Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование» Дисциплина «Операционные системы»

Лабораторная работа №4

Тема: File mapping

Студент: Будникова В.П.

Группа: М8О-207Б-19

Преподаватель: Миронов Е. С.

Дата:

Оценка:

Москва, 2020

Цель работы: Приобретение практических навыков в освоении принципов работы с файловыми системами и обеспечении обмена данных между процессами посредством технологии «File mapping».

Задача: 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.

Вариант(22): 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 дочерние процессы в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: с вероятностью 80% строки отправляются в дочерний процесс child1, иначе в дочерний процесс child1 Дочерние процессы инвертируют строки.

Лабораторная работа аналогична второй лабораторной работе, только обмен данными между программами осуществляется за счет общей памяти, а также синхронизации за счет мьютексов.

Листинг программы

main.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <pthread.h>
#include <errno.h>
#include <assert.h>

int file(char filename[256]) {
    int f = creat(filename, S_IRWXU | S_IRWXO);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return f;
}

int shmOpen(const char * name) {
    int fd = shm_open(name, O_RDWR | O_CREAT, S_IRWXO | S_IRWXU);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return fd;
}

char * charmmap(int fd) {
    char * f = (char *)mmap(0, sizeof(char), PROT_READ | PROT_WRITE,  MAP_SHARED, fd,
0);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return f;
}
pthread_mutex_t * mutexmmap(int fd) {
    pthread_mutex_t * f = (pthread_mutex_t *)mmap(0, sizeof(pthread_mutex_t), PROT_READ |
PROT_WRITE,  MAP_SHARED, fd, 0);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return f;
}

void mutex(pthread_mutex_t * mutex) {
    pthread_mutexattr_t attr;
    pthread_mutexattr_init(&attr);
    pthread_mutexattr_setpshared(&attr, PTHREAD_PROCESS_SHARED);
    pthread_mutex_init(mutex, &attr);
}
```

```
void lock_mutex(pthread_mutex_t * mutex) {
    int m = pthread_mutex_lock(mutex);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
}
void unlock_mutex(pthread_mutex_t * mutex) {
    int m = pthread_mutex_unlock(mutex);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
}

int main() {
    errno = 0;
    const char * name_inout = "inout";
    const char * mutex_child1 = "mutex1";
    const char * mutex_child2 = "mutex2";
    const char * mutex_parent = "mutex3";
    char filename1[256];
    char filename2[256];
    scanf("%s", filename1);
    scanf("%s", filename2);

    int f1 = file(filename1);
    int f2 = file(filename2);

    int inout = shmOpen(name_inout);
    int mutex_ch1 = shmOpen(mutex_child1);
    int mutex_ch2 = shmOpen(mutex_child2);
    int mutex_par = shmOpen(mutex_parent);

    ftruncate(inout, sizeof(char));
    ftruncate(mutex_ch1, sizeof(pthread_mutex_t));
    ftruncate(mutex_ch2, sizeof(pthread_mutex_t));
    ftruncate(mutex_par, sizeof(pthread_mutex_t));
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }

    char * fd = charmmap(inout);
    pthread_mutex_t * m_ch1 = mutexmmap(mutex_ch1);
    pthread_mutex_t * m_ch2 = mutexmmap(mutex_ch2);
    pthread_mutex_t * m_par = mutexmmap(mutex_par);

    mutex(m_ch1);
    mutex(m_ch2);
    mutex(m_par);
```

```
int id = fork();
if (id == -1) {
    printf("error");
    assert(errno == 0);
} else if (id == 0) {                                          //CHILD1

    dup2(f1, fileno(stdout));
    execl("child1", name_inout, mutex_parent, mutex_child1, NULL);

} else {

    int id2 = fork();
    if (id2 == -1) {
        printf("error");
        assert(errno == 0);
    } else if (id2 == 0) {                                      //CHILD2

        // printf("child2: %d\n", getpid());
        dup2(f2, fileno(stdout));
        execl("child1", name_inout, mutex_parent, mutex_child2, NULL);

    } else {                                                    //PARENT
        // printf("parent: %d\n", getppid());
        char c;
        int f = 1;
        while (!0) {
            if (c == '\n') {
                if (rand() % 9 + 1 < 4) {
                    f = 0;
                } else {
                    f = 1;
                }
            }
            if (f) {
                lock_mutex(m_par);
                if (scanf("%c", &c) <= 0) {
                    fd[0] = '\0';
                    unlock_mutex(m_ch1);
                    lock_mutex(m_par);
                    fd[0] = '\0';
                    unlock_mutex(m_ch2);
                    break;
                }
                fd[0] = c;
                unlock_mutex(m_ch1);
            } else {
                lock_mutex(m_par);
                if (scanf("%c", &c) <= 0) {
                    fd[0] = '\0';
                    unlock_mutex(m_ch1);
                    lock_mutex(m_par);
                    fd[0] = '\0';
```

```
                unlock_mutex(m_ch2);
                break;
            }
            fd[0] = c;
            unlock_mutex(m_ch2);
        }
      }
    }
  }

  int b1 = munmap(fd, sizeof(char));
  int b2 = munmap(m_ch1, sizeof(pthread_mutex_t));
  int b3 = munmap(m_ch2, sizeof(pthread_mutex_t));
  int b4 = munmap(m_par, sizeof(pthread_mutex_t));
  assert(b1 == 0 || b2 == 0 || b3 == 0 || b4 == 0);

  if (close(f1) == -1 || close(f2) == -1) {
    printf("error");
    assert(errno == 0);
  }
      return 0;
}
```

child.c

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <pthread.h>
#include <errno.h>
#include <assert.h>

int shmOpen(const char * name) {
  int fd = shm_open(name, O_RDWR, S_IRWXO | S_IRWXU);
  if (errno != 0) {
    printf("%s\n", strerror(errno));
    assert(errno == 0);
  }
  return fd;
}

char * charmmap(int fd) {
  char * f = (char *)mmap(0, sizeof(char), PROT_READ | PROT_WRITE,  MAP_SHARED, fd,
0);
```

```
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return f;
}

pthread_mutex_t * mutexmmap(int fd) {
    pthread_mutex_t * f = (pthread_mutex_t *)mmap(0, sizeof(pthread_mutex_t), PROT_READ |
PROT_WRITE,  MAP_SHARED, fd, 0);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
    return f;
}

void lock_mutex(pthread_mutex_t * mutex) {
    int m = pthread_mutex_lock(mutex);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
}
void unlock_mutex(pthread_mutex_t * mutex) {
    int m = pthread_mutex_unlock(mutex);
    if (errno != 0) {
        printf("%s\n", strerror(errno));
        assert(errno == 0);
    }
}

void invert(char * str, int size) {
    char c;
    for (int i = 0; i < size / 2 + 1; ++i) {
        c = str[i];
        str[i] = str[size - i];
        str[size - i] = c;
    }
}

int main(int argc, char * argv[]) {
    errno = 0;
    if (argc == 3) {

        int inout = shmOpen(argv[0]);
        int mutex_par = shmOpen(argv[1]);
        int mutex_ch = shmOpen(argv[2]);

        char * fd = charmmap(inout);
        pthread_mutex_t * m_par = mutexmmap(mutex_par);
        pthread_mutex_t * m_ch = mutexmmap(mutex_ch);
```

```
        char c;
        int temp = 64;
        int count = 0;
        char * str;
        str = (char *)malloc(sizeof(char) * temp);

        while (!0) {
            lock_mutex(m_ch);
            c = fd[0];
            if (c == '\0') {
                unlock_mutex(m_par);
                break;
            }
            if (c != '\n') {
                if (count == temp) {
                    temp *= 2;
                    str = (char *)realloc(str, temp);
                }
                str[count] = c;
                count++;
            } else {
                if (count != 0) {
                    str[count] = '\0';
                    invert(str, count - 1);
                    printf("%s\n", str);
                    count = 0;
                }
            }
            unlock_mutex(m_par);
        }
        free(str);
        int a1 = shm_unlink(argv[0]);
        int a2 = shm_unlink(argv[1]);
        int a3 = shm_unlink(argv[2]);

        int b1 = munmap(fd, sizeof(char));
        int b2 = munmap(m_ch, sizeof(pthread_mutex_t));
        int b3 = munmap(m_par, sizeof(pthread_mutex_t));

    } else {
        printf("ошибка аргументов для ребенка");
    }
}
```

Тесты и протокол исполнения

```
Lera:lab4 valeriabudnikova$ cat 1.t
1.txt
2.txt
123456789
asdf
lkjj
dffgh
345
1.txt
2.txt
absde
bcre
ce
dr
ee
123dfg
asdf
lkjj
dffgh
345
Lera:lab4 valeriabudnikova$ make run
./main < 1.t
Lera:lab4 valeriabudnikova$ cat 1.txt
987654321
fdsa
jjkl
543
txt.1
txt.2
edsba
ercb
ee
gfd321
fdsa
543
Lera:lab4 valeriabudnikova$ cat 2.txt
hgffd
ec
rd
jjkl
hgffd
```

Вывод strace

```
strace ./main < 1.t > stracelab4.txt
execve("./main", ["./main"], 0x7fff185f66c0 /* 67 vars */) = 0
brk(NULL)                               = 0x56305a70d000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc94afaf60) = -1 EINVAL (Недопустимый аргумент)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=118993, ...}) = 0
mmap(NULL, 118993, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff05e653000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
```

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff05e651000

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O\305\3743\364B\2216\244\224\306@\261\23\327o"..., 68, 824) = 68

mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff05e62e000

mmap(0x7ff05e635000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7ff05e635000

mmap(0x7ff05e646000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18000) = 0x7ff05e646000

mmap(0x7ff05e64b000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c000) = 0x7ff05e64b000

mmap(0x7ff05e64d000, 13432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff05e64d000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 7\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=40040, ...}) = 0

mmap(NULL, 44000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff05e623000

mprotect(0x7ff05e626000, 24576, PROT_NONE) = 0

mmap(0x7ff05e626000, 16384, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7ff05e626000

mmap(0x7ff05e62a000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0x7ff05e62a000

mmap(0x7ff05e62c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8000) = 0x7ff05e62c000

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\360q\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

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 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

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\363\377?\332\200\270\27\304d\245n\355Y\377\t\334"..., 68, 880) = 68

mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff05e431000

mprotect(0x7ff05e456000, 1847296, PROT_NONE) = 0

mmap(0x7ff05e456000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7ff05e456000

mmap(0x7ff05e5ce000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7ff05e5ce000

mmap(0x7ff05e619000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7ff05e619000

mmap(0x7ff05e61f000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff05e61f000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff05e42e000

arch_prctl(ARCH_SET_FS, 0x7ff05e42e740) = 0

mprotect(0x7ff05e619000, 12288, PROT_READ) = 0

mprotect(0x7ff05e64b000, 4096, PROT_READ) = 0

```
mprotect(0x7ff05e62c000, 4096, PROT_READ) = 0
mprotect(0x563058769000, 4096, PROT_READ) = 0
mprotect(0x7ff05e69e000, 4096, PROT_READ) = 0
munmap(0x7ff05e653000, 118993)          = 0
set_tid_address(0x7ff05e42ea10)         = 53181
set_robust_list(0x7ff05e42ea20, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0x7ff05e635bf0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO,
sa_restorer=0x7ff05e6433c0}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7ff05e635c90, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|
SA_SIGINFO, sa_restorer=0x7ff05e6433c0}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
fstat(0, {st_mode=S_IFREG|0664, st_size=101, ...}) = 0
brk(NULL)                               = 0x56305a70d000
brk(0x56305a72e000)                     = 0x56305a72e000
read(0, "1.txt\n2.txt\n123456789\nasdf\nlkjj\n"..., 4096) = 101
creat("1.txt", 0707)                    = 3
creat("2.txt", 0707)                    = 4
statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=978459, f_bfree=843921, f_bavail=843921,
f_files=978459, f_ffree=978278, f_fsid={val=[0, 0]}, f_namelen=255, f_frsize=4096, f_flags=ST_VALID|
ST_NOSUID|ST_NODEV}) = 0
futex(0x7ff05e650390, FUTEX_WAKE_PRIVATE, 2147483647) = 0
openat(AT_FDCWD, "/dev/shm/inout", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0707) = 5
openat(AT_FDCWD, "/dev/shm/mutex1", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0707) = 6
openat(AT_FDCWD, "/dev/shm/mutex2", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0707) = 7
openat(AT_FDCWD, "/dev/shm/mutex3", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0707) = 8
ftruncate(5, 1)                         = 0
ftruncate(6, 40)                        = 0
ftruncate(7, 40)                        = 0
ftruncate(8, 40)                        = 0
mmap(NULL, 1, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7ff05e69d000
mmap(NULL, 40, PROT_READ|PROT_WRITE, MAP_SHARED, 6, 0) = 0x7ff05e670000
mmap(NULL, 40, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) = 0x7ff05e66f000
mmap(NULL, 40, PROT_READ|PROT_WRITE, MAP_SHARED, 8, 0) = 0x7ff05e66e000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7ff05e42ea10) = 53184
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7ff05e42ea10) = 53185
//далее производится работа с мьютексами.
futex(0x7ff05e66e000, FUTEX_WAIT, 2, NULL) = 0
futex(0x7ff05e66e000, FUTEX_WAIT, 2, NULL) = 0
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66e000, FUTEX_WAIT, 2, NULL) = 0
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66e000, FUTEX_WAIT, 2, NULL) = 0
```

```
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
```

```
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66e000, FUTEX_WAIT, 2, NULL) = 0
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
read(0, "", 4096)                       = 0
futex(0x7ff05e670000, FUTEX_WAKE, 1)    = 1
futex(0x7ff05e66f000, FUTEX_WAKE, 1)    = 1
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=53184, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
munmap(0x7ff05e69d000, 1)               = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=53185, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
munmap(0x7ff05e670000, 40)              = 0
munmap(0x7ff05e66f000, 40)              = 0
munmap(0x7ff05e66e000, 40)              = 0
close(3)                                = 0
close(4)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Strace:

creat - системный вызов для создания файла(возвращает -1, при ошибке). Если файл существует, то creat сделает длину файла равной 0 (очистит файл) для записи в него.

close - системный вызов, который закрывает файловый дескриптор, после этого дескриптор не будет ссылаться на файл и может быть(при необходимости) использован еще раз.

clone - системный вызов для создания нового процесса. Обеспечивает разделяемые адресные пространства, родительский процесс(откуда происходит вызов) не может выполняться в том же стеке, что и дочерний. родительский процесс передает указатель на пространство памяти для дочернего.

openat - системный вызов, который работает аналогично open, но при определенных аргументах меняется относительный путь к файлу(если такой был задан).

ftruncate - системный вызов, первым аргументом которого является файловый дескриптор, вторым - количество байт. Этот системный вызов устанавливает длину файлу, на который ссылается дескриптор в количество байт, поданное вторым аргументом. При успешной работе возвращаемое значение равно 0, иначе возвращается -1.

mmap - системный вызов. Первый аргумент - адрес, второй - количество байт, третий - режим защиты памяти, четвертый - флаг, который назначает тип отражаемого объекта, пятый - файловый дескриптор. Этот системный вызов отражает количество байт из файла, на который ссылается файловый дескриптор, начиная с указанного адреса. При ошибке возвращается -1.

futex - системный вызов, который обеспечивает ожидание изменения указанного адреса в памяти. В моем случае этот системный вызов срабатывает когда при работе с разделяемой памятью я использую мьютексы.

Выводы

В данной лабораторной работе я научилась работать с разделяемой памятью с помощью системных вызовов shm_open, ftruncate и mmap. Также я научилась синхронизировать общую память для разных процессов. При реализации данной программы мне было трудно правильно блокировать и разблокировать мьютексы, так как длина строк может быть любой, передавать в дочерние процессы пришлось по одному символу, надо была следить за тем, чтобы дочерний процесс прочитал символ и потом разблокировать родительский мьютекс, чтобы тот, в свою

очередь мог писать. Также проблемой было понять, как в конечном итоге завершить программу - когда мы считали все данные родительский и дочерние мьютексы нужно разблокировать, эту проблему я решила передачей пустого символа, который служил флагом для разблокировки родительского мьютекса в дочернем процессе и наоборот.

Список литературы

1. Таненбаум Э., Бос Х. *Современные операционные системы. — 4-е изд.* — СПб.: Издательский дом «Питер», 2018. — С. 111 - 123

2. Поисковик Google [электронный ресурс] URL: https://google.com/ (дата обращения: 22.09.2020)