Код возврата — это: 1. переменная 2. функция 3. литерал 4. _____
#include — это: 1. тип 2. объявление 3. комментарий 4. _____
В результате выполнения gcc -c, получается файл: 1. исполняемый 2. статический 3. байт-код 4. _____
void main() возвращает: 1. число «0» 2. число «1» 3. указатель NULL 4. _____
Прерывание — это: 1. исключение 2. код операции 3. цикл команды 4. _____
libc — это: 1. стандарт 2. компилятор 3. ассемблер 4. _____
.globl _start — это: 1. метка 2. инструкция 3. системный вызов 4. _____
Файл — это: 1. поток байт 2. поток ядра 3. дескриптор 4. _____
O_WRONLY — это: 1. функция 2. макрофункция 3. дескриптор 4. _____
lseek(2) задаёт для файла: 1. размер 2. права доступа 3. id владельца 4. _____
struct stat — это: 1. тип 2. константа 3. массив 4. _____
errno — это: 1. указатель 2. макрос 3. файл 4. _____
STDERR_FILENO — это: 1. поток ошибок 2. буфер записи 3. указатель 4. _____
Makefile — это: 1. директива 2. переменная 3. цель 4. _____
iov — это: 1. тип 2. функция 3. смещение 4. _____
F_SETLKW — это: 1. флаг доступа 2. дескриптор 3. системный вызов 4. _____
fchmod(2) оперирует с: 1. маской 2. файлом 3. процессом 4. _____
Получить текущее значение umask: 1. можно используя syscall 2. невозможно 3. можно прочитав диск 4. _____
unlink(2) удаляет: 1. прямую ссылку 2. косвенную ссылку 3. файл 4. _____
rmdir(2) оперирует с: 1. дескриптором 2. потоком 3. указателем 4. _____
dirent — это: 1. тип 2. функция 3. массив 4. _____
Нода устройства — это:

1. смещение 2. указатель 3. адрес 4. _____
brk(2) изменяет указатель на начало сегмента: 1. стека 2. кода 3. данных 4. _____
Процесс — это: 1. файл 2. сущность вне ядра 3. атомарный набор инструкций 4. _____
В состояние «зомби» нельзя перейти из состояния: 1. «готов» 2. «выполняется» 3. «ожидает» 4. _____
Функция execlp(2) последним принимает: 1. путь к файлу 2. указатель на аргументы 3. указатель на первый аргумент 4. _____
Корректный способ убить «зомби», кроме перезагрузки: 1. послать сигнал 2. вызвать join для потока 3. подождать 4. _____
Модификатор volatile: 1. игнорируется компилятором 2. эквивалентен static 3. объект синхронизации 4. _____
SIG_HOLD — это: 1. стандартная реакция на SIG_INT 2. указатель 3. число 4. _____
ФАП и ЛАП соотносятся: 1. ФАП > ЛАП 2. ЛАП > ФАП 3. ФАП = ЛАП 4. _____
BSD сокет может адресоваться по: 1. порту 2. MAC-адресу 3. ARP-записи 4. _____
Вызов listen(2): 1. создаёт слушающий сокет 2. закрывает соединение 3. принимает данные 4. _____
Вызов sendto(2): 1. эквивалентен write(2) 2. не требует сокет 3. используется для TCP 4. _____
Функция htons(3) преобразует данные в: 1. Big Endian 2. Little Endian 3. Network Endian 4. _____
Переменные *buf и buf[]: 1. ничем не отличаются 2. эквивалентны 3. комплиментарны 4. _____
Функция perror(3): 1. эквивалентна write(2) 2. эквивалентна exit(3) 3. останавливает программу 4. _____
В файле unistd.h присутствуют: 1. функции 2. include-guard 3. макросы 4. _____
Дескриптор файла — это: 1. указатель 2. имя файла 3. ссылка на файл 4. _____
Семафор и мьютекс: 1. эквивалентны 2. взаимозаменяемы 3. служат для IPC 4. _____
Прочитать содержимое файла с правами 000: 1. невозможно 2. можно через mmap(2) 3. может владелец 4. _____

Простенькие задачки:

```
char dotter(int a, int b) {
   int i = 2;
   for(i=0; i<10; i++) {
      if(i==5) {
         if (write(i, "-", 1) < 0)
```

```
            perror("write");
         break;
      }
   }
   return i;
}

int *xor4bit_2(unsigned char data) {
   static int result = 0;
   while (result == 0 && data != 0) {
      result ^= data & 1;
      data >>= 1;
   }
   return result;
}

char send(int s, int rsize) {
   char output[1024] = "Hello!";
   struct sockaddr_in outputa;
   if(sendto(s, output, 512, 0, (struct sockaddr *) &output,
rsize) == SOCKET_ERROR) {
      perror("socket");
   }
   return s;
}

void main(int argv,char *argc[]){
   if(fork() != 0) goto _EXIT;
   close(0);
   return 0;
_EXIT:
   return 0;
}

int main(int argc, char **args) {
   if (args.first ? 1 : 0 + args.second ? 1 : 0 + args.third ? 1 :
0 > 1)
      return "No such args";
   else
      return args;
}

short *default_port(short p) {
   if (p < 0 || p > 65536)
      p = 80;
      return 1; //value replaced
   else
      return 0; //no changes
}

char list_cleanup(void *first) {
   void *p;
```

```
   for(p=first; p!=NULL; p=p->next)
      free(p);
   return p;
}

char* print2() {
   char *buf = malloc(8);
   strcat(buf, "abcdefgh");
   return printf("%s", buf);
}

void file_exist(char *filename) {
   char s[256];
   sprintf(s, "test -e %s", filename);
   if (system(s) == 0){
      return 1;
   } else {
      return 0;
   }
}

int set_country_code(char *code) {
   char buf[2] = {'R', 'U'};
   sprintf(code, "%s", buf);
}

char print() {
   char *src = "Hello!";
   char dst[64]; //output buffer
   int i;
   for(i=0; i<=sizeof(src); i++) {
      dst[i] = src[i];
   }
   printf("%s", dst);
   return i;
}

char getID(){
   uint32_t id = calcuate_id();
   static char str[16] = {0};
   sprintf(str, "%ud", id);
   return str;
}
```