Operating Systems. Homework 7

Operating Systems
Александр Васюков | БПИ235

Задание

`Разработать программы клиента и сервера, взаимодействующих через разделяемую память с использованием функций UNIX SYSTEM V. Клиент генерирует случайные числа в том же диапазоне, что и ранее рассмотренный пример. Сервер осуществляет их вывод. Предполагается (но специально не контролируется), что запускаются только один клиент и один сервер. Необходимо обеспечить корректное завершение работы для одного клиента и одного сервера, при котором удаляется сегмент разделяемой памяти. Предложить и реализовать свой вариант корректного завершения. Описать этот вариант в отчете.`

`Опционально до +2 баллов`
`Реализовать и описать в отчете дополнительно один или два варианта общего завершения клиента и сервера (+1 балл за каждый вариант). В отчете отразить решения, используемые для корректного завершения обеих программ.`

Решение

Программы на Github: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/Operating%20Systems/Homeworks/07

Сервер

- Создаёт разделяемую память `shm_open`.
- Устанавливает `ready = 0` и `terminate = 0`.
- В бесконечном цикле:
  Если `terminate == 1` — сервер завершает работу и удаляет память (`shm_unlink`).
  Если `ready == 1` — читает число, выводит его, сбрасывает `ready`.
- Корректное завершение:
  При получении `Ctrl+C` сервер:
  1. Устанавливает `terminate = 1`, сигнализируя клиенту о завершении.
  2. Отсоединяет и удаляет разделяемую память (`munmap`, `shm_unlink`).
- Разделяемая память создаётся сервером и удаляется при завершении работы.
- Семафор используется для синхронизации записи и чтения данных.

Клиент

- Открывает уже существующий сегмент памяти.

- В бесконечном цикле:
  Если `terminate == 1`, завершает работу.
  Если `ready == 0`, генерирует случайное число, записывает в `number`, устанавливает `ready = 1`.
- Корректное завершение: Клиент отслеживает флаг `terminate`. Если сервер завершился и установил `terminate = 1`, клиент корректно завершает работу и освобождается от памяти.

Server

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

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)


typedef struct {
    int number;
    int ready;
    int terminate;
} shared_data_t;

shared_data_t *shm_ptr;
int shm_fd;

void cleanup(int signum) {
    printf("\nСервер завершает работу...\n");
    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);
    shm_unlink(SHM_NAME);
    exit(0);
}

int main() {
    signal(SIGINT, cleanup);

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

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("ftruncate");
```

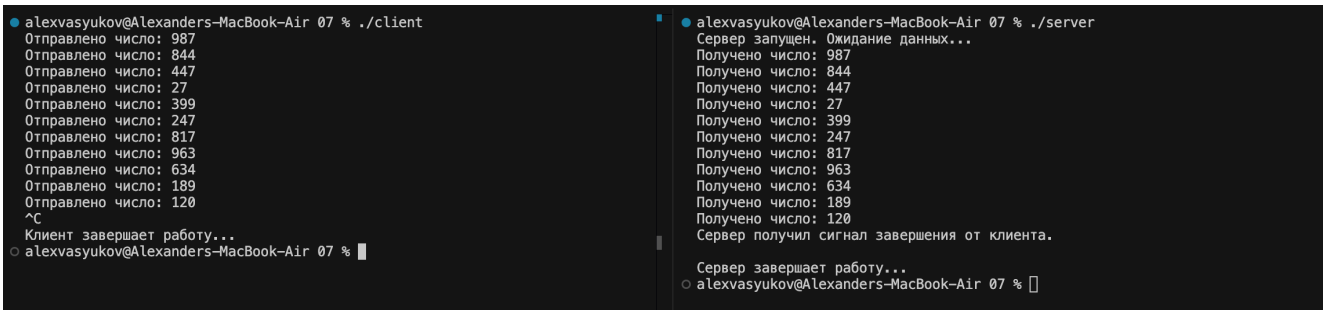
```
        exit(1);
    }

    shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("mmap");
        exit(1);
    }

    shm_ptr->ready = 0;
    shm_ptr->terminate = 0;

    printf("Сервер запущен. Ожидание данных...\n");

    while (1) {
        if (shm_ptr->terminate) {
            printf("Сервер получил сигнал завершения от клиента.\n");
            break;
        }

        if (shm_ptr->ready) {
            printf("Получено число: %d\n", shm_ptr->number);
            shm_ptr->ready = 0;
        }

        usleep(100000);
    }

    cleanup(SIGINT);
    return 0;
}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include <string.h>

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)

typedef struct {
    int number;
    int ready;
    int terminate;
```

```
} shared_data_t;

shared_data_t *shm_ptr;
int shm_fd;

void cleanup(int signum) {
    printf("\nКлиент завершает работу...\n");
    shm_ptr->terminate = 1;
    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);
    exit(0);
}

int main() {
    signal(SIGINT, cleanup);

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

    shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("mmap");
        exit(1);
    }

    srand(time(NULL));

    while (1) {
        if (!shm_ptr->ready) {
            int num = rand() % 1000;
            shm_ptr->number = num;
            shm_ptr->ready = 1;
            printf("Отправлено число: %d\n", num);
        }
        sleep(1);
    }

    return 0;
}
```

Результат:



Дополнительные программы:

1. Завершение через клиента

1. Клиент отправляет числа в разделяемую память и ждёт подтверждения.
2. Сервер читает числа и устанавливает `ready`, позволяя клиенту отправлять новые.
3. Клиент предлагает пользователю решить, отправлять ли следующее число (`y/n`).
4. Если пользователь выбирает `n`, клиент устанавливает `terminate = 1`, сервер завершает работу и очищает ресурсы.

Server

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

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)

typedef struct {
    int number;
    int ready;
    int terminate;
} shared_data_t;

int main() {
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка создания разделяемой памяти");
        exit(1);
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("Ошибка при ftruncate");
        exit(1);
    }

    shared_data_t *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
```

```
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap");
        exit(1);
    }

    shm_ptr->ready = 0;
    shm_ptr->terminate = 0;

    printf("Сервер запущен. Ожидание данных...\n");

    while (1) {
        if (shm_ptr->terminate) {
            printf("Сервер завершает работу по сигналу клиента...\n");
            munmap(shm_ptr, SHM_SIZE);
            close(shm_fd);
            shm_unlink(SHM_NAME);
            exit(0);
        }

        if (shm_ptr->ready) {
            printf("Получено число: %d\n", shm_ptr->number);
            shm_ptr->ready = 0;
        }
        usleep(100000);
    }

    return 0;
}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>
#include <string.h>

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)

typedef struct {
    int number;
    int ready;
    int terminate;
} shared_data_t;

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
```

```c
    if (shm_fd == -1) {
        perror("Ошибка подключения к разделяемой памяти");
        exit(1);
    }

    shared_data_t *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap");
        exit(1);
    }

    srand(time(NULL));

    while (1) {
        int num = rand() % 1000;
        while (shm_ptr->ready) {
            usleep(100000);
        }

        shm_ptr->number = num;
        shm_ptr->ready = 1;
        printf("Отправлено число: %d\n", num);

        printf("Продолжить отправку? (y/n): ");
        char choice;
        scanf(" %c", &choice);

        if (choice == 'n' || choice == 'N') {
            shm_ptr->terminate = 1;
            printf("Клиент завершает сервер...\n");
            break;
        }
    }

    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);
    return 0;
}
```

Результат:

2. Завершение через сервер

1. Сервер создаёт разделяемую память и ждёт числа от клиента.

2. Клиент отправляет случайные числа в память, сервер их читает и выводит.
3. Если на сервере нажать `Ctrl+C`:
   - Устанавливается `terminate = 1`, чтобы предупредить клиента.
   - Удаляет память и завершает работу.
4. Клиент проверяет `terminate` в цикле. Если сервер закрылся, клиент тоже завершает работу.

Server

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

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)

typedef struct {
    int number;
    int ready;
    int terminate;
} shared_data_t;

shared_data_t *shm_ptr = NULL;
int shm_fd;

void cleanup(int signum) {
    printf("\nСервер завершает работу по Ctrl+C...\n");
    if (shm_ptr != NULL) {
        shm_ptr->terminate = 1;
        sleep(1);
        munmap(shm_ptr, SHM_SIZE);
    }
    if (shm_fd >= 0) close(shm_fd);
    shm_unlink(SHM_NAME);
    exit(0);
}

int main() {
    signal(SIGINT, cleanup);

    shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Ошибка создания разделяемой памяти");
        exit(1);
    }
    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("Ошибка при ftruncate");
```

```
        exit(1);
    }

    shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap");
        exit(1);
    }

    shm_ptr->ready = 0;
    shm_ptr->terminate = 0;

    printf("Сервер запущен. Ожидание данных...\n");

    while (1) {
        if (shm_ptr->terminate) {
            printf("Сервер завершает работу по сигналу от клиента...\n");
            cleanup(0);
        }

        if (shm_ptr->ready) {
            printf("Получено число: %d\n", shm_ptr->number);
            shm_ptr->ready = 0;
        }

        usleep(100000);
    }
}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>

#define SHM_NAME "/my_shared_memory"
#define SHM_SIZE sizeof(shared_data_t)

typedef struct {
    int number;
    int ready;
    int terminate;
} shared_data_t;

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

```
        perror("Ошибка подключения к разделяемой памяти");
        exit(1);
    }

    shared_data_t *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Ошибка при mmap");
        exit(1);
    }

    srand(time(NULL));

    while (1) {
        if (shm_ptr->terminate) {
            printf("Сервер завершил работу. Клиент тоже завершает
работу...\n");
            break;
        }

        if (!shm_ptr->ready) {
            int num = rand() % 1000;
            shm_ptr->number = num;
            shm_ptr->ready = 1;
            printf("Отправлено число: %d\n", num);
        }

        sleep(1);
    }

    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);
    return 0;
}
```

Результат: