Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

КУРСОВАЯ РАБОТА

По дисциплине: «Операционные системы»

Выполнил: студент группы ИП-211

Оганесян Альберт

Проверил:

Профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2024

**Задание:** В качестве задания был выбрал третий уровень на оценку «отлично» реализующий сетевую службу с функционалом диспетчера задач и просмотра ELF-файлов.

**Выполнение работы:**

Программа выполняет роль сервера, к которой можно подключиться и получить информацию о текущих задачах на сервере, совершить информацию над ними, и посмотреть ELF-файлы сервера.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <pthread.h>

#include <sys/resource.h>

#include <errno.h>

#include <elf.h>

#include <signal.h>

#define PORT 8080

#define BUF\_SIZE 1024

void list(int client\_fd)

{

FILE \*fp\_tasks = popen("ps -eo pid,pri,%cpu,%mem,cmd", "r");

if (fp\_tasks == NULL)

{

perror("popen");

write(client\_fd, "Failed to list tasks and resources\n", 35);

return;

}

char buffer[BUF\_SIZE];

write(client\_fd, "PID PRI %CPU %MEM CMD\n", 31);

while (fgets(buffer, sizeof(buffer), fp\_tasks) != NULL)

{

write(client\_fd, buffer, strlen(buffer));

}

pclose(fp\_tasks);

}

void kill\_task(int client\_fd, int pid)

{

if (kill(pid, SIGKILL) == 0)

{

char response[BUF\_SIZE];

snprintf(response, sizeof(response), "Task %d terminated successfully\n", pid);

write(client\_fd, response, strlen(response));

}

else

{

perror("kill");

write(client\_fd, "Failed to terminate task\n", 25);

}

}

void read\_elf(int client\_fd, const char \*file\_path)

{

int fd = open(file\_path, O\_RDONLY);

if (fd < 0)

{

perror("open");

write(client\_fd, "Failed to open file\n", 20);

return;

}

Elf64\_Ehdr ehdr;

if (read(fd, &ehdr, sizeof(ehdr)) != sizeof(ehdr))

{

perror("read");

write(client\_fd, "Failed to read ELF header\n", 26);

close(fd);

return;

}

if (memcmp(ehdr.e\_ident, ELFMAG, SELFMAG) != 0)

{

write(client\_fd, "Not a valid ELF file\n", 22);

close(fd);

return;

}

char response[BUF\_SIZE];

snprintf(response, sizeof(response),

"ELF File Info:\nType: %d\nMachine: %d\nVersion: %d\nEntry point: 0x%lx\n",

ehdr.e\_type, ehdr.e\_machine, ehdr.e\_version, ehdr.e\_entry);

write(client\_fd, response, strlen(response));

close(fd);

}

// Функция для обработки запросов клиента

void \*handle\_client(void \*arg)

{

int client\_fd = \*(int \*)arg;

free(arg);

char buffer[BUF\_SIZE];

while (1)

{

memset(buffer, 0, BUF\_SIZE);

int bytes\_read = read(client\_fd, buffer, BUF\_SIZE);

if (bytes\_read <= 0)

{

break;

}

char command[BUF\_SIZE], arg1[BUF\_SIZE], arg2[BUF\_SIZE];

int pid, priority;

if (sscanf(buffer, "%s %s %s", command, arg1, arg2) >= 1)

{

if (strcmp(command, "list") == 0)

{

list(client\_fd);

}

else if (strcmp(command, "kill") == 0)

{

pid = atoi(arg1);

kill\_task(client\_fd, pid);

}

else if (strcmp(command, "elf") == 0)

{

read\_elf(client\_fd, arg1);

}

else if (strcmp(command, "exit") == 0)

{

break;

}

else

{

write(client\_fd, "Unknown command\n", 17);

}

}

else

{

write(client\_fd, "Invalid input\n", 14);

}

}

close(client\_fd);

return NULL;

}

int main()

{

int server\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_len = sizeof(client\_addr);

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd == -1)

{

perror("socket");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{

perror("bind");

close(server\_fd);

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 5) == -1)

{

perror("listen");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d\n", PORT);

while (1)

{

int \*client\_fd = malloc(sizeof(int));

if (client\_fd == NULL)

{

perror("malloc");

continue;

}

\*client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &client\_len);

if (\*client\_fd == -1)

{

perror("accept");

free(client\_fd);

continue;

}

pthread\_t thread\_id;

if (pthread\_create(&thread\_id, NULL, handle\_client, client\_fd) != 0)

{

perror("pthread\_create");

free(client\_fd);

continue;

}

pthread\_detach(thread\_id);

}

close(server\_fd);

return 0;

}

Листинг 1 – файл программы task\_manager.c

### **1. Функция list**

* Выполняет команду ps -eo pid,pri,%cpu,%mem,cmd для получения списка текущих процессов, их идентификаторов (PID), приоритетов, команд и нагрузки на процессор и память
* **Параметры**:
  + client\_fd: Дескриптор сокета клиента для отправки данных.

### **2. Функция kill\_task**

* Завершает процесс с указанным PID с помощью сигнала SIGKILL.
* **Параметры**:
  + client\_fd: Дескриптор сокета клиента для отправки данных.
  + pid: Идентификатор процесса, который необходимо завершить.

### **3. Функция change\_priority**

* Изменяет приоритет процесса с указанным PID, используя функцию setpriority().
* **Параметры**:
  + client\_fd: Дескриптор сокета клиента для отправки данных.
  + pid: Идентификатор процесса, для которого нужно изменить приоритет.
  + priority: Новый приоритет.

### **4. Функция read\_elf**

* Открывает ELF-файл, проверяет его валидность и извлекает базовую информацию из заголовка ELF.
* **Параметры**:
  + client\_fd: Дескриптор сокета клиента для отправки данных.
  + file\_path: Путь к ELF-файлу.

### **5. Функция handle\_client**

* Обрабатывает команды, полученные от клиента, и вызывает соответствующие функции для их выполнения. Поддерживает команды:
  + list — получение списка процессов.
  + kill <pid> — завершение процесса.
  + priority <pid> <priority> — изменение приоритета.
  + elf <file\_path> — информация о ELF-файле.
  + resources — мониторинг использования ресурсов.
  + exit — завершение соединения.
* **Параметры**:
  + arg: Указатель на дескриптор сокета клиента.

### **6. Функция main**

* Основная функция программы. Создает серверный сокет, принимает подключения от клиентов и создает новый поток для обработки каждого клиента.
* **Основной функционал**:
  + Инициализация серверного сокета.
  + Привязка сокета к порту 8080.
  + Слушание входящих соединений.
  + Создание потоков для обработки каждого клиента.

### **7. Переменные и структуры данных:**

* BUF\_SIZE: Размер буфера для приема и отправки данных.
* PORT: Порт для работы сервера (8080).
* pthread\_t thread\_id: Идентификатор потока для обработки клиента.
* struct sockaddr\_in: Структура для хранения адреса сервера и клиента.

Также для теста создадим программу, которая просто будет нагружать процессор:

#include <stdio.h>

int main()

{

while (1)

{

printf("I do something!\n");

}

return 0;

}

Листинг 2 – файл программы test.c

Команда компиляции и результат запуска сервера:

| albert@DESKTOP-70OAJI4:/mnt/c/Users/User/Documents/GitHub/OS/TaskManager&ELFviewer/Network$ gcc -o task\_manager task\_manager.c albert@DESKTOP-70OAJI4:/mnt/c/Users/User/Documents/GitHub/OS/TaskManager&ELFviewer/Network$ ./task\_manager Server is listening on port 8080 |
| --- |

Листинг 3 – Запуск сервера

Теперь подключимся к серверу при помощи утилиты telnet

| albert@DESKTOP-70OAJI4:/mnt/c/Users/User/Documents/GitHub/OS/TaskManager&ELFviewer/Network$ telnet 127.0.0.1 8080 Trying 127.0.0.1... Connected to 127.0.0.1. Escape character is '^]'. |
| --- |

Листинг 4 – подключение клиента

Введем команду list для просмотра процессов, предварительно запустив тестовую программу

| list PID PRI %CPU %MEM CMD  PID PRI %CPU %MEM CMD  1 19 0.0 0.1 /sbin/init  2 19 0.0 0.0 /init  6 19 0.0 0.0 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd 9 --log-truncate  51 20 0.0 0.2 /usr/lib/systemd/systemd-journald  96 19 0.0 0.0 /usr/lib/systemd/systemd-udevd  106 19 0.0 0.1 /usr/lib/systemd/systemd-resolved  107 19 0.0 0.1 /usr/lib/systemd/systemd-timesyncd  155 19 0.0 0.0 /usr/sbin/cron -f -P  156 19 0.0 0.0 @dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only  169 19 0.0 0.1 /usr/lib/systemd/systemd-logind  175 19 0.0 0.2 /usr/libexec/wsl-pro-service -vv  185 19 0.0 0.0 /sbin/agetty -o -p -- \u --noclear --keep-baud - 115200,38400,9600 vt220  192 19 0.0 0.0 /sbin/agetty -o -p -- \u --noclear - linux  204 19 0.0 0.0 /usr/sbin/rsyslogd -n -iNONE  229 19 0.0 0.2 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal  363 19 0.0 0.0 /init  364 19 0.0 0.0 /init  365 19 0.0 0.0 -bash  366 19 0.0 0.0 /bin/login -f  447 19 0.0 0.1 /usr/lib/systemd/systemd --user  454 19 0.0 0.0 (sd-pam)  468 19 0.0 0.0 -bash  862 19 0.0 0.0 /usr/lib/polkit-1/polkitd --no-debug  1266 19 0.0 0.0 /init  1267 19 0.0 0.0 /init  1272 19 0.0 0.0 -bash  1524 19 0.0 0.0 ./task\_manager  1525 19 0.0 0.0 telnet 127.0.0.1 8080  1529 19 0.0 0.0 /init  1530 19 0.5 0.0 /init  1535 19 0.1 0.0 -bash  1555 19 4.5 0.0 ./test  1556 19 0.0 0.0 (udev-worker)  1557 19 0.0 0.0 (udev-worker)  1558 19 0.0 0.0 (udev-worker)  1559 19 0.0 0.0 (udev-worker)  1560 19 0.0 0.0 sh -c -- ps -eo pid,pri,%cpu,%mem,cmd  1561 19 0.0 0.0 ps -eo pid,pri,%cpu,%mem,cmd |
| --- |

Листинг 5 - вывод информации о процессах

Убьем наш тестовый процесс с PID 1555 при помощи команды kill (лист.6) и посмотрим, как отреагировала программа test.c (лист. 7)

| kill 1555 Task 1555 terminated successfully |
| --- |

Листинг 6 - Вывод программы после программы kill

| I do something! I do something! I do something! I do something!Killed |
| --- |

Листинг 7 - Фрагмент вывода программы test

Теперь прочитаем ELF-файл init:

| elf /init ELF File Info: Type: 2 Machine: 62 Version: 1 Entry point: 0x2a0bf0 |
| --- |

Листинг 8 - ответ на команду elf