Operating Systems. Homework 6

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. Не забыть приложить к отчету исходные тексты программ с различными вариантами решений.`

Решение

Программы на Github: https://github.com/vasyukov1/HSE-FCS-SE-2-year/tree/main/Operating%20Systems/Homeworks/06

1. Разделяемая память создаётся сервером и удаляется при завершении работы.
2. Семафор используется для синхронизации записи и чтения данных.
3. Обработчик сигналов (`SIGINT`) используетя для корректного завершения сервера и клиента через `Ctrl+C` .

Server

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>
#include <unistd.h>

#define SHM_KEY 1234
#define SEM_KEY 5678
#define SHM_SIZE sizeof(shared_data_t)
```

```c
typedef struct {
    int number;
    int ready;
} shared_data_t;


int shm_id, sem_id;
shared_data_t *shm_ptr;

void cleanup(int signum) {
    printf("\nСервер завершает работу...\n");
    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, NULL);
    semctl(sem_id, 0, IPC_RMID);
    exit(0);
}

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

    shm_id = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT | 0666);
    shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);
    shm_ptr->ready = 0;

    sem_id = semget(SEM_KEY, 1, IPC_CREAT | 0666);
    semctl(sem_id, 0, SETVAL, 1);

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

    while (1) {
        if (shm_ptr->ready) {
            printf("Получено число: %d\n", shm_ptr->number);
            shm_ptr->ready = 0;
        }
        usleep(100000);
    }

    return 0;
}
```

Client

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>
#include <unistd.h>
#include <time.h>
```

```c
#define SHM_KEY 1234
#define SEM_KEY 5678
#define SHM_SIZE sizeof(shared_data_t)

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

int shm_id;
shared_data_t *shm_ptr;

void cleanup(int signum) {
    printf("\nКлиент завершает работу...\n");
    shmdt(shm_ptr);
    exit(0);
}

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

    shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);

    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;
}
```

Дополнительные программы:

1. Завершение через клиента

1. Клиент отправляет числа в разделяемую память и ждёт подтверждения (`ready == 0`).
2. Сервер читает числа и устанавливает `ready = 0` , 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 <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

#define SHM_KEY 1235
#define SHM_SIZE sizeof(shared_data_t)

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

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("Ошибка создания разделяемой памяти");
        exit(1);
    }

    shared_data_t *shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (void *)-1) {
        perror("Ошибка присоединения к разделяемой памяти");
        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...\n");
            shmdt(shm_ptr);
            shmctl(shm_id, IPC_RMID, NULL);
            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 <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <time.h>

#define SHM_KEY 1235
#define SHM_SIZE sizeof(shared_data_t)

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

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id == -1) {
        perror("Ошибка подключения к разделяемой памяти");
        exit(1);
    }

    shared_data_t *shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (void *)-1) {
        perror("Ошибка присоединения к разделяемой памяти");
        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;
        }
    }

    shmdt(shm_ptr);
    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.
   - Ждёт 2 секунды, затем удаляет память и завершает работу.
4. Клиент проверяет `terminate` в цикле. Если сервер закрылся, клиент тоже завершает работу.

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

#define SHM_KEY 1235
#define SHM_SIZE sizeof(shared_data_t)

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

int shm_id;
shared_data_t *shm_ptr;
```

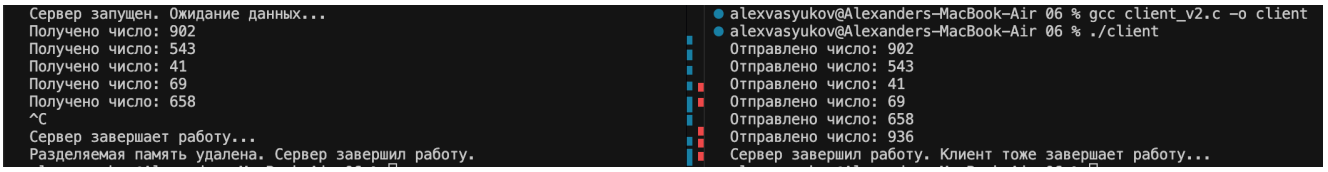
```c
void cleanup(int signum) {
    printf("\nСервер завершает работу...\n");
    shm_ptr->terminate = 1;

    sleep(2);

    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, NULL);
    printf("Разделяемая память удалена. Сервер завершил работу.\n");
    exit(0);
}

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

    shm_id = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("Ошибка создания разделяемой памяти");
        exit(1);
    }

    shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (void *)-1) {
        perror("Ошибка присоединения к разделяемой памяти");
        exit(1);
    }

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

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

    while (1) {
        if (shm_ptr->ready) {
            printf("Получено число: %d\n", shm_ptr->number);
            shm_ptr->ready = 0;
        }
        sleep(1);
    }

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

Client

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
```

```c
#include <time.h>

#define SHM_KEY 1235
#define SHM_SIZE sizeof(shared_data_t)

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

int main() {
    int shm_id = shmget(SHM_KEY, SHM_SIZE, 0666);
    if (shm_id == -1) {
        perror("Ошибка подключения к разделяемой памяти");
        exit(1);
    }

    shared_data_t *shm_ptr = (shared_data_t *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (void *)-1) {
        perror("Ошибка присоединения к разделяемой памяти");
        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);
    }

    shmdt(shm_ptr);
    return 0;
}
```

Результат:


Сервер запущен. Ожидание данных...
Получено число: 902
Получено число: 543
Получено число: 41
Получено число: 69
Получено число: 658
^C
Сервер завершает работу...
Разделяемая память удалена. Сервер завершил работу.
alexvasyukov@Alexanders-MacBook-Air 06 % gcc client_v2.c -o client
alexvasyukov@Alexanders-MacBook-Air 06 % ./client
Отправлено число: 902
Отправлено число: 543
Отправлено число: 41
Отправлено число: 69
Отправлено число: 658
Отправлено число: 936
Сервер завершил работу. Клиент тоже завершает работу...