Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

Операційні системи

Лабораторна №10

Виконав:
Студент групи ФБ-82
Козачок Вячеслав
Перевірив:
Кіреєнко О.В.

Київ - 2020

Варіант 9

Нехай процес, який відкрив N файлів, перед породженням процесу-нащадка за допомогою системного виклику `fork()` закриває K<N файлів. Процес-нащадок відразу після породження закриває M<N-K файлів і через деякий час завершується (в цей час процес-предок очікує його завершення). Розробити програму, яка демонструвала б динаміку зміни даних в системі керування введенням-виведенням ОС Linux (таблиці відкритих файлів і масиви файлових дескрипторів процесів). Наприклад, сценарій програми може бути таким:

1. відкриття процесом-предком стандартних файлів введення-виведення і чотирьох призначених для користувача файлів для зчитування;
2. закриття процесом-предком двох призначених для користувача файлів;
3. процес-предок породжує процес, який успадковує таблиці файлів і відкритих файлів процесу-предка;
4. завершується процес-нащадок. Після кожного з етапів друкуються таблиці відкритих файлів і масиви файлових дескрипторів для обох процесів.

Code

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

#define FILES 8

int * fd = new int[FILES];


void print_descriptors(int * fd, int print_offset=1)
{
    for(int i = 0; i < print_offset; i++) printf(" ");

    int opened_files = 0;
    for(int i = 0; i < FILES; i++)
    {
        struct stat buff;
        if (fstat(fd[i], &buff) == -1)
            continue;
        else
        {
            opened_files++;
            printf("%d ", fd[i]);
        }
    }
    printf("\n");
    for(int i = 0; i < print_offset; i++) printf(" ");
    printf("Total opened files:%d\n\n", opened_files);
}

int main() {
    // Array of file descriptors
    fd[0] = open("/dev/stdin", O_RDWR);
    fd[1] = open("/dev/stdout", O_RDWR);
    fd[2] = open("/dev/stderr", O_RDWR);
    fd[3] = open("./files/dum.txt", O_RDWR);
    fd[4] = open("./files/rob.txt", O_RDWR);
    fd[5] = open("./files/ann.txt", O_RDWR);
    fd[6] = open("./files/kate.txt", O_RDWR);
    fd[7] = open("./files/file.txt", O_RDWR);
```

```
45
46
47      printf(" - PID:%d All opened descriptors:\n", getpid());
48      print_descriptors(fd);
49
50      printf(" - PID:%d Closing some descriptors ... \n", getpid());
51      close(fd[2]);
52      close(fd[3]);
53
54
55      printf(" - PID:%d Show descriptors again:\n", getpid());
56      print_descriptors(fd);
57
58      printf(" - PID:%d Creating Child Process...\n", getpid());
59      pid_t pID = fork();
60
61      switch(pID)
62      {
63          case 0:
64              // Child process section
65              printf(" - - PID:%d Child process showing descriptors...\n", getpid());
66              print_descriptors(fd, 5);
67
68              printf(" - - PID:%d Child process closing some descriptors...\n", getpid());
69              close(fd[6]);
70              close(fd[5]);
71
72              printf(" - - PID:%d Show descriptors in child process after closing:\n",
73               getpid());
74              print_descriptors(fd, 5);
75
76              printf(" - - PID:%d Child process EXITING...\n", getpid());
77              break;
78
79          case -1:
80              // Error Section
81              printf("Something bad happened when creating process\n");
82              perror("fork");
83              break;
84
85          default:
86              // Parent process section
87              wait(&pID);
88              printf(" - PID:%d Parent process waited for child and now prints descriptors
    again\n", getpid());
89              print_descriptors(fd);
90              printf(" - PID:%d Parent process EXITING...\n", getpid());
91              break;
92      }
93
94
95  }
```

Output

```
- PID:43563 All opened descriptors:
3 4 5 6 7 8 9 10
Total opened files:8

- PID:43563 Closing some descriptors ...
- PID:43563 Show descriptors again:
3 4 7 8 9 10
Total opened files:6

- PID:43563 Creating Child Process...
- - PID:43564 Child process showing descriptors...
    3 4 7 8 9 10
    Total opened files:6

- - PID:43564 Child process closing some descriptors...
- - PID:43564 Show descriptors in child process after closing:
    3 4 7 10
    Total opened files:4

- - PID:43564 Child process EXITING...
- PID:43563 Parent process waited for child and now prints descriptors again
3 4 7 8 9 10
Total opened files:6

- PID:43563 Parent process EXITING...
```

Висновки

В цій лабораторній роботі навчився працювати з файловими дескрипторами. Відкривати їх, закривати дочірнім процессом, але як ми можемо бачити, коли ми закриваємо дочірнім процессом дескриптор, він залишається відкритим у батьківському. Так виходить тому що до дочірнього процесу потрапляють лише копії дискрипторів, і тому у батьківському вони залишаються.

У функції `print_descriptors` ми виводимо тільки відкриті дескриптори, адже якщо `fstat()` повертає -1 то файл не відкритий і дескриптор є не дійсним і тому ми не виводимо дескриптор, а якщо 0, то ми можемо працювати з файлом, отже виводимо дескриптор. Це і реалізовано у функції.