Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисциплине «Системы Ввода-Вывода»

Выполнили:
Студенты группы P3331
Дворкин Борис Александрович
Краков Кирилл Константинович

Вариант: 2

Преподаватель:
Быковский Сергей Вячеславович

г. Санкт-Петербург

2024 г.

Содержание

Описание задания

Цель

Познакомится с принципами организации ввода/вывода без операционной системы

на примере компьютерной системы на базе процессора с архитектурой RISC-V и

интерфейсом OpenSBI с использованием эмулятора QEMU.

Задачи

1. Реализовать функцию putchar вывода данных в консоль

2. Реализовать функцию getchar для получения данных из консоли

3. На базе реализованных функций *putchar* и *getchar* написать *программу*, позволяющую вызывать **определенным вариантом** функции OpenSBI посредством взаимодействия пользователя через меню.

4. Запустить программу и выполнить вызов пунктов меню, получив результаты их работы.

5. Оформить отчет по работе в электронном формате.

Функции Open SBI

1. Get SBI implementation version

2. Hart get status (должно быть возможно задавать номер ядра)

3. Hart stop

4. System Shutdown

Выполнение

Репозиторий с иерархией реализации и исходным кодом:

```
https://github.com/Imtjl/io-systems
```

Реализация printf:

* (самая сложная по сути часть, в виду невозможности вывода long в простом putchar):

```c
#include "common.h"

void putchar(char ch);

void printf(const char *fmt, ...) {
      va_list vargs;
      va_start(vargs, fmt);

      while (*fmt) {
      if (*fmt == '%') {
            fmt++;            // Skip '%'
            switch (*fmt) { // Считываем следующий символ
            case '\0':        // '%' в конце строки формата.
                  putchar('%');
                  goto end;
            case '%': // Выводим '%'
                  putchar('%');
                  break;
            case 's': { // Выводим NULL-терминированную строку.
                  const char *s = va_arg(vargs, const char *);
                  while (*s) {
                  putchar(*s);
                  s++;
                  }
                  break;
            }
            case 'd': { // Выводим целое число в десятичном формате.
                  int value = va_arg(vargs, int);
                  if (value < 0) {
                  putchar('-');
```

```
                value = -value;
                }

                int divisor = 1;
                while (value / divisor > 9)
                divisor *= 10;

                while (divisor > 0) {
                putchar('0' + value / divisor);
                value %= divisor;
                divisor /= 10;
                }

                break;
          }
          case 'x': { // Выводим целое число в шестнадцатеричном
формате.
                int value = va_arg(vargs, int);
                for (int i = 7; i >= 0; i--) {
                int nibble = (value >> (i * 4)) & 0xf;
                putchar("0123456789abcdef"[nibble]);
                }
          }
          }
     } else {
          putchar(*fmt);
     }

     fmt++;
     }

end:
     va_end(vargs);
}
```

Простая утилитная функция **puts** для вывода длинных строчек с помощью единичного вывода putchar:

```
void puts(const char *s) {
	while (*s) {
	putchar(*s++);
	}
}
```

Реализация putchar & getchar “в лоб”:

*без обработки ошибок, блокирующей операции для удобного ввода и т.п.

```
void putchar(char ch) { sbi_call(ch, 0, 0, 0, 0, 0, 0, 0x1); }
int getchar(void) { return sbi_call(0, 0, 0, 0, 0, 0, 0, 0x02).error; }
```

Организация меню для вызова функций OpenSBI:

```
void kernel_main(void) {
	puts("\nHi there!) It's an interactive menu!\n");
	puts("Chose some OpenSBI commands:\n");
	puts("1. Get SBI implementation version\n");
	puts("2. Hart get status\n");
	puts("3. Hart stop\n");
	puts("4. System Shutdown\n");

	while (1) {
	puts("\n");
	int input;
	while ((input = getchar()) == -1)
		;

	switch (input) {
	case '1': {
		struct sbiret result = sbi_call(0, 0, 0, 0, 0, 0, 2, 0x10);
		int major = (result.value >> 16) & 0xFFFF;
		int minor = result.value & 0xFFFF;

		printf("SBI implementation version: %d.%d\n", major, minor);
		break;
	}
	case '2': {
```

```
            puts("Enter hart ID: ");
            int hart_id;
            while ((hart_id = getchar()) == -1)
                  ;
            putchar(hart_id);
            puts("\n");

            struct sbiret res = sbi_call(hart_id, 0, 0, 0, 0, 0, 2,
0x48534D);

            puts("Hart status:");
            putchar(res.value + '0');
            puts("");
            break;
      }
      case '3': {
            sbi_call(0, 0, 0, 0, 0, 0, 1, 0x48534D);
            break;
      }
      case '4': {
            sbi_call(0, 0, 0, 0, 0, 0, 0, 0x08);
            break;
      }
      default:
            puts("\nplease type in the variant :)\n");
      }

      puts("\ngive me another one, i'm hungry:");
      }
}
```

Спецификация используемых аргументов в вызове sbi_call:

4.3. Function: Get SBI implementation version (FID #2)

```
struct sbiret sbi_get_impl_version(void);
```

Returns the current SBI implementation version. The encoding of this version number is specific to the SBI implementation.

9.3. Function: Hart get status (FID #2)

```
struct sbiret sbi_hart_get_status(unsigned long hartid)
```

Get the current status (or HSM state id) of the given hart in `sbiret.value`, or an error through `sbiret.error`.

The `hartid` parameter specifies the target hart for which status is required.

The possible status (or HSM state id) values returned in `sbiret.value` are described in Table 17.

The possible error codes returned in `sbiret.error` are shown in the Table 21 below.

Table 21. HSM Hart Get Status Errors

Error code	Description
SBI_ERR_INVALID_PARAM	The given `hartid` is not valid.

The harts may transition HSM states at any time due to any concurrent `sbi_hart_start()` or `sbi_hart_stop()` or `sbi_hart_suspend()` calls, the return value from this function may not represent the actual state of the hart at the time of return value verification.

9.2. Function: Hart stop (FID #1)

```
struct sbiret sbi_hart_stop(void)
```

Request the SBI implementation to stop executing the calling hart in supervisor-mode and return its ownership to the SBI implementation. This call is not expected to return under normal conditions. The `sbi_hart_stop()` must be called with supervisor-mode interrupts disabled.

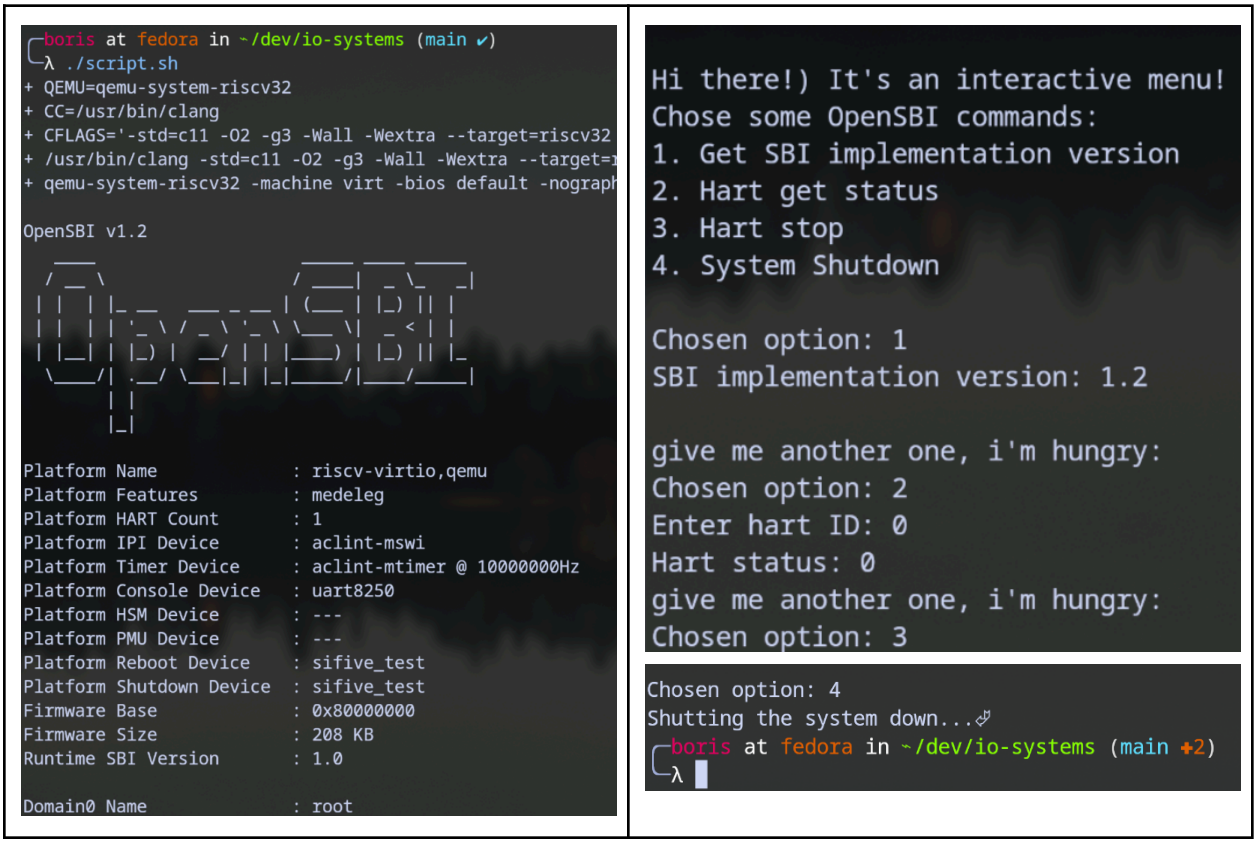
5.9. Extension: System Shutdown (EID #0x08)

```
void sbi_shutdown(void)
```

Puts all the harts to shutdown state from supervisor point of view.

This SBI call doesn't return irrespective whether it succeeds or fails.

Демонстрация работы



Вывод

Тамада хороший и конкурсы интересные. Очень близко к рабочей среде, спасибо за существование этого курса, иначе можно было б помереть со скуки в этом семестре.