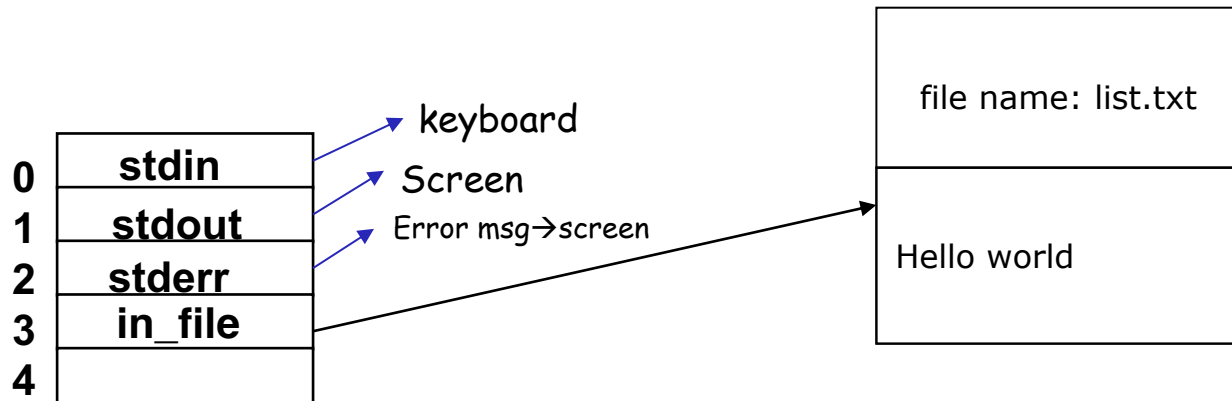
Fork and File

- ❑ Every process has a process file descriptor table
 - ❑ Each entry in process file descriptor table represents stdin, stdout, stderr, and file pointer.

Process File Descriptor Table

Assume that there was something like this in a program

```
int in_file;  
in_file = open("list.txt", O_RDONLY);
```



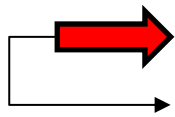
Process File Descriptor table

Fork and Files

- ❑ Open a file before a fork
 - ❑ The child process gets a copy of its parent's process file descriptor table.
 - ❑ The child and parent **share a file pointer** because the **open** came before the **fork**.
- ❑ Open a file after a fork
 - ❑ Assume that parent and child each open a file after the fork
 - ❑ They get their own entries in the file descriptor table
 - ❑ This implies that the **file position information is different**

fdopen() before fork

```
int main()
{
    int fd;  char c;  pid_t pid;
    fd = open("hello.txt", O_RDONLY);
    pid = fork ();
    if (pid > 0)
    {
        read(fd, &c, 1);
        printf("parent: c = %c\n", c);
        wait(NULL);
    }
    else if (pid == 0)
    {
        read(fd, &c, 1);
        printf("child: c = %c\n", c);
    }
    return 0;
}
```



Open a file before fork()

Parent File Descriptor table

0	stdin
1	stdout
2	stderr
3	fd
4	

hello.txt

Hello world

Child File Descriptor table

0	stdin
1	stdout
2	stderr
3	fd
4	

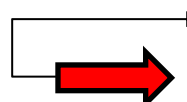
The parent and child point to the same file offset

Fork and Files

- ❑ Open a file after a fork
 - ❑ Assume that parent and child each open a file after the fork
 - ❑ They get their own entries
 - ❑ This implies that the file position information is different

fopen() after fork

```
int main()
{
    int fd;  char c;  pid_t pid;
    pid = fork ();
    fd = open("hello.txt", O_RDONLY);
    if (pid > 0)
    {
        read(fd, &c, 1);
        printf("parent: c = %c\n", c);
        wait(NULL);
    }
    else if (pid == 0)
    {
        read(fd, &c, 1);
        printf("child: c = %c\n", c);
    }
    return 0;
}
```



Open a file after fork()

Parent File Descriptor table

0	stdin
1	stdout
2	stderr
3	fd
4	

hello.txt

Hello world

Child File Descriptor table

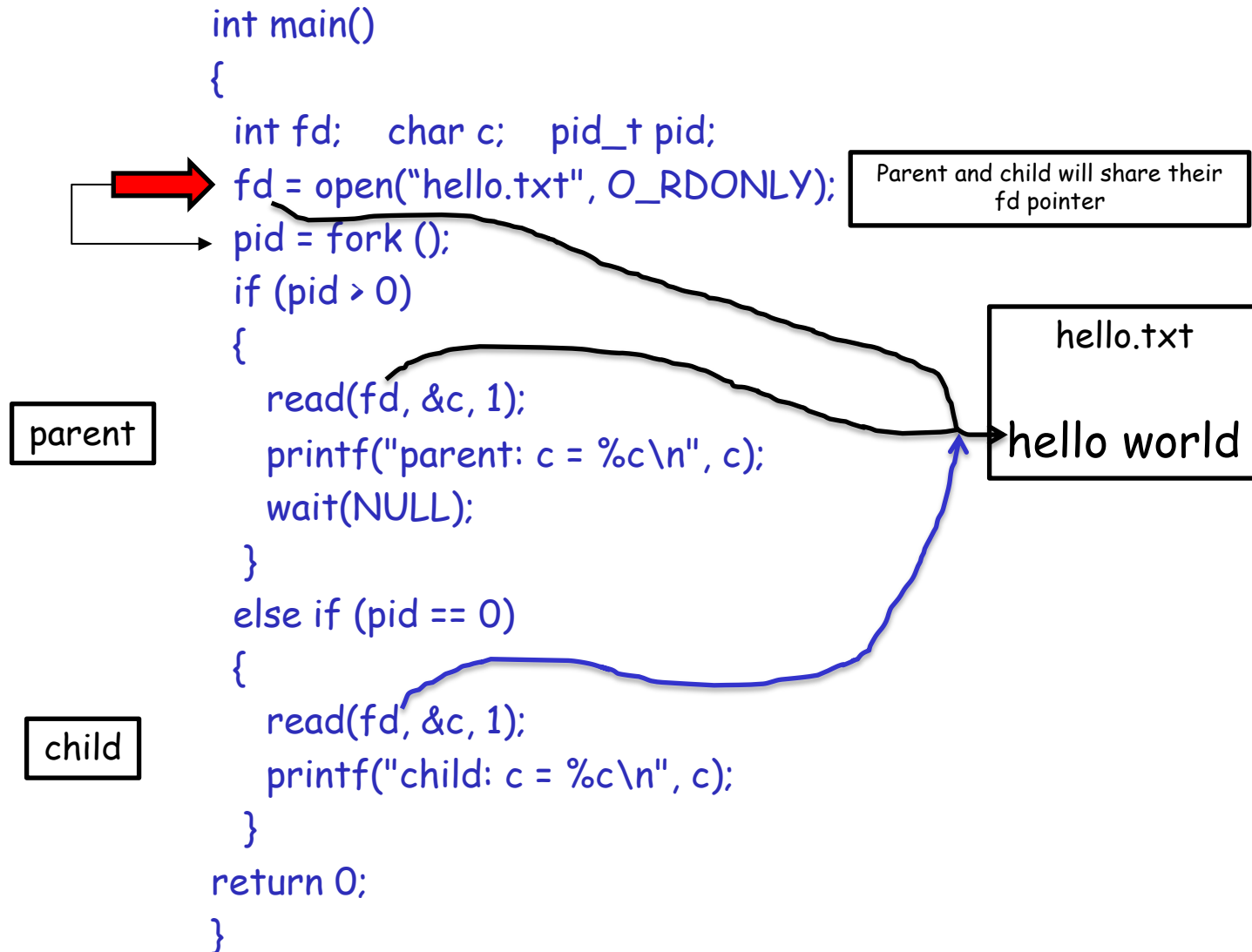
0	stdin
1	stdout
2	stderr
3	fd
4	

The parent and child has their own individual entries in the system file descriptor table

Question

- ❑ Suppose that `hello.txt` consists of `hello world how are you`. Then what is the output of the following programs?
 - ❑ `open()` before `fork()`
 - ❑ `open()` after `fork()`

fopen() before fork



fopen() after fork

```
int main()
{
    int fd;  char c;  pid_t pid;
    pid = fork ();
    fd = open("hello.txt", O_RDONLY);
    if (pid > 0)
    {
        read(fd, &c, 1);
        printf("parent: c = %c\n", c);
        wait(NULL);
    }
    else if (pid == 0)
    {
        read(fd, &c, 1);
        printf("child: c = %c\n", c);
    }
    return 0;
}
```

Parent and child will have their individual fd pointing to file separately (no sharing)

parent

child

hello.txt

hello world

Fork and Files

- ❑ Be careful
 - ❑ It is much better to open files after forks!