Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Жданович Е.Т.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 11.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем, с использованием shared memory и memory mapping. Для синхронизации чтения и записи из shared memory будем использовать семафор. Родительский процесс создает два дочерних процесса. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. 11 вариант) Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- **`sem_open`**:

  Открывают или создают именованные семафоры. В данном случае, семафоры `SEM_EMPTY` и `SEM_FULL` используются для синхронизации обмена сообщениями между процессами. `SEM_EMPTY` инициализируется значением 1, а `SEM_FULL` — значением 0. Эти значения позволяют процессам обмениваться данными по принципу "производитель-потребитель".
- **`shm_open`**:

  Создает или открывает объект общей памяти с именем `SHM_NAME`. Этот вызов предоставляет дескриптор для дальнейшего обращения к общей памяти.

- **`ftruncate`**:

  Обрезает или изменяет размер объекта общей памяти. В данном случае, размер общей памяти устанавливается на размер структуры `SharedData`, которая включает в себя массив символов для сообщения и флаг для синхронизации.
- **`mmap`**:

  Открывает отображение в память объекта общей памяти, полученного через `shm_open`. Это позволяет процессам читать и записывать в общую память.

- **`read`**:

Считывает данные из стандартного ввода (консоли) в общую память, в массив `shmaddr->message`. В коде предполагается, что пользователь введет сообщение.

- **`fork`**:
  Создают два дочерних процесса. После каждого `fork` программа продолжает выполнение как в родительском процессе, так и в дочернем. Эти дочерние процессы затем выполняют программы `child1` и `child2`, используя `execl`.

- **`execl`**:
  Выполняет программы `child1` и `child2` в дочерних процессах. Эти процессы будут работать с общими семафорами для синхронизации доступа к общей памяти.

- **`sem_post`**:
  Увеличивает значение семафора `sem_empty`. Это уведомляет другие процессы, что можно работать с общей памятью. Этот вызов может быть использован для синхронизации между производителем и потребителем данных.

- **`wait`**:
  Ожидает завершения дочернего процесса. Используется дважды для ожидания завершения `child1` и `child2`.

- **`write`**:
  Пишет финальное сообщение из общей памяти на стандартный вывод (консоль).

- **`sem_close`**:
  Закрывает дескрипторы семафоров после завершения работы с ними.

- **`sem_unlink:`**
  Удаляет именованные семафоры после их использования.

- **`munmap`**:
  Отключает отображение общей памяти, освобождая ресурсы, связанные с ней.

- **`shm_unlink`**:
  Удаляет объект общей памяти, освобождая ресурсы.

Код программы

Parent.c

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <sys/mman.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    #include <semaphore.h>
    #include <unistd.h>
    #include <sys/wait.h>
    #define SHM_NAME "/shared_memory"
    #define SEM_EMPTY "/sem_empty"
    #define SEM_FULL "/sem_full"
    #define BUFFER_SIZE 256
    struct SharedData {
     char message[BUFFER_SIZE];
     int flag;
    };
    int main() {
     sem_t *sem_empty = sem_open(SEM_EMPTY, O_CREAT, 0666, 1);
     sem_t *sem_full = sem_open(SEM_FULL, O_CREAT, 0666, 0);
if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED) {
```

```
perror("sem_open failed");
    exit(EXIT_FAILURE);
}

int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
if (shm_fd == -1) {
    perror("shm_open failed");
    exit(EXIT_FAILURE);
}

if (ftruncate(shm_fd, sizeof(struct SharedData)) == -1) {
    perror("ftruncate failed");
    exit(EXIT_FAILURE);
}

struct SharedData *shmaddr = mmap(NULL, sizeof(struct SharedData),
                                  PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
if (shmaddr == MAP_FAILED) {
    perror("mmap failed");
    exit(EXIT_FAILURE);
}

shmaddr->flag = 0;

write(1, "Enter a message: ", 17);
ssize_t bytes_read = read(0, shmaddr->message, BUFFER_SIZE);
if (bytes_read == -1) {
    perror("read failed");
    exit(EXIT_FAILURE);
}


if (shmaddr->message[bytes_read - 1] == '\n') {
    shmaddr->message[bytes_read - 1] = '\0';
}

pid_t child1 = fork();
if (child1 == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
```

```
}

if (child1 == 0) {
    execl("./child1", "child1", SEM_EMPTY, SEM_FULL, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

pid_t child2 = fork();
if (child2 == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
}

if (child2 == 0) {
    execl("./child2", "child2", SEM_EMPTY, SEM_FULL, NULL);
    perror("execl failed");
    exit(EXIT_FAILURE);
}

sem_post(sem_empty);

wait(NULL);
wait(NULL);

write(1, "Final message: ", 15);
write(1, shmaddr->message, strlen(shmaddr->message));
write(1, "\n", 1);

sem_close(sem_empty);
sem_close(sem_full);
sem_unlink(SEM_EMPTY);
sem_unlink(SEM_FULL);

munmap(shmaddr, sizeof(struct SharedData));
shm_unlink(SHM_NAME);

return 0;


    }
```

Child1.c

```
    #include <stdio.h>

    #include <stdlib.h>

    #include <sys/mman.h>

    #include <fcntl.h>

    #include <semaphore.h>

    #include <unistd.h>

    #include <ctype.h>

    #include <string.h>

    #define SHM_NAME "/shared_memory"

    #define BUFFER_SIZE 256

    struct SharedData {

     char message[BUFFER_SIZE];

     int flag;

     };

    int main(int argc, char *argv[]) {

     if (argc != 3) {

     write(2, "Usage: child1 \n", 37); exit(EXIT_FAILURE);

     }

sem_t*sem_empty=sem_open(argv[1],0);
sem_t*sem_full=sem_open(argv[2],0);

if (sem_empty == SEM_FAILED || sem_full == SEM_FAILED) {


    perror("sem_open                                                    failed");
    exit(EXIT_FAILURE);
}

int       shm_fd       =       shm_open(SHM_NAME,       O_RDWR,       0666);
if               (shm_fd               ==               -1)               {
```

```
    perror("shm_open                                                    failed");
    exit(EXIT_FAILURE);
}

struct   SharedData   *shmaddr   =   mmap(NULL,   sizeof(struct   SharedData),
                                PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
if             (shmaddr             ==             MAP_FAILED)             {
    perror("mmap                                                        failed");
    exit(EXIT_FAILURE);
}

sem_wait(sem_empty);

for    (int    i    =    0;    shmaddr->message[i]    !=    '\0';    i++)    {
    shmaddr->message[i]   =   toupper((unsigned   char)shmaddr->message[i]);
}

shmaddr->flag                                =                                1;

sem_post(sem_full);

munmap(shmaddr,                  sizeof(struct                  SharedData));
close(shm_fd);
return                                                                      0;


    }
```

Child2.c

```
    #include <stdio.h>

    #include <stdlib.h>

    #include <sys/mman.h>

    #include <fcntl.h>

    #include <semaphore.h>

    #include <unistd.h>

    #include <string.h>
```

```
    #define SHM_NAME "/shared_memory"

    #define BUFFER_SIZE 256

    struct SharedData {

     char message[BUFFER_SIZE];

     int flag;

     };

    int main(int argc, char *argv[]) {

     if (argc != 3) {

     write(2, "Usage: child2 \n", 37);

     exit(EXIT_FAILURE);

    }

    sem_t*sem_empty=sem_open(argv[1],0);
sem_t*sem_full=sem_open(argv[2],0);

if(sem_empty == SEM_FAILED||sem_full == SEM_FAILED) {


    perror("sem_open                                                    failed");
    exit(EXIT_FAILURE);
}

int       shm_fd       =       shm_open(SHM_NAME,       O_RDWR,       0666);
if               (shm_fd               ==               -1)               {
    perror("shm_open                                                    failed");
    exit(EXIT_FAILURE);
}

struct   SharedData   *shmaddr   =   mmap(NULL,   sizeof(struct   SharedData),
                                PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
if             (shmaddr             ==             MAP_FAILED)             {
    perror("mmap                                                        failed");
    exit(EXIT_FAILURE);
}

sem_wait(sem_full);

for    (int    i    =    0;    shmaddr->message[i]    !=    '\0';    i++)    {
    if         (shmaddr->message[i]         ==         '         ')         {
```

```
            shmaddr->message[i]                    =                    '_';
        }
    }

    write(1,      "Child      2      processed      message:      ",      26);
    write(1,            shmaddr->message,            strlen(shmaddr->message));
    write(1,                              "\n",                              1);

    munmap(shmaddr,                  sizeof(struct                  SharedData));
    close(shm_fd);
    return                                                                  0;


        }
```

Протокол работы программы

Тестирование:

```
Enter your string or (Enter / CTRL + D) for stop: ijon kj lk
Processed result: IJON_KJ_LK


Enter your string or (Enter / CTRL + D) for stop: knm kl kjl
Processed result: KNM_KL_KJL


Enter your string or (Enter / CTRL + D) for stop: Hello you
Processed result: HELLO_YOU
```

Strace:

lizka@LizaAlisa:~/ЛАБЫ_ОС/Лаба1$ strace -f ./parent

execve("./parent", ["./parent"], 0x7fffe34dbaf8 /* 29 vars */) = 0 brk(NULL) = 0x55e49d2bc000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7e3bf2d000

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=19163, ...}) = 0

mmap(NULL, 19163, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f7e3bf28000 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"..., 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) =
0x7f7e3bd16000

mmap(0x7f7e3bd3e000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f7e3bd3e000

mmap(0x7f7e3bec6000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f7e3bec6000

mmap(0x7f7e3bf15000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f7e3bf15000

mmap(0x7f7e3bf1b000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f7e3bf1b000 close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f7e3bd13000

arch_prctl(ARCH_SET_FS, 0x7f7e3bd13740) = 0

set_tid_address(0x7f7e3bd13a10) = 575

set_robust_list(0x7f7e3bd13a20, 24) = 0

rseq(0x7f7e3bd14060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f7e3bf15000, 16384, PROT_READ) = 0

mprotect(0x55e479cfe000, 4096, PROT_READ) = 0

mprotect(0x7f7e3bf65000, 8192, PROT_READ) = 0

*prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,

rlim_max=RLIM64_INFINITY}) = 0 munmap(0x7f7e3bf28000, 19163) = 0 pipe2([3, 4], 0) = 0
pipe2([5, 6], 0) = 0

clone(child_stack=NULL,

flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace:

Process 576 attached , child_tidptr=0x7f7e3bd13a10) = 576

[pid 576] set_robust_list(0x7f7e3bd13a20, 24 <unfinished ...>

[pid 575] close(3 <unfinished ...> [pid 576] <... set_robust_list resumed>) = 0

[pid 575] <... close resumed>) = 0

[pid 576] close(4 <unfinished ...>

[pid 575] close(6 <unfinished ...> [pid 576] <... close resumed>) = 0

[pid 575] <... close resumed>) = 0

[pid 576] close(5 <unfinished ...>

[pid 575] read(0, <unfinished ...>

[pid 576] <... close resumed>) = 0

[pid 576] dup2(3, 0) = 0

[pid 576] dup2(6, 1) = 1

[pid 576] close(3) = 0

[pid 576] close(6) = 0

[pid 576] execve("./out/child1", ["child1"], 0x7ffe4841ed68 /* 29 vars */) = -1 ENOENT (No such file or directory)

[pid 576] exit_group(1) = ?

[pid 576] +++ exited with 1 +++ <... read resumed>0x7ffe4841eb20, 256) = ? ERESTARTSYS (To be restarted if SA_RESTART is set) --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=576, si_uid=1000, si_status=1, si_utime=0, si_stime=0} --- read(0,

Вывод

Во время выполнения лабораторной работы я разработала программу, которая использует механизмы межпроцессного взаимодействия, такие как общая память и семафоры, для обработки строк, вводимых пользователем, с участием нескольких процессов. Основная сложность возникла при управлении ресурсами, такими как семафоры и общая память, а также в синхронизации процессов. Проблемы возникали из-за некорректной работы с семафорами и общей памятью в дочерних процессах, что приводило к возможным гонкам состояний и неправильному завершению процессов. Я решила эту проблему, добавив дополнительные проверки ошибок и убедившись, что дочерние процессы корректно завершаются.

В будущем хотелось бы больше времени уделить отладке и тестированию многозадачности и синхронизации между процессами, чтобы избежать подобных проблем с ресурсами. Также важно будет улучшить код дочерних процессов, чтобы они корректно работали с общей памятью и семафорами. В целом, эта работа была полезной и помогла мне лучше понять принципы работы с межпроцессным взаимодействием в операционной системе.