Коллоквиум по ОС с ответами, 2021 год

1. В компьютере используется 8-сегментная модель организации памяти. Описать функцию с заголовком `unsigned VirtIntoPhis(segment* SegTable, unsigned VirtAddr)`, преобразующую виртуальный сегментный адрес (заданный параметром `VirtAddr`) некоторого процесса в физический (возвращается как результат функции).
Все сегменты процесса размещены в физической памяти. Обнаружение ошибки преобразования виртуального адреса в физический должно приводить к завершению программы с кодом 25.
Таблица сегментов реализуется в виде массива структур типа `segment` и передается в программу по указателю `SegTable`.
Описать и прокомментировать все необходимые структуры данных при условии, что физические и виртуальные адреса имеют размер 32 бита.

*Ответ:* Виртуальный адрес содержит номер сегмента и смещение в сегменте. Ввиду использования 8-сегментной организации памяти под номер сегмента отводится три первых бита в виртуальном адресе, а под смещение (offset) – оставшиеся 29 бит.
Номер строки таблицы сегментов – это номер сегмента. Содержимое строки таблицы сегментов – это размера сегмента (size) и адрес начала сегмента (SegAddr).
Физический адрес вычисляется как сумма адреса начала сегмента SegAddr и смещения offset. Прерывание происходит, если смещение больше либо равно размеру сегмента.

```
typedef struct Seg
    unsigned size;     /* размер сегмента */
    unsigned SegAddr;  /* физ. адрес начала сегмента */
} segment;

enum {SegErr=25}; /* код ошибки сегментации */

unsigned VirtIntoPhis(segment* SegTable, unsigned VirtAddr){

    unsigned SegNum = VirtAddr >> 29; /* получаем номер сегмента */
    unsigned offset = VirtAddr & 0x1fffffff; /*смещение в сегменте */
    if (SegTable[SegNum].size > offset)
         return SegTable[SegNum].SegAddr + offset;
    exit(SegErr);
}
```

2. Программное моделирование работы Raid 0, построенного из четырех дисков (вместо реальных дисков используются файлы с именами “Disk0”, “Disk1”, “Disk2”, “Disk3”). Размер блока – `BlockSize` (определенная в программе константа, измеряется в байтах), нумерация блоков начинается с нуля.
Требуется описать: 1) функцию с заголовком `void Init_Raid0(void)` – открытие файлов “Disk0”, “Disk1”, “Disk2”, “Disk3” – считаем, что они существуют, доступны и нужного размера);
2) функцию с заголовком `void Read_Raid0(int num, char * buf)` – чтение блока с номером `num` в буфер по указателю `buf`.

*Ответ:* Raid 0 устроен так, что нулевой блок хранится на нулевом диске, первый – на первом, второй – на втором, третий – на третьем, четвертый – снова на нулевом диске, пятый – снова на первом диске и т.д. Таким образом, для получения номера нужного диска

(файла) нужен остаток от деления номера блока на 4. Для получения смещения в файле нужно разделить номер блока на 4 и умножить его на размер блока.

```
enum {BlockSize=1024}; /* размер блока */

int fd[4];

void Raid0(void){
    char files[4][6]={"Disk0", "Disk1", "Disk2", "Disk3"};
    int i;
    for (i=0; i<4; i++)
        fd[i]=open(files[i], O_RDWR);
}

void Read_Raid0(void){
    int FileNum = num%4;
    int offset = num/4*BlockSize;
    lseek(fd[FileNum],offset,SEEK_SET);
    read(fd[FileNum], buf, BlockSize);
}
```

3. Что будет выведено на экран в результате работы программы? Если возможны несколько вариантов ответа, привести их все и пояснить. Обращение к функции вывода на экран прорабатывает атомарно и без буферизации. Все системные вызовы прорабатывают успешно. Подключение заголовочных файлов опущено.

```
int main(){
    int pid, fd[2];
    char c='a';
    pipe(fd);
    if ((pid = fork()>0) {
        read(fd[0], &c,1);
        kill (pid, SIGKILL);
        wait(NULL);
    }else {
        putchar(c);
        c='b';
        write(fd[1],&c,1);
        c='c';
    }
    putchar(c);
    return 0;
}
```

*Ответ:* *ab* либо *acb.*

4. Сколько раз система обратится к содержимому индексных дескрипторов при вызове: `open("/dir1/dir2/dir3/file", O_RDONLY)` ? Обосновать ответ. Считаем, что ни один из элементов пути к файлу не является символической ссылкой.

*Ответ:* 5

```
1 – inod для корня /, чтобы найти inod для подкаталога "dir1"
2 – inod для /dir1,  чтобы найти inod для подкаталога "dir2"
3 – inod для /dir1/dir2, чтобы найти inod для подкаталога "dir3"
4 – inod для /dir1/dir2/dir3, чтобы найти inod для файла "file"
5 – inod для /dir1/dir2/dir/3/file, чтобы проверить права
доступа и загрузить файл в память.
```