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-209БВ-24

Студент: Ле Шон Лыонг

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 26.11.25

Москва, 2025

Постановка задачи

Вариант 14.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Общий метод и алгоритм решения

Использованные системные вызовы:

Системные вызовы для работы с отображаемыми файлами (file mapping):

• **open(path, oflag, mode)** — открывает или создаёт обычный файл, который используется как основа (backing file) для отображаемой памяти.
• **ftruncate(fd, length)** — устанавливает размер файла, обеспечивая достаточное место для размещения структуры разделяемой памяти.
• **mmap(addr, length, prot, flags, fd, offset)** — отображает файл в адресное пространство процесса, позволяя нескольким процессам совместно использовать один и тот же участок памяти.
• **munmap(addr, length)** — снимает отображение памяти, освобождая ресурсы.
• **close(fd)** — закрывает файловый дескриптор после того, как файл был отображён.

Системные вызовы для работы с процессами:

• **fork()** — создаёт дочерний процесс, копируя контекст родителя.
• **execl(path, arg0, arg1, ..., NULL)** — заменяет текущий образ процесса новым, загружая исполняемую программу (child1, child2).
• **waitpid(pid, status, options)** — ожидает завершения дочернего процесса и получает его код выхода.
• **exit(status)** — завершает выполнение процесса с указанным кодом.

Системные вызовы для обработки сигналов:

• **sigaction(signo, act, oldact)** — устанавливает новый обработчик сигнала (SIGUSR1, SIGUSR2) для синхронизации процессов.
• **kill(pid, signo)** — отправляет указанный сигнал другому процессу.
• **pause()** — приостанавливает процесс до получения сигнала.

Алгоритм работы программы:

1. Инициализация родительского процесса

• Создаётся файл, который будет использоваться как область совместной памяти всеми процессами.
• Через вызов `ftruncate()` файл расширяется до размера структуры shared memory.
• Файл отображается в адресное пространство родительского процесса через `mmap()`, что позволяет работать с памятью как с обычным массивом.
• Буфер и управляющие поля структуры инициализируются нулями (`stage = 0`).
• Устанавливается обработчик сигнала `SIGUSR2`, который будет использоваться для уведомления родителя о завершении обработки строки дочерними процессами.

2. Создание дочерних процессов

• Через `fork()` создаётся первый дочерний процесс (Child1).
• В дочернем процессе образ заменяется программой `child1` с помощью `execl()`.
• Аналогично создаётся второй дочерний процесс (Child2).
• После запуска каждый дочерний процесс выполняет собственное подключение к общей памяти, вызывая `mmap()` к тому же файлу.
• Child1 и Child2 устанавливают обработчики сигнала `SIGUSR1`, который используется для уведомления о готовности данных.

3. Взаимодействие процессов через разделяемую память и сигналы

Процессы взаимодействуют последовательно в строгой цепочке:
Parent → Child1 → Child2 → Parent.

Работа родительского процесса

• Родитель принимает от пользователя строку произвольной длины.
• Записывает полученную строку в общий буфер `shm->buf`.
• Устанавливает управляющий флаг `stage = 1`, что означает готовность данных для обработки первым дочерним процессом.
• Отправляет сигнал `SIGUSR1` процессу Child1.
• Ожидает сигнал `SIGUSR2`, подтверждающий завершение работы Child1.

Работа Child1

• После получения сигнала `SIGUSR1` Child1 проверяет значение поля `stage`.
• Если `stage == 1`, выполняется перевод строки в **нижний регистр**.
• Результат записывается обратно в общий буфер.
• Устанавливается `stage = 2`, что означает передачу управления Child2.
• Child1 отправляет родителю сигнал `SIGUSR2` для уведомления о завершении обработки.

Работа родителя (вторая фаза)

• После получения `SIGUSR2` родитель устанавливает `stage = 2`.
• Отправляет сигнал `SIGUSR1` процессу Child2, информируя его о готовности данных ко второй фазе обработки.
• Родитель снова ожидает сигнал `SIGUSR2`.

Работа Child2

• После получения сигнала `SIGUSR1` Child2 проверяет значение поля `stage`.
• Если `stage == 2`, выполняется удаление **задвоенных пробелов** (сжатие последовательностей пробелов).
• Обработанная строка записывается обратно в общий буфер.
• Устанавливается `stage = 3`, что означает готовность результата родителю.
• Child2 отправляет родителю `SIGUSR2`.

Получение результата родителем

• Получив сигнал `SIGUSR2`, родитель считывает содержимое `shm->buf`.
• Выводит результат пользователю.
• Далее цикл обработки продолжается для следующей строки.

4. Завершение работы

• После завершения ввода (Ctrl + D) родитель устанавливает управляющий флаг `stage = 4`, означающий команду завершения для дочерних процессов.
• Child1 и Child2 получают `SIGUSR1`, проверяют `stage == 4` и корректно завершают работу.
• Родитель вызывает `waitpid()` для ожидания завершения обоих процессов.
• Освобождаются ресурсы программы:
– снимается отображение памяти через `munmap()`,
– файл закрывается и при необходимости удаляется.

Код программы

parent.c

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>

#include "shared.h"

static shared_data_t *shm = NULL;
static volatile sig_atomic_t sigusr1_flag = 0;

static void child1_sig_handler(int signo) {
    (void)signo;
    sigusr1_flag = 1;
}

// Перевод строки в нижний регистр
```

```c
static void to_lower_inplace(char *s) {
    for (; *s; ++s) {
        *s = (char)tolower((unsigned char)*s);
    }
}

// Подключение к общей памяти
static void attach_shared(void) {
    int fd = open(SHM_FILE, O_RDWR);
    if (fd < 0) {
        perror("child1: open");
        exit(EXIT_FAILURE);
    }

    shm = mmap(NULL, sizeof(shared_data_t),
               PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm == MAP_FAILED) {
        perror("child1: mmap");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
}

int main(void) {
    attach_shared();

    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = child1_sig_handler;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGUSR1, &sa, NULL) < 0) {
        perror("child1: sigaction");
        exit(EXIT_FAILURE);
    }

    pid_t parent_pid = getppid();

    for (;;) {
        while (!sigusr1_flag) pause();
        sigusr1_flag = 0;

        if (shm->stage == 4) {
            break;
        }

        if (shm->stage == 1) {
            to_lower_inplace(shm->buf);
            shm->stage = 2;
            if (kill(parent_pid, SIGUSR2) < 0) {
                perror("child1: kill parent");
                break;
            }
        }
```

```
    }

    munmap(shm, sizeof(*shm));
    return 0;
}
```

child1.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <signal.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>

#include "shared.h"

static shared_data_t *shm = NULL;
static volatile sig_atomic_t sigusr1_flag = 0;

static void child1_sig_handler(int signo) {
    (void)signo;
    sigusr1_flag = 1;
}

// Перевод строки в нижний регистр
static void to_lower_inplace(char *s) {
    for (; *s; ++s) {
        *s = (char)tolower((unsigned char)*s);
    }
}

// Подключение к общей памяти
static void attach_shared(void) {
    int fd = open(SHM_FILE, O_RDWR);
    if (fd < 0) {
        perror("child1: open");
        exit(EXIT_FAILURE);
    }

    shm = mmap(NULL, sizeof(shared_data_t),
               PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm == MAP_FAILED) {
        perror("child1: mmap");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
```

```c
}

int main(void) {
    attach_shared();

    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = child1_sig_handler;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGUSR1, &sa, NULL) < 0) {
        perror("child1: sigaction");
        exit(EXIT_FAILURE);
    }

    pid_t parent_pid = getppid();

    for (;;) {
        while (!sigusr1_flag) pause();
        sigusr1_flag = 0;

        if (shm->stage == 4) {
            break;
        }

        if (shm->stage == 1) {
            to_lower_inplace(shm->buf);
            shm->stage = 2;
            if (kill(parent_pid, SIGUSR2) < 0) {
                perror("child1: kill parent");
                break;
            }
        }
    }

    munmap(shm, sizeof(*shm));
    return 0;
}
```

child2.c

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <signal.h>
#include <string.h>
#include <errno.h>

#include "shared.h"
```

```c
static shared_data_t *shm = NULL;
static volatile sig_atomic_t sigusr1_flag = 0;

static void child2_sig_handler(int signo) {
    (void)signo;
    sigusr1_flag = 1;
}

static void remove_double_spaces_inplace(char *s) {
    char *dst = s;
    int prev_space = 0;
    while (*s) {
        if (*s == ' ') {
            if (!prev_space) {
                *dst++ = *s;
                prev_space = 1;
            }
        } else {
            *dst++ = *s;
            prev_space = 0;
        }
        s++;
    }
    *dst = '\0';
}

static void attach_shared() {
    int fd = open(SHM_FILE, O_RDWR);
    if (fd < 0) {
        perror("child2: open shared file");
        exit(EXIT_FAILURE);
    }

    shm = mmap(NULL, sizeof(shared_data_t),
               PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shm == MAP_FAILED) {
        perror("child2: mmap");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
}

int main(void) {
    attach_shared();

    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = child2_sig_handler;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGUSR1, &sa, NULL) < 0) {
        perror("child2: sigaction");
        exit(EXIT_FAILURE);
    }
```

```c
    pid_t parent_pid = getppid();

    for (;;) {
        while (!sigusr1_flag) pause();
        sigusr1_flag = 0;

        if (shm->stage == 4) {
            break;
        }

        if (shm->stage == 2) {
            remove_double_spaces_inplace(shm->buf);
            shm->stage = 3;
            if (kill(parent_pid, SIGUSR2) < 0) {
                perror("child2: kill parent");
                break;
            }
        }
    }

    munmap(shm, sizeof(*shm));
    return 0;
}
```

Протокол работы программы

Тестирование:

```
oh@DESKTOP-F8Q4L97:/mnt/c/Users/Son/LabsOS/lab3$ ./parent
Enter lines (Ctrl+D to finish):
HeLlO  worLd!
hello world!
```

Strace:

execve("./parent", ["./parent"], 0x7ffc5aab30b8 /* 26 vars */) = 0
brk(NULL) = 0x59e89f8be000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x757482931000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=71671, ...}) = 0
mmap(NULL, 71671, PROT_READ, MAP_PRIVATE, 3, 0) = 0x75748291f000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) =

832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x757482600000
mmap(0x757482628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x757482628000
mmap(0x7574827b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7574827b0000
mmap(0x7574827ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7574827ff000
mmap(0x757482805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x757482805000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x75748291c000
arch_prctl(ARCH_SET_FS, 0x75748291c740) = 0
set_tid_address(0x75748291ca10) = 723
set_robust_list(0x75748291ca20, 24) = 0
rseq(0x75748291d060, 0x20, 0, 0x53053053) = 0
mprotect(0x7574827ff000, 16384, PROT_READ) = 0
mprotect(0x59e86f858000, 4096, PROT_READ) = 0
mprotect(0x757482969000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x75748291f000, 71671) = 0
openat(AT_FDCWD, "shared.bin", O_RDWR|O_CREAT|O_TRUNC, 0600) = 3
ftruncate(3, 1028) = 0
mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x757482930000
close(3) = 0
rt_sigaction(SIGUSR2, {sa_handler=0x59e86f8567b0, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x757482645330}, NULL, 8) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x75748291ca10) = 724
strace: Process 724 attached

[pid 723] **clone(child_stack=NULL,**
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD
<unfinished ...>
[pid 724] set_robust_list(0x75748291ca20, 24) = 0
[pid 724] **execve("./child1", ["child1"], 0x7ffc7788dc08 /* 26 vars */strace: Process
725 attached**
<unfinished ...>
[pid 723] **<... clone resumed>, child_tidptr=0x75748291ca10) = 725**
[pid 725] set_robust_list(0x75748291ca20, 24 <unfinished ...>
[pid 723] **clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=1, tv_nsec=0},**
<unfinished ...>
[pid 725] <... set_robust_list resumed>) = 0
[pid 725] **execve("./child2", ["child2"], 0x7ffc7788dc08 /* 26 vars / <*unfinished ...*>***
[pid 724] <... execve resumed>) = 0
[pid 724] brk(NULL) = 0x56b150021000
[pid 724] **mmap(NULL, 8192, PROT_READ|PROT_WRITE,**
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7c341cbdc000
[pid 724] access("/etc/ld.so.preload", R_OK <unfinished ...>
[pid 725] <... execve resumed>) = 0
[pid 724] <... access resumed>) = -1 ENOENT (No such file or directory)
[pid 725] brk(NULL <unfinished ...>
[pid 724] **openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC**
<unfinished ...>
[pid 725] <... brk resumed>) = 0x651e90620000
[pid 724] <... openat resumed>) = 3
[pid 725] **mmap(NULL, 8192, PROT_READ|PROT_WRITE,**
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>
[pid 724] fstat(3, <unfinished ...>
[pid 725] <... mmap resumed>) = 0x7d68cca3d000
[pid 724] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=71671, ...}) = 0
[pid 725] access("/etc/ld.so.preload", R_OK <unfinished ...>
[pid 724] **mmap(NULL, 71671, PROT_READ, MAP_PRIVATE, 3, 0 <unfinished ...>**
[pid 725] <... access resumed>) = -1 ENOENT (No such file or directory)
[pid 725] **openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC**
<unfinished ...>
[pid 724] <... mmap resumed>) = 0x7c341cbca000
[pid 725] <... openat resumed>) = 3
[pid 724] **close(3 <unfinished ...>**
[pid 725] fstat(3, <unfinished ...>
[pid 724] <... close resumed>) = 0
[pid 725] <... fstat resumed>{st_mode=S_IFREG|0644, st_size=71671, ...}) = 0

[pid 724] **openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>**
[pid 725] **mmap(NULL, 71671, PROT_READ, MAP_PRIVATE, 3, 0 <unfinished ...>**
[pid 724] <... openat resumed>) = 3
[pid 725] <... mmap resumed>) = 0x7d68cca2b000
[pid 724] read(3, <unfinished ...>
[pid 725] **close(3 <unfinished ...>**
[pid 724] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
[pid 725] <... close resumed>) = 0
[pid 724] pread64(3, <unfinished ...>
[pid 725] **openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC <unfinished ...>**
[pid 724] <... pread64 resumed>"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[pid 725] <... openat resumed>) = 3
[pid 724] fstat(3, <unfinished ...>
[pid 725] read(3, <unfinished ...>
[pid 724] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid 725] <... read resumed>"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
[pid 724] pread64(3, <unfinished ...>
[pid 725] pread64(3, <unfinished ...>
[pid 724] <... pread64 resumed>"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[pid 725] <... pread64 resumed>"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[pid 725] fstat(3, <unfinished ...>
[pid 724] **mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>**
[pid 725] <... fstat resumed>{st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid 724] <... mmap resumed>) = 0x7c341c800000
[pid 725] pread64(3, <unfinished ...>
[pid 724] **mmap(0x7c341c828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000 <unfinished ...>**
[pid 725] <... pread64

resumed>"\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
[pid 724] <... mmap resumed>) = 0x7c341c828000
[pid 725] **mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>**
[pid 724] **mmap(0x7c341c9b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cc800000
[pid 724] <... mmap resumed>) = 0x7c341c9b0000
[pid 725] **mmap(0x7d68cc828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000 <unfinished ...>**
[pid 724] **mmap(0x7c341c9ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cc828000
[pid 724] <... mmap resumed>) = 0x7c341c9ff000
[pid 725] **mmap(0x7d68cc9b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000 <unfinished ...>**
[pid 724] **mmap(0x7c341ca05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cc9b0000
[pid 724] <... mmap resumed>) = 0x7c341ca05000
[pid 725] **mmap(0x7d68cc9ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000 <unfinished ...>**
[pid 724] **close(3 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cc9ff000
[pid 724] <... close resumed>) = 0
[pid 725] **mmap(0x7d68cca05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>**
[pid 724] **mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cca05000
[pid 724] <... mmap resumed>) = 0x7c341cbc7000
[pid 725] **close(3 <unfinished ...>**
[pid 724] **arch_prctl(ARCH_SET_FS, 0x7c341cbc7740 <unfinished ...>**
[pid 725] <... close resumed>) = 0
[pid 724] <... arch_prctl resumed>) = 0
[pid 725] **mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>**
[pid 724] set_tid_address(0x7c341cbc7a10 <unfinished ...>
[pid 725] <... mmap resumed>) = 0x7d68cca28000
[pid 724] <... set_tid_address resumed>) = 724

[pid 725] **arch_prctl(ARCH_SET_FS, 0x7d68cca28740 <unfinished ...>**
[pid 724] set_robust_list(0x7c341cbc7a20, 24 <unfinished ...>
[pid 725] <... arch_prctl resumed>) = 0
[pid 724] <... set_robust_list resumed>) = 0
[pid 725] set_tid_address(0x7d68cca28a10 <unfinished ...>
[pid 724] rseq(0x7c341cbc8060, 0x20, 0, 0x53053053 <unfinished ...>
[pid 725] <... set_tid_address resumed>) = 725
[pid 724] <... rseq resumed>) = 0
[pid 725] set_robust_list(0x7d68cca28a20, 24 <unfinished ...>
[pid 724] **mprotect(0x7c341c9ff000, 16384, PROT_READ <unfinished ...>**
[pid 725] <... set_robust_list resumed>) = 0
[pid 724] <... mprotect resumed>) = 0
[pid 725] rseq(0x7d68cca29060, 0x20, 0, 0x53053053 <unfinished ...>
[pid 724] mprotect(0x56b1332c4000, 4096, PROT_READ <unfinished ...>
[pid 725] <... rseq resumed>) = 0
[pid 724] <... mprotect resumed>) = 0
[pid 725] **mprotect(0x7d68cc9ff000, 16384, PROT_READ <unfinished ...>**
[pid 724] mprotect(0x7c341cc14000, 8192, PROT_READ <unfinished ...>
[pid 725] <... mprotect resumed>) = 0
[pid 724] <... mprotect resumed>) = 0
[pid 725] mprotect(0x651e552f2000, 4096, PROT_READ <unfinished ...>
[pid 724] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>
[pid 725] <... mprotect resumed>) = 0
[pid 724] <... prlimit64 resumed>{rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 725] mprotect(0x7d68cca75000, 8192, PROT_READ <unfinished ...>
[pid 724] **munmap(0x7c341cbca000, 71671 <unfinished ...>**
[pid 725] <... mprotect resumed>) = 0
[pid 724] <... munmap resumed>) = 0
[pid 725] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>
[pid 724] **openat(AT_FDCWD, "shared.bin", O_RDWR <unfinished ...>**
[pid 725] <... prlimit64 resumed>{rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 725] **munmap(0x7d68cca2b000, 71671) = 0**
[pid 725] **openat(AT_FDCWD, "shared.bin", O_RDWR) = 3**
[pid 724] <... openat resumed>) = 3
[pid 725] **mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0 <unfinished ...>**
[pid 724] **mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0 <unfinished ...>**
[pid 725] <... mmap resumed>) = 0x7d68cca3c000

[pid 724] <... mmap resumed>) = 0x7c341cbdb000
[pid 725] **close(3 <unfinished ...>**
[pid 724] **close(3 <unfinished ...>**
[pid 725] <... close resumed>) = 0
[pid 724] <... close resumed>) = 0
[pid 725] **rt_sigaction(SIGUSR1, {sa_handler=0x651e552f04d0, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7d68cc845330}, <unfinished ...>**
[pid 724] **rt_sigaction(SIGUSR1, {sa_handler=0x56b1332c24c0, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7c341c845330}, <unfinished ...>**
[pid 725] <... rt_sigaction resumed>NULL, 8) = 0
[pid 724] <... rt_sigaction resumed>NULL, 8) = 0
[pid 725] **getppid(<unfinished ...>**
[pid 724] **getppid(<unfinished ...>**
[pid 725] <... getppid resumed>) = 723
[pid 724] <... getppid resumed>) = 723
[pid 725] **pause(<unfinished ...>**
[pid 724] **pause(<unfinished ...>**
[pid 723] <... clock_nanosleep resumed>0x7ffc7788d5b0) = 0
[pid 723] fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
[pid 723] getrandom("\xa7\x96\xe8\x6a\xd1\x46\x99\x70", 8, GRND_NONBLOCK) = 8
[pid 723] brk(NULL) = 0x59e89f8be000
[pid 723] brk(0x59e89f8df000) = 0x59e89f8df000
[pid 723] **write(1, "Enter lines (Ctrl+D to finish):\n", 32) = 32**
[pid 723] fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
[pid 723] **read(0, "HeLlo woRlD!\n", 1024) = 13**
[pid 723] **kill(724, SIGUSR1) = 0**
[pid 724] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 723] **pause(<unfinished ...>**
[pid 724] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=723, si_uid=1000} ---
[pid 724] **rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system call)**
[pid 724] **kill(723, SIGUSR2 <unfinished ...>**
[pid 723] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 724] <... kill resumed>) = 0
[pid 723] --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=724, si_uid=1000} ---
[pid 724] **pause(<unfinished ...>**
[pid 723] **rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system call)**
[pid 723] **kill(725, SIGUSR1) = 0**
[pid 725] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 723] **pause(<unfinished ...>**

[pid 725] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=723, si_uid=1000} ---
[pid 725] **rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system call)**
[pid 725] **kill(723, SIGUSR2) = 0**
[pid 723] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 725] **pause(<unfinished ...>**
[pid 723] --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=725, si_uid=1000} ---
[pid 723] **rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system call)**
[pid 723] **write(1, "hello world!\n", 13) = 13**
[pid 723] **read(0, "", 1024) = 0**
[pid 723] **kill(724, SIGUSR1) = 0**
[pid 724] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 723] **kill(725, SIGUSR1 <unfinished ...>**
[pid 724] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=723, si_uid=1000} ---
[pid 723] <... kill resumed>) = 0
[pid 725] <... pause resumed>) = ? ERESTARTNOHAND (To be restarted if no handler)
[pid 723] **wait4(724, <unfinished ...>**
[pid 724] rt_sigreturn({mask=[]} <unfinished ...>
[pid 725] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=723, si_uid=1000} ---
[pid 724] <... rt_sigreturn resumed>) = -1 EINTR (Interrupted system call)
[pid 725] rt_sigreturn({mask=[]} <unfinished ...>
[pid 724] **munmap(0x7c341cbdb000, 1028 <unfinished ...>**
[pid 725] <... rt_sigreturn resumed>) = -1 EINTR (Interrupted system call)
[pid 724] <... munmap resumed>) = 0
[pid 725] **munmap(0x7d68cca3c000, 1028 <unfinished ...>**
[pid 724] **exit_group(0 <unfinished ...>**
[pid 725] <... munmap resumed>) = 0
[pid 724] <... exit_group resumed>) = ?
[pid 725] **exit_group(0) = ?**
[pid 724] +++ exited with 0 +++
[pid 723] **<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 724**
[pid 725] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=724, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
wait4(725, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 725
munmap(0x757482930000, 1028) = 0

exit_group(0) = ?
+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы были освоены основы межпроцессного взаимодействия с использованием отображаемых файлов (Memory-Mapped Files). Реализован обмен данными между процессами через общий участок памяти, доступный всем участникам конвейера. Сигналы использовались для синхронизации этапов обработки. Работа продемонстрировала практическое применение механизмов совместной памяти и взаимодействия процессов в Linux.