

Домашнее задание 6. Мусаев Умахан Рашидович.

Условие:

До 8 баллов

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.**

Работа выполнена с опциональной частью.

Отчет:

Основной вариант:

- **server.cpp**
- **clienr.cpp**

В этом варианте клиент и сервер используют разделяемую память для передачи данных и флага завершения. Клиент устанавливает флаг `exit_flag` в разделяемой памяти при завершении, что позволяет серверу корректно завершить работу и удалить ресурсы.

server.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>

#define SHM_KEY 0x1234
#define SEM_KEY 0x5678

struct SharedData {
    int number;
    int exit_flag;
};

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};

int shm_id;
int sem_id;
SharedData *shm_ptr;

void cleanup(int sig) {
    std::cout << "\nCleaning up resources..." << std::endl;
    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, nullptr);
```

```

    semctl(sem_id, 0, IPC_RMID);
    exit(EXIT_SUCCESS);
}

void init_semaphores(int sem_id) {
    union semun arg;
    unsigned short values[2] = {1, 0};
    arg.array = values;
    if (semctl(sem_id, 0, SETALL, arg) == -1) {
        perror("semctl SETALL");
        exit(EXIT_FAILURE);
    }
}

int main() {
    shm_id = shmget(SHM_KEY, sizeof(SharedData), IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("shmget");
        return EXIT_FAILURE;
    }

    shm_ptr = (SharedData*)shmat(shm_id, nullptr, 0);
    if (shm_ptr == (void*)-1) {
        perror("shmat");
        semctl(shm_id, IPC_RMID, nullptr);
        return EXIT_FAILURE;
    }

    shm_ptr->exit_flag = 0;

    sem_id = semget(SEM_KEY, 2, IPC_CREAT | 0666);
    if (sem_id == -1) {
        perror("semget");
        shmdt(shm_ptr);
        shmctl(shm_id, IPC_RMID, nullptr);
        return EXIT_FAILURE;
    }

    init_semaphores(sem_id);
    signal(SIGINT, cleanup);

    std::cout << "Server started. Waiting for client..." << std::endl;

    while (true) {
        struct sembuf sops = {1, -1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop server wait");
            break;
        }

        if (shm_ptr->exit_flag) {
            std::cout << "Client requested exit. Exiting..." << std::endl;
            break;
        }

        std::cout << "Received number: " << shm_ptr->number << std::endl;

        sops = {0, 1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop client post");
            break;
        }
    }

    cleanup(0);
    return EXIT_SUCCESS;
}

```

client.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>
#include <ctime>

#define SHM_KEY 0x1234
#define SEM_KEY 0x5678

struct SharedData {
    int number;
    int exit_flag;
};

int shm_id;
int sem_id;
SharedData *shm_ptr;

void cleanup(int sig) {
    std::cout << "\nExiting client..." << std::endl;
    shmdt(shm_ptr);
    exit(EXIT_SUCCESS);
}

int main() {
    shm_id = shmget(SHM_KEY, sizeof(SharedData), 0666);
    if (shm_id == -1) {
        perror("shmget");
        return EXIT_FAILURE;
    }

    shm_ptr = (SharedData*)shmat(shm_id, nullptr, 0);
    if (shm_ptr == (void*)-1) {
        perror("shmat");
        return EXIT_FAILURE;
    }

    sem_id = semget(SEM_KEY, 2, 0666);
    if (sem_id == -1) {
        perror("semget");
        shmdt(shm_ptr);
        return EXIT_FAILURE;
    }

    signal(SIGINT, [](int sig) {
        shm_ptr->exit_flag = 1;
        struct sembuf sops = {1, 1, 0};
        semop(sem_id, &sops, 1);
        cleanup(sig);
    });

    srand(time(nullptr));

    while (true) {
        int num = rand() % 100;

        struct sembuf sops = {0, -1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop client wait");
            break;
        }

        shm_ptr->number = num;
    }
}
```

```

        sops = {1, 1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop server post");
            break;
        }

        sleep(1);
    }

    shmdt(shm_ptr);
    return EXIT_SUCCESS;
}

```

Дополнительный вариант:

- client_sigusr1.cpp
- server_sigusr1.cpp

В этом варианте клиент отправляет сигнал SIGUSR1 серверу при завершении. Сервер, получив сигнал, выполняет очистку ресурсов.

server_sigusr1.cpp

```

#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>

#define SHM_KEY 0x1234
#define SEM_KEY 0x5678

struct SharedData {
    int number;
    int exit_flag;
    pid_t server_pid; // Добавлено поле для PID сервера
};

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};

int shm_id;
int sem_id;
SharedData *shm_ptr;

void cleanup(int sig) {
    std::cout << "\nCleaning up resources..." << std::endl;
    shmdt(shm_ptr);
    shmctl(shm_id, IPC_RMID, nullptr);
    semctl(sem_id, 0, IPC_RMID);
    exit(EXIT_SUCCESS);
}

void handle_exit_signal(int sig) {
    std::cout << "\nReceived exit signal. Exiting..." << std::endl;
}

```

```

        cleanup(sig);
    }

void init_semaphores(int sem_id) {
    union semun arg;
    unsigned short values[2] = {1, 0};
    arg.array = values;
    if (semctl(sem_id, 0, SETALL, arg) == -1) {
        perror("semctl SETALL");
        exit(EXIT_FAILURE);
    }
}

int main() {
    shm_id = shmget(SHM_KEY, sizeof(SharedData), IPC_CREAT | 0666);
    if (shm_id == -1) {
        perror("shmget");
        return EXIT_FAILURE;
    }

    shm_ptr = (SharedData*)shmat(shm_id, nullptr, 0);
    if (shm_ptr == (void*)-1) {
        perror("shmat");
        shmctl(shm_id, IPC_RMID, nullptr);
        return EXIT_FAILURE;
    }

    shm_ptr->exit_flag = 0;
    shm_ptr->server_pid = getpid(); // Записываем PID сервера

    sem_id = semget(SEM_KEY, 2, IPC_CREAT | 0666);
    if (sem_id == -1) {
        perror("semget");
        shmdt(shm_ptr);
        shmctl(shm_id, IPC_RMID, nullptr);
        return EXIT_FAILURE;
    }

    init_semaphores(sem_id);
    signal(SIGINT, cleanup); // Обработчик Ctrl+C
    signal(SIGUSR1, handle_exit_signal); // Обработчик SIGUSR1

    std::cout << "Server started. PID: " << getpid() << ". Waiting for client..." <<
std::endl;

    while (true) {
        struct sembuf sops = {1, -1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop server wait");
            break;
        }

        if (shm_ptr->exit_flag) {
            std::cout << "Client requested exit. Exiting..." << std::endl;
            break;
        }

        std::cout << "Received number: " << shm_ptr->number << std::endl;

        sops = {0, 1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop client post");
            break;
        }
    }

    cleanup(0);
    return EXIT_SUCCESS;
}

```

client_sigur1.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <signal.h>
#include <ctime>

#define SHM_KEY 0x1234
#define SEM_KEY 0x5678

struct SharedData {
    int number;
    int exit_flag;
    pid_t server_pid; // Добавлено поле для PID сервера
};

int shm_id;
int sem_id;
SharedData *shm_ptr;

void cleanup(int sig) {
    std::cout << "\nExiting client..." << std::endl;
    shmdt(shm_ptr);
    exit(EXIT_SUCCESS);
}

int main() {
    shm_id = shmget(SHM_KEY, sizeof(SharedData), 0666);
    if (shm_id == -1) {
        perror("shmget");
        return EXIT_FAILURE;
    }

    shm_ptr = (SharedData*)shmat(shm_id, nullptr, 0);
    if (shm_ptr == (void*)-1) {
        perror("shmat");
        return EXIT_FAILURE;
    }

    sem_id = semget(SEM_KEY, 2, 0666);
    if (sem_id == -1) {
        perror("semget");
        shmdt(shm_ptr);
        return EXIT_FAILURE;
    }

    signal(SIGINT, [](int sig) {
        pid_t server_pid = shm_ptr->server_pid;
        std::cout << "\nSending SIGUSR1 to server (PID: " << server_pid << ")..." <<
std::endl;
        kill(server_pid, SIGUSR1); // Отправляем SIGUSR1 серверу
        cleanup(sig);
    });

    srand(time(nullptr));

    while (true) {
        int num = rand() % 100;

        struct sembuf sops = {0, -1, 0};
        if (semop(sem_id, &sops, 1) == -1) {
            perror("semop client wait");
            break;
        }
    }
}
```

```

shm_ptr->number = num;

sops = {1, 1, 0};
if (semop(sem_id, &sops, 1) == -1) {
    perror("semop server post");
    break;
}

sleep(1);
}

shmdt(shm_ptr);
return EXIT_SUCCESS;
}

```

Запуск программы:

1. Сборка программы:

- `g++ server.cpp -o server`
- `g++ clienr.cpp -o clienr`
- `g++ server_sigusr1.cpp -o server_sigusr1`
- `g++ client_sigur1.cpp -o client_sigur1`

2. Запуск программы(нужно запускать на разных консолях)

- `./server` или `./server_sigusr1`
- `./clienr` или `./client_sigur1`

3. Для остановки программы, в консоли клиента нужно прописать `Ctrl+C`.

Примеры работы программы:

Основной вариант:

```

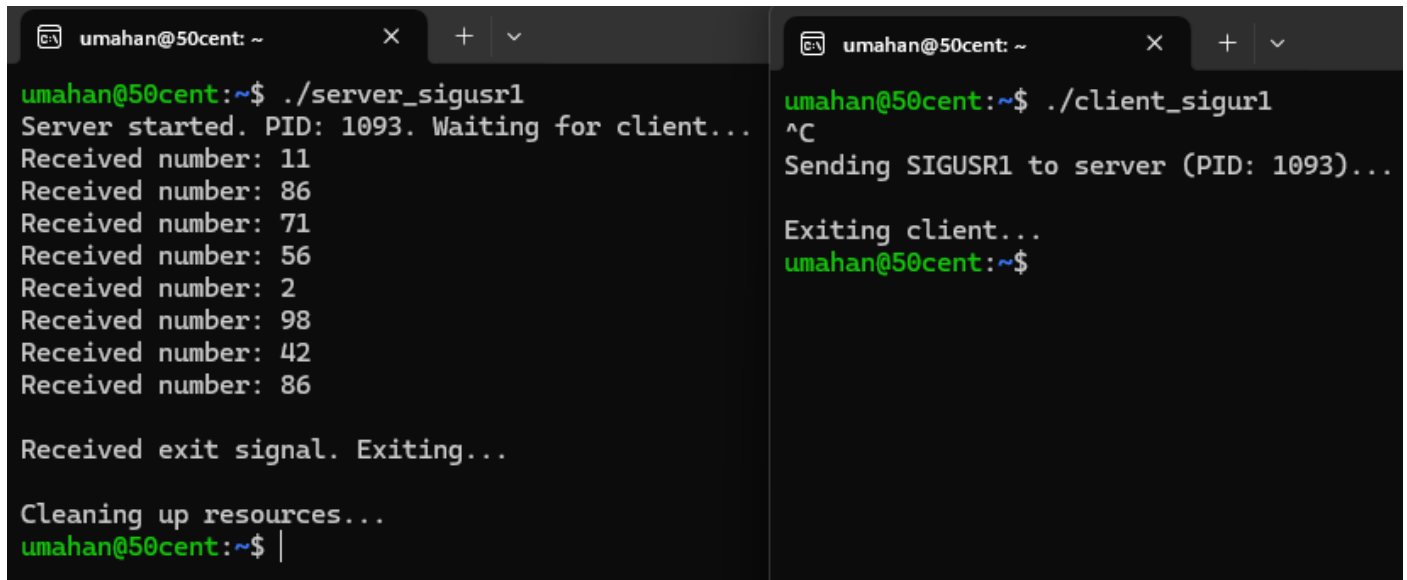
umahan@50cent: ~
umahan@50cent:~$ ./server
Server started. Waiting for client...
Received number: 60
Received number: 59
Received number: 91
Received number: 56
Received number: 47
Received number: 15
Received number: 13
Received number: 9
Received number: 38
Received number: 51
Received number: 58
Received number: 51
Client requested exit. Exiting...

Cleaning up resources...
umahan@50cent:~$ |

umahan@50cent: ~
umahan@50cent:~$ ./clienr
^C
Exiting client...
umahan@50cent:~$

```

Дополнительный вариант:



The image shows two terminal windows side-by-side. The left window shows the execution of a server program, and the right window shows the execution of a client program. Both are running on a system named 'umahan@50cent'.

```
umahan@50cent: ~  
umahan@50cent:~$ ./server_sigusr1  
Server started. PID: 1093. Waiting for client...  
Received number: 11  
Received number: 86  
Received number: 71  
Received number: 56  
Received number: 2  
Received number: 98  
Received number: 42  
Received number: 86  
  
Received exit signal. Exiting...  
  
Cleaning up resources...  
umahan@50cent:~$ |
```

```
umahan@50cent: ~  
umahan@50cent:~$ ./client_sigusr1  
^C  
Sending SIGUSR1 to server (PID: 1093)...  
  
Exiting client...  
umahan@50cent:~$
```

В архиве также содержится видеопример работы программы.