

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-213Б-23

Студент: Арсельгов А. Б.

Преподаватель: Бахарев В. Д.

Оценка: _____

Дата: 12.11.24

Москва, 2024

Постановка задачи

Вариант 21.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 21) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- fork – создает дочерний процесс.
- Wait - Ожидает завершения дочернего процесса.
- Exec1 - Загружает и выполняет новый процесс.
- sem_unlink - Удаляет именованный семафор.
- shm_unlink - Удаляет объект разделяемой памяти.
- Open - Открывает файл.
- Close - Закрывает файловый дескриптор.
- Read - Читает данные из файла или стандартного ввода.
- Write - Пишет данные в файл или в стандартный вывод.
- sem_wait - Уменьшает значение семафора (блокирует процесс до получения сигнала).
- sem_post - Увеличивает значение семафора (освобождение ресурса).
- sem_open - Открывает (или создает) именованный семафор.
- Mmap - Отображает файл или разделяемую память в адресное пространство процесса.
- shm_open - Открывает (или создает) объект разделяемой памяти.
- Ftruncate - Устанавливает размер файла, создавая или обрезаю его.

В этой лабораторной работе была реализована программа, которая использует разделяемую память и семафоры для взаимодействия между процессами. Родительский процесс записывает строки в разделяемую память, а два дочерних процесса инвертируют эти строки и записывают их в файлы.

Код программы

parent.c

```
#include <string.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/types.h>
```

```
#define BUFFER_SIZE 1024
```

```
void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}
```

```
int main() {
    char buffer[BUFFER_SIZE];
    int count = 1;
    char filename1[BUFFER_SIZE], filename2[BUFFER_SIZE];
```

```
    int shmid1 = shmget(IPC_PRIVATE, BUFFER_SIZE, IPC_CREAT | 0666);
    if (shmid1 == -1) {
        write(2, "shmget failed for child 1", strlen("shmget failed for child 1"));
        exit(EXIT_FAILURE);
    }
    int shmid2 = shmget(IPC_PRIVATE, BUFFER_SIZE, IPC_CREAT | 0666);
    if (shmid2 == -1) {
        write(2, "shmget failed for child 2", strlen("shmget failed for child 2"));
        exit(EXIT_FAILURE);
    }
```

```
    char *shared_memory1 = (char *)shmat(shmid1, NULL, 0);
    if (shared_memory1 == (char *)-1) {
        write(2, "shmat failed for child 1", strlen("shmat failed for child 1"));
        exit(EXIT_FAILURE);
    }
```

```
    char *shared_memory2 = (char *)shmat(shmid2, NULL, 0);
    if (shared_memory2 == (char *)-1) {
        write(2, "shmat failed for child 2", strlen("shmat failed for child 2"));
        exit(EXIT_FAILURE);
    }
```

```
    write(1, "Enter a filename for child1: ", strlen("Enter a filename for child1: "));
    read(0, filename1, BUFFER_SIZE);
    filename1[strcspn(filename1, "\n")] = '\0';
```

```
    write(1, "Enter a filename for child2: ", strlen("Enter a filename for child2: "));
```

```
read(0, filename2, BUFFER_SIZE);
filename2[strcspn(filename2, "\n")] = '\0';
```

```
pid_t pid1 = fork();
if (pid1 == -1) {
write(2, "Error creating child 1", strlen("Error creating child 1"));
exit(EXIT_FAILURE);
}
```

```
if (pid1 == 0) {
int fd1 = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd1 == -1) {
write(2, "Error opening file for child 1", strlen("Error opening file for child 1"));
exit(EXIT_FAILURE);
}
```

```
while (1) {
if (strlen(shared_memory1) > 0) {
reverse_string(shared_memory1);
write(fd1, shared_memory1, strlen(shared_memory1));
write(fd1, "\n", 1);
memset(shared_memory1, 0, BUFFER_SIZE);
}
usleep(100000);
}
```

```
close(fd1);
exit(EXIT_SUCCESS);
}
```

```
pid_t pid2 = fork();
if (pid2 == -1) {
write(2, "Error creating child 2", strlen("Error creating child 2"));
exit(EXIT_FAILURE);
}
```

```
if (pid2 == 0) {
int fd2 = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0644);
if (fd2 == -1) {
write(2, "Error opening file for child 2", strlen("Error opening file for child 2"));
exit(EXIT_FAILURE);
}
```

```
while (1) {
if (strlen(shared_memory2) > 0) {
reverse_string(shared_memory2);
write(fd2, shared_memory2, strlen(shared_memory2));
write(fd2, "\n", 1);
memset(shared_memory2, 0, BUFFER_SIZE);
}
usleep(100000);
}
```

```

close(fd2);
exit(EXIT_SUCCESS);
}

while (1) {
write(1, "Enter the line: ", 16);
ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
if (bytes_read <= 0) {
break;
}
buffer[bytes_read - 1] = '\0';

if (strlen(buffer) == 0) {
break;
}

if (count % 2 == 1) {
strncpy(shared_memory1, buffer, BUFFER_SIZE);
} else {
strncpy(shared_memory2, buffer, BUFFER_SIZE);
}

if (count % 2 == 1) {
write(1, "Sending to child1\n", 18);
} else {
write(1, "Sending to child2\n", 18);
}

count++;
usleep(100000);
}

shmdt(shared_memory1);
shmdt(shared_memory2);
shmctl(shmid1, IPC_RMID, NULL);
shmctl(shmid2, IPC_RMID, NULL);

return 0;
}

#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define BUFFER_SIZE 1024

void reverse_string(char *str) {
int len = strlen(str);
for (int i = 0; i < len / 2; i++) {
char temp = str[i];
str[i] = str[len - 1 - i];

```

```
str[len - 1 - i] = temp;
}
}
```

```
int main() {
char buffer[BUFFER_SIZE];
```

```
while (1) {
ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
if (bytes_read <= 0) {
exit(EXIT_SUCCESS);
}
buffer[bytes_read - 1] = '\0';
```

```
if (strlen(buffer) == 0) {
break;
}
```

```
reverse_string(buffer);
write(1, buffer, strlen(buffer));
write(1, "\n", 1);
}
```

```
return 0;
}
```

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
```

```
#define BUFFER_SIZE 1024
```

```
void reverse_string(char *str) {
int len = strlen(str);
for (int i = 0; i < len / 2; i++) {
char temp = str[i];
str[i] = str[len - 1 - i];
str[len - 1 - i] = temp;
}
}
```

```
int main() {
char buffer[BUFFER_SIZE];
```

```
while (1) {
ssize_t bytes_read = read(0, buffer, BUFFER_SIZE);
if (bytes_read <= 0) {
exit(EXIT_SUCCESS);
}
buffer[bytes_read - 1] = '\0';
```

```
if (strlen(buffer) == 0) {
```

```
break;  
}
```

```
reverse_string(buffer);  
write(1, buffer, strlen(buffer));  
write(1, "\n", 1);  
}
```

```
return 0;  
}
```

Протокол работы программы

Тестирование:

```
user@adamarselgov:~/MAI_OS/lab03/src$ gcc parent.c -o parent -lrt  
user@adamarselgov:~/MAI_OS/lab03/src$ gcc child1.c -o child1 -lrt  
user@adamarselgov:~/MAI_OS/lab03/src$ gcc child2.c -o child2 -lrt  
user@adamarselgov:~/MAI_OS/lab03/src$ ./parent  
Enter a filename for child1: f1.txt  
Enter a filename for child2: f2.txt  
Enter the line: Adam  
Sending to child1  
Enter the line: grhyyr  
Sending to child2  
Enter the line: uutnt  
Sending to child1  
Enter the line: tuu  
Sending to child2  
Enter the line: thrtgd  
Sending to child1  
Enter the line: jkiyutyrt  
Sending to child2  
Enter the line:  
user@adamarselgov:~/MAI_OS/lab03/src$  
user@adamarselgov:~/MAI_OS/lab03/src$ gcc parent.c -o parent -lrt  
user@adamarselgov:~/MAI_OS/lab03/src$ gcc child1.c -o child1 -lrt  
^[[Auser@adamarselgov:~/MAI_OS/lab03/src$ gcc child2.c -o child2 -lrt  
user@adamarselgov:~/MAI_OS/lab03/src$ ./parent  
Enter a filename for child1: f1.txt  
Enter a filename for child2: f2.txt  
Enter the line: asdft1  
Sending to child1  
Enter the line: rehyrt2  
Sending to child2  
Enter the line: rhtyrhe3  
Sending to child1
```



```

shmat(1441809, NULL, 0) = 0x724439b5a000
write(1, "Enter a filename for child1: ", 29Enter a filename for child1: ) = 29
read(0,
"\n", 1024) = 1
write(1, "Enter a filename for child2: ", 29Enter a filename for child2: ) = 29
read(0,
"\n", 1024) = 1
child_tidptr=0x724439b44a10 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x724439b44a10) = 21482
Error opening file for child 1
si_status=1, si_utime=0, si_stime=0} --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=21482, si_uid=1000,
= 16 Error opening file for child 2write(1, "Enter the line: ", 16Enter the line: )
is set) read(0, 0x7ffd807fb660, 1024) = ? ERESTARTSYS (To be restarted if SA_RESTART
si_status=1, si_utime=0, si_stime=0} --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=21483, si_uid=1000,
read(0,
"\n", 1024) = 1
shmdt(0x724439b5b000) = 0
shmdt(0x724439b5a000) = 0
shmctl(1441808, IPC_RMID, NULL) = 0
shmctl(1441809, IPC_RMID, NULL) = 0
exit_group(0) = ?
+++ exited with 0 +++

```

Вывод

Программа запрашивает у пользователя два имени файлов и строки, которые необходимо инвертировать. Родительский процесс записывает строки в разделяемую память, а два дочерних процесса инвертируют их и записывают в соответствующие файлы. В результате в каждом файле содержатся инвертированные строки, введенные пользователем.