Федеральное государственное автономное образовательное

учреждение высшего образования

Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина *«Системы ввода-вывода»*

**Отчет по лабораторной работе №1**

**«Принципы организации ввода/вывода без операционной системы»**

**Вариант: 1**

*Студент:*

Барсуков Максим Андреевич,   
поток 1.3, группа P3315

*Преподаватель:*

Табунщик Сергей Михайлович

г. Санкт-Петербург, 2025 г.

# **Оглавление**

[Введение 3](#_4wv7arx2x4xz)

[Цели работы 3](#_rymypilpit2s)

[Задачи работы 3](#_rfqtgunsc9pf)

[Выполнение 4](#_z5ouzacohqk7)

[Описание функций 4](#_2ag8cldjolhe)

[putchar 4](#_g0geehv59b1r)

[getchar 4](#_n6pwxarm0ayd)

[readline 5](#_s7ki9dkktrot)

[puts 5](#_i42wjguuxgwi)

[printf 6](#_6ex6lndie4a6)

[Описание интерфейса вызова функций OpenSBI, заданных вариантом задания 10](#_v9lkrkv2597u)

[1. Get SBI specification version 10](#_lvoha9b6yi15)

[2. Get number of counters 11](#_yknoj2sf7028)

[3. Get details of a counter 12](#_ndw2fbosvwkj)

[4. System Shutdown 13](#_hbxhh8j8hgkx)

[Демонстрация 14](#_ur5208a6hjk5)

[Вывод 15](#_sbim1ukgwjm0)

# **Введение**

## **Цели работы**

Познакомиться 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 работы**

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.

| № варианта | Пункты меню |
| --- | --- |
| 1 | 1. Get SBI specification version  2. Get number of counters  3. Get details of a counter (должно быть возможно задавать номер счетчика)  4. System Shutdown |

# **Выполнение**

Ссылка на репозиторий: [GitHub](https://github.com/maxbarsukov/itmo/tree/master/6%20%D1%81%D0%B2%D0%B2/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D1%8B%D0%B5/lab1).

## **Описание функций**

### **putchar**

| #define SBI\_ECALL\_0\_1\_PUTCHAR 0x01  void  putchar(char ch)  {  sbi\_call(ch, 0, 0, 0, 0, 0, 0, SBI\_ECALL\_0\_1\_PUTCHAR);  } |
| --- |

Используем легаси расширение, которое выводит символ из аргумента. Является блокирующей функцией, т.е. до вывода текущего символа не будет выведен следующий.

### **getchar**

| #define SBI\_ECALL\_0\_1\_GETCHAR 0x02  #define SBI\_ERR\_FAILED -1  int  getchar(void)  {  struct sbiret ret;  do  {  ret = sbi\_call(0, 0, 0, 0, 0, 0, 0, SBI\_ECALL\_0\_1\_GETCHAR);  } while (ret.error == SBI\_ERR\_FAILED);  return (int) ret.error;  } |
| --- |

Используем легаси расширение, который принимает символ, введённый пользователем. Функция будет ждать до тех пор (путём возврата -1), пока пользователь не введет символ через клавиатуру.

### **readline**

| void  readline(char \*buf, int max\_len)  {  int i = 0;  char c;  while (i < max\_len - 1) {  c = getchar();  if (c == 8 || c == 127) {  if (i > 0) {  i--;  buf[i] = '\0';  putchar(8); // Перемещаем курсор назад  putchar(' '); // Стираем символ пробелом  putchar(8); // Снова перемещаем курсор назад  }  continue;  }  if (c == '\r' || c == '\n') {  putchar('\n');  break;  }  if (c >= 32 && c < 127) {  putchar(c);  buf[i++] = c;  }  }  buf[i] = '\0';  } |
| --- |

Функция реализует ввод строки с консоли с поддержкой удаления символов (backspace).

### **puts**

| void  puts(const char \*s)  {  while (\*s)  {  putchar(\*s++);  }  } |
| --- |

Простая утилитарная функция для посимвольного вывода строки с помощью putchar.

### **printf**

Для реализации printf я написал несколько других простых функций: strlen, reverse, atoi, itoa (см. [common.c](https://github.com/maxbarsukov/itmo/blob/master/6%20%D1%81%D0%B2%D0%B2/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D1%8B%D0%B5/lab1/common.c)).

| void  printf(const char \*fmt, ...)  {  va\_list vargs;  va\_start(vargs, fmt);  while (\*fmt)  {  if (\*fmt == '%')  {  fmt++;  int width = 0;  int zero\_pad = 0;  int left\_align = 0;  while (1)  {  if (\*fmt == '0') zero\_pad = 1;  else if (\*fmt == '-') left\_align = 1;  else break;  fmt++;  }  while (\*fmt >= '0' && \*fmt <= '9')  {  width = width \* 10 + (\*fmt++ - '0');  }  switch (\*fmt)  {  case 'c':  {  char c = (char) va\_arg(vargs, int);  putchar(c);  break;  }  case 's':  {  const char \*s = va\_arg(vargs, const char \*);  if (!s) s = "(null)";  int len = strlen(s);  if (width > len && !left\_align)  {  for (int i = len; i < width; i++)  putchar(zero\_pad ? '0' : ' ');  }  puts(s);  if (width > len && left\_align)  {  for (int i = len; i < width; i++)  putchar(' ');  }  break;  }  case 'd':  case 'i':  {  int num = va\_arg(vargs, int);  char buffer[32];  itoa(num, buffer);  int len = strlen(buffer);  if (width > len && !left\_align)  {  for (int i = len; i < width; i++)  putchar(zero\_pad ? '0' : ' ');  }  puts(buffer);  if (width > len && left\_align)  {  for (int i = len; i < width; i++)  putchar(' ');  }  break;  }  case 'u':  {  unsigned num = va\_arg(vargs, unsigned);  char buffer[32];  itoa((int)num, buffer);  int len = strlen(buffer);  if (width > len && !left\_align)  {  for (int i = len; i < width; i++)  putchar(zero\_pad ? '0' : ' ');  }  puts(buffer);  if (width > len && left\_align)  {  for (int i = len; i < width; i++)  putchar(' ');  }  break;  }  case 'x':  case 'X':  {  unsigned num = va\_arg(vargs, unsigned);  char buffer[32];  char \*p = buffer;  const char \*digits = (\*fmt == 'X') ? "0123456789ABCDEF" : "0123456789abcdef";  if (num == 0)  {  \*p++ = '0';  } else {  while (num > 0)  {  \*p++ = digits[num & 0xF];  num >>= 4;  }  }  \*p = '\0';  reverse(buffer);  int len = strlen(buffer);  if (width > len && !left\_align)  {  for (int i = len; i < width; i++)  putchar(zero\_pad ? '0' : ' ');  }  puts(buffer);  if (width > len && left\_align)  {  for (int i = len; i < width; i++)  putchar(' ');  }  break;  }  case '\0':  {  putchar('%');  goto end;  }  case '%':  {  putchar('%');  break;  }  default:  {  putchar('%');  putchar(\*fmt);  break;  }  }  }  else  {  putchar(\*fmt);  }  fmt++;  }  end:  va\_end(vargs);  } |
| --- |

Функция реализует форматированный вывод с поддержкой спецификаторов:

* %c – символ
* %s – строка
* %d, %i – целое число со знаком
* %u – целое число без знака
* %x, %X – шестнадцатеричное число
* Поддержка ширины вывода и выравнивания

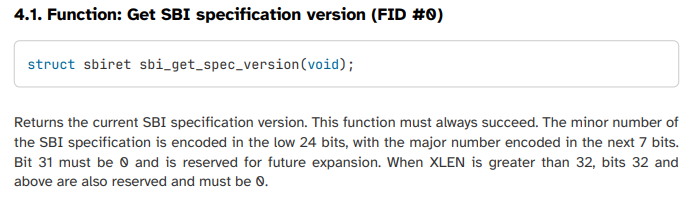
## **Описание интерфейса вызова функций OpenSBI, заданных вариантом задания**

### **1. Get SBI specification version**

| #define SBI\_EXT\_BASE 0x10  #define SBI\_EXT\_VER 0x00  void  get\_sbi\_version()  {  struct sbiret ret = sbi\_call(0, 0, 0, 0, 0, 0, SBI\_EXT\_VER, SBI\_EXT\_BASE);  long major = ret.value >> 24;  long minor = ret.value & 0xFFFFFF;  printf("\nВерсия SBI: %d.", major);  if(minor < 10)  {  putchar('0');  }  if(minor < 100)  {  putchar('0');  }  printf("%d\n", minor);  } |
| --- |

Получает версию спецификации. Minor-версия закодирована в последних 24 битах численного значения, возвращаемого вызовом, а Major — в предстоящих 7 битах. Для этого в функции есть дополнительные битовые операции.

Расширение: 0x10 (Base Extension). Функция: 0x0.

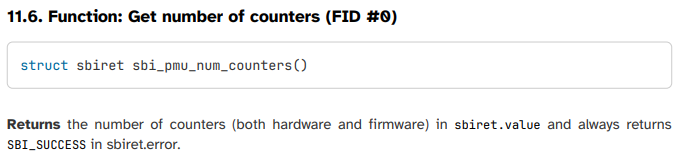


### **2. Get number of counters**

| #define SBI\_EXT\_PMU 0x504D55  #define SBI\_EXT\_CTR\_NUM 0x00  void  get\_num\_counters()  {  struct sbiret ret = sbi\_call(0, 0, 0, 0, 0, 0, SBI\_EXT\_CTR\_NUM, SBI\_EXT\_PMU);  printf("\nЧисло счётчиков: %d\n", ret.value);  } |
| --- |

Получает число счётчиков в ОС. Они доступны в read-only режиме.

Расширение: 0x544D (Timer Extension). Функция: 0x0.

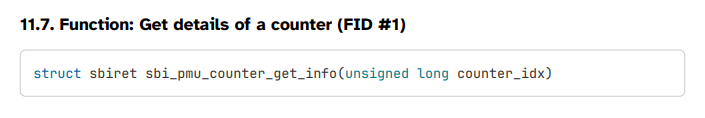


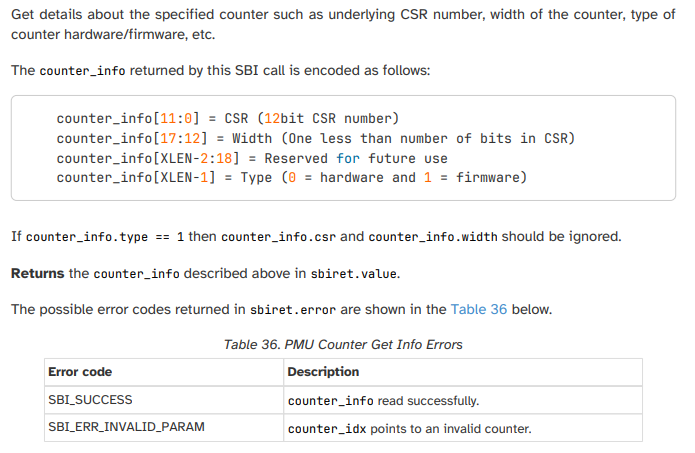
### **3. Get details of a counter**

| #define SBI\_EXT\_PMU 0x504D55  #define SBI\_EXT\_CTR\_DTLS 0x01  void  get\_counter\_details()  {  printf("\nВведите номер счётчика: ");  char input[32];  readline(input, sizeof(input));  long counter\_num = atoi(input);  struct sbiret ret = sbi\_call(counter\_num, 0, 0, 0, 0, 0, SBI\_EXT\_CTR\_DTLS, SBI\_EXT\_PMU);  printf("\nСчётчик: %d\n", counter\_num);  printf("Детали: \n");  if(ret.error)  {  printf("Ошибка: %d\n", ret.error);  return;  }  unsigned long counter\_info = ret.value;  int type = (counter\_info >> (\_\_riscv\_xlen - 1)) & 0x1;  int width = (counter\_info >> 12) & 0x3F;  int csr = counter\_info & 0xFFF;  printf("%4s\* Тип: %s", "", type ? "Прошивка\n" : "Аппаратура\n");  if(!type)  {  printf("%4s\* CSR: %d\n", "", csr);  printf("%4s\* Ширина: %d бит\n", "", width + 1);  }  else  {  printf("%4s\* CSR и ширина не применимы для прошивки.\n", "");  }  } |
| --- |

Получает специфичные детали одного счётчика. Пользователь вводит номер счётчика, который передается в вызов интерфейса. В итоге выводится такая информация, как CSR (Control and Status Register), ширина счётчика и тип (аппаратура или прошивка).

Расширение: 0x544D (Timer Extension). Функция: 0x1.

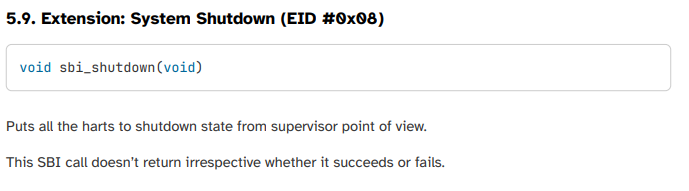




### **4. System Shutdown**

| #define SBI\_EXT\_SRST 0x53525354  #define SBI\_EXT\_SHUTDOWN 0x00  void  system\_shutdown()  {  printf("Завершение работы системы...\n");  printf("\nСпасибо за использование программы!\n");  printf(" |\\ \_,,,---,,\_\nZZZzz /,`.-'`' -. ;-;;,\_\n |,4- ) )-,\_. ,\\ ( `'-'\n '---''(\_/--' `-'\\\_)\n\n"); // Это ASCII-арт котенка  sbi\_call(0, 0, 0, 0, 0, 0, SBI\_EXT\_SHUTDOWN, SBI\_EXT\_SRST);  while(1);  } |
| --- |

Завершает работу системы. Расширение: 0x53525354 (SRST Extension). Функция: 0x0.



## 

## **Демонстрация**

|  |  |
| --- | --- |
|  |  |

# **Вывод**

В ходе выполнения лабораторной работы я познакомился 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.