Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Кудаева В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 24.12.24

Москва, 2024

Постановка задачи

Вариант 16:

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия file с таким именем на запись. Родительский процесс принимает от пользователя строки произвольной длины. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в консоль выводится информация об ошибке. Правило проверки: строка должна оканчиваться на «.» или «;».

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс;
- int shm_open(const char *name, int oflag, mode_t mode) - создает или открывает объект разделяемой памяти;
- int write(int fd, const void* buff, int count) – записывает по дескриптору fd count байт из buff;
- void exit(int number) – вызывает нормальное завершение программы с кодом number;
- int ftruncate(int fd, off_t length) - устанавливает размер объекта разделяемой памяти;
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) - отображает объект разделяемой памяти в адресное пространство процесса;
- sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value) - создает или открывает именованный семафор;
- int sem_wait(sem_t *sem) - уменьшает значение семафора на 1. Если семафор уже равен 0, процесс блокируется;
- int sem_post(sem_t *sem) - увеличивает значение семафора на 1. Разблокирует процесс, ожидающий семафора;
- int sem_close(sem_t *sem) - закрывает именованный семафор в текущем процессе;
- int sem_unlink(const char *name) - удаляет именованный семафор из системы;
- int execl(char* path, const char* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
- int close(int fd); – закрывает дескриптор fd;
- pid_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.

В файле parent.c создается родительский и дочерний процессы. Пользователь вводит строку в консоль. Если строка равна "exit", родитель записывает это слово в разделяемую память, уведомляет дочерний процесс через sem_post(sem_read) и завершает цикл. Если строка не равна "exit", родитель копирует строку в разделяемую память. Затем он вызывает sem_post(sem_read), чтобы разрешить дочернему процессу прочитать данные. Дочерний процесс ждет (sem_wait(sem_read)), пока родитель не заполнит память, и только после этого читает данные. После этого дочерний процесс проверяет строку на валидность и если строка корректна, то выводит её в файл, в противном случае выводит сообщение об ошибке в консоль. Затем уведомляет (sem_post(sem_write)) родителя о том, что данные обработаны. Родительский процесс ждет завершения обработки данных, прежде чем записать новые. Таким образом, семафор sem_read обеспечивает, что дочерний процесс не будет читать данные из разделяемой памяти до тех пор, пока родительский процесс не заполнит их, а семафор sem_write обеспечивает, что родительский процесс не запишет новые данные в разделяемую память, пока дочерний процесс не обработает текущие данные.

Код программы

parent.c

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

#define BUFFER_SIZE 1024
#define SHM_NAME "/shared_memory"
#define SEM_WRITE_NAME "/sem_write"
#define SEM_READ_NAME "/sem_read"
#define SHM_SIZE sizeof(char) * 1024

int main() {
    int shm_fd;
    char *shm_ptr;
    sem_t *sem_write, *sem_read;
    pid_t pid;
    char buffer[BUFFER_SIZE];
    char filename[256];

```

```c
    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (shm_fd == -1) {
        perror("Error creating shared memory\n");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("Error resizing shared memory\n");
        exit(EXIT_FAILURE);
    }

    shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Error mapping shared memory\n");
        exit(EXIT_FAILURE);
    }

    sem_write = sem_open(SEM_WRITE_NAME, O_CREAT, S_IRUSR | S_IWUSR, 0);
    sem_read = sem_open(SEM_READ_NAME, O_CREAT, S_IRUSR | S_IWUSR, 1);

    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        perror("Error creating semaphore\n");
        exit(EXIT_FAILURE);
    }

    write(STDOUT_FILENO, "Enter the file name to record: ", 32);
    ssize_t len = read(STDIN_FILENO, filename, sizeof(filename));
    if (filename[len - 1] == '\n') {
        filename[len - 1] = '\0';
    }
```

```c

    pid = fork();

    if (pid < 0) {
        perror("Fork failed\n");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        close(shm_fd);

        sem_wait(sem_read);
        execl("./child", "child", filename, NULL);
        perror("Execl failed\n");
        exit(EXIT_FAILURE);
    } else {
        close(shm_fd);

        write(STDOUT_FILENO, "Enter a line (or 'exit' to exit): ", 35);
        while ((len = read(STDIN_FILENO, buffer, BUFFER_SIZE)) > 0) {
            if (buffer[len - 1] == '\n') {
                buffer[len - 1] = '\0';
            }
            if (strcmp(buffer, "exit") == 0) {

```

```
79
80                  memcpy(shm_ptr, "exit", 5);
81                  sem_post(sem_read);
82                  break;
83              }
84
85              memcpy(shm_ptr, buffer, strlen(buffer) + 1);
86              sem_post(sem_read);
87              sem_wait(sem_write);
88          }
89
90          sem_post(sem_read);
91          sem_close(sem_write);
92          sem_close(sem_read);
93          sem_unlink(SEM_WRITE_NAME);
94          sem_unlink(SEM_READ_NAME);
95
96          munmap(shm_ptr, SHM_SIZE);
97          shm_unlink(SHM_NAME);
98
99          wait(NULL);
100         exit(EXIT_SUCCESS);
101     }
102
103     return 0;
104 }
```

child.c

```
1   #include <stdlib.h>
2   #include <string.h>
3   #include <unistd.h>
4   #include <stdio.h>
5   #include <stdbool.h>
6   #include <sys/mman.h>
7   #include <fcntl.h>
8   #include <semaphore.h>
9   #include <sys/stat.h>
10
11  #define BUFFER_SIZE 1024
12  #define SHM_NAME "/shared_memory"
13  #define SEM_WRITE_NAME "/sem_write"
14  #define SEM_READ_NAME "/sem_read"
15  #define SHM_SIZE sizeof(char) * 1024
16
17  bool validate_string(const char *str) {
18      size_t len = strlen(str);
19      if (len == 0) return false;
20      return (str[len - 1] == '.' || str[len - 1] == ';');
21  }
22
23  int main(int argc, char *argv[]) {
24      if (argc < 2) {
25          perror("Invalid number of arguments\n");
26          exit(EXIT_FAILURE);
27      }
```

```c

    int shm_fd;
    char *shm_ptr;
    sem_t *sem_write, *sem_read;
    char buffer[BUFFER_SIZE];
    char filename[256];

    int fd = open(argv[1], O_WRONLY | O_TRUNC | O_CREAT | O_APPEND, S_IRUSR | S_IWUSR);
        if (fd == -1) {
            perror("Error open input file\n");
            exit(EXIT_FAILURE);
        }

    shm_fd = shm_open(SHM_NAME, O_RDONLY, 0);
    if (shm_fd == -1) {
        perror("Error creating shared memory\n");
        exit(EXIT_FAILURE);
    }

    shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Error mapping shared memory\n");
        exit(EXIT_FAILURE);
    }

    sem_write = sem_open(SEM_WRITE_NAME, 0);
    sem_read = sem_open(SEM_READ_NAME, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        perror("Error creating semaphore\n");
        exit(EXIT_FAILURE);
    }
```

```c

     char error_message[] = "Invalid string\n";

     while (1) {
         sem_wait(sem_read);

         if (strcmp(shm_ptr, "exit") == 0) {
             break;
         }

         if (validate_string(shm_ptr)) {

             write(fd, shm_ptr, strlen(shm_ptr));
             write(fd, "\n", 1);

         } else {
             write(STDOUT_FILENO, error_message, sizeof(error_message) - 1);
         }

         sem_post(sem_write);
     }

```

```
81      close(fd);
82      sem_close(sem_write);
83      sem_close(sem_read);
84      munmap(shm_ptr, SHM_SIZE);
85      shm_unlink(SHM_NAME);
86      exit(EXIT_SUCCESS);
87  }
```

Протокол работы программы

Тестирование

```
Enter the file name to record: file1.txt
Enter a line (or 'exit' to exit): ewgrv;
474j8GH%-=^
Error: Invalid string
sfg84jdifh;
+==3de.
afca00000adca7yh3JJJJKN
Error: Invalid string
exit
```

file1.txt

```
1   ewgrv;
2   sfg84jdifh;
3   +==3de.
```

Вывод strace

execve("./parent", ["./parent"], 0x7ffee35fd0c0 /* 73 vars */) = 0

brk(NULL) = 0x55e0c4363000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd47f40af0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7fb776086000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

newfstatat(AT_FDCWD,
"/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy",
{st_mode=S_IFDIR|0755, st_size=4096, ...}, 0) = 0

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=27331, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 27331, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb77607f000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

mprotect(0x7fb7760c0000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7fb77607f000, 27331) = 0

openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|
O_NOFOLLOW|O_CLOEXEC, 0600) = 3

ftruncate(3, 1024) = 0

mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) =
0x7fb7760bf000

openat(AT_FDCWD, "/dev/shm/sem.sem_write", O_RDWR|O_NOFOLLOW) = 4

newfstatat(4, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0

getrandom("\xa3\xc5\x56\x94\xb6\x05\xf1\x5f", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55e0c4363000

brk(0x55e0c4384000) = 0x55e0c4384000

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) =
0x7fb776085000

close(4) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_read", O_RDWR|O_NOFOLLOW) = 4

newfstatat(4, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) =
0x7fb776084000

close(4) = 0

write(1, "Enter the file name to record: \0", 32Enter the file name to record:) = 32

read(0, file1.txt

"file1.txt\n", 256) = 10

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fb775e53a10) = 52936

close(3) = 0

write(1, "Enter a line (or 'exit' to exit)"..., 35Enter a line (or 'exit' to exit):) = 35

read(0, dhdjde

"dhdjde\n", 1024) = 7

futex(0x7fb776084000, FUTEX_WAKE, 1) = 1

read(0, dnjdjd;

"dnjdjd;\n", 1024) = 8

futex(0x7fb776084000, FUTEX_WAKE, 1) = 1

read(0, exit

"exit\n", 1024) = 5

futex(0x7fb776084000, FUTEX_WAKE, 1) = 1

futex(0x7fb776084000, FUTEX_WAKE, 1) = 0

munmap(0x7fb776085000, 32) = 0

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=52936,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

munmap(0x7fb776084000, 32) = 0

unlink("/dev/shm/sem.sem_write") = 0

unlink("/dev/shm/sem.sem_read") = 0

munmap(0x7fb7760bf000, 1024) = 0

unlink("/dev/shm/shared_memory") = 0

wait4(-1, NULL, 0, NULL) = 52936

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы мне удалось реализовать взаимодействие между родительским и дочерним процессами с использованием разделяемой памяти и семафоров для организации межпроцессного взаимодействия. Мною были изучены механизмы управления доступом к разделяемым ресурсам для предотвращения гонок данных. В данной работе потребовалось использование двух семафоров для обеспечения корректного порядка выполнения операций чтения и записи между процессами.