1. В файловой системе используются битовые массивы для хранения информации о свободных и занятиях блоках. Область блоков содержит $N_{блоков}$. Написать функцию, принимающую в качестве параметров указатель на область памяти, в которой находится битовый массив и размер (в блоках) области блоков файловой системы, которая находит выводит на стандартное устройство вывода размер самой большой непрерывной области свободных блоков файловой системы.
2. В файловой системе используются битовые массивы для хранения информации о свободных и занятиях блоках. Область блоков содержит $N_{блоков}$. Написать функцию, принимающую в качестве параметров указатель на область памяти, в которой находится битовый массив и размер (в блоках) области блоков файловой системы, которая находит и выводит на стандартное устройство вывода количество непрерывных областей свободных блоков файловой системы, каждая из которой имеет размер, превышающий размер предшествующей области занятых блоков. Считаем, что область блоков занимает $N_{блоков}$, и что начальные блоки области заняты.
   Считаем, что от начала области располагается массив битов. I-бит содержит информацию о статуса (занят/свободен – как конкретно кодируется, на выбор студента, проблему машинного представления не рассматриваем (т.е. не требуем, какого-либо учета при программировании в каком порядке располагаются байты в переменных различных типов).
3. В компьютере используется 4-х сегментная модель организации памяти. Написать программу, реализующую преобразование виртуального сегментного адреса в физический (считаем, что используются все 4 сегмента). Ситуацию прерывания реализовать отправкой процессом сигнала SigUsr самому себе. Описать в программе все необходимые структуры данных, при учете того, что в машине используются виртуальные и физические адреса, равные по размеру Int.
   Описать таблицу сегментов: массив: <размер сегмента><база>. Описать структуру исполнительного адреса <номер сегмента><смещение>
   Прерывание происходит если База+СМЕЩЕНИЕ >(больше) размера.
4. Пусть дан 32-х разрядный компьютер, использующий страничную модель организации памяти. Размер страницы – 4096 байтов. Написать программу, моделирующий алгоритм преобразования виртуального адреса в физический с использованием инвертированной таблицы страниц. Описать в программе все необходимые структуры данных, при учете того, что в машине используются виртуальные и физические адреса, равные по размеру Int. Считаем, что объем физической памяти, по объему, равен объему виртуального адресного пространства и что размер беззнакового целого 32 бита. Также считаем, что всевозможные виртуальные страницы процесса размещены в физической памяти
   Описываем инвертированную таблицу страниц: Массив структур <PID><№ Вирт. страницы>
   Описываем виртуальный исполнительный адрес: <PID><Виртуальный адрес>
   <Виртуальный адрес> есть: <№ Вирт. страницы> <Смещение> - в беззнаковом целом
   Нужно: вырезать номер страницы из виртуального адреса, найти в таблице совпадение PID и номер страницы – номер записи есть номер физической страницы. Совпадение всегда есть (все странички находятся в памяти).

5. Написать функцию, которая принимает в качестве параметра указатель на массив адресации блоков файла (из индексного дескриптора) которая выводит на стандартное устройство вывода найденный номер блока файла, имеющий минимальное значение. Размер элемента массива – int.
6. Написать функцию, которая принимает в качестве параметра указатель на массив адресации блоков файла (из индексного дескриптора) которая определяет и выводит на стандартное устройство размер файла в блоках. Размер элемента массива – int.
   В программе должно быть как-то определен размер блока в байтах или количество ссылок в одном блоке (здесь была некорректность в постановке задачи, мы ее откорректировали по ходу. Т.е. принимаем все и расчет количества по размеру блока в байтах и заданную константу). Все остальное просто. Перемещаемся по списку. Либо просматриваем весь до конца или доходим до первого NULL (или просто значения 0).
7. Что будет выведено на экран в результате работы фрагмента программы? Если возможны несколько вариантов – привести все. Предполагается, что обращение к функции вывода на экран прорабатывает атомарно и без буферизации. Все системные вызовы прорабатывают успешно. Подключение заголовочных файлов опущено.

   Задача 2

   msgId – идентификатор существующей пустой очереди сообщений.

```
struct
{
  long type;
  char data[1];
} msg;
……..
msg.type = 1; msg.data[0] = 'a'; msgsnd(msgId, &msg, 1, 0);
msg.type = 2; msg.data[0] = 'b'; msgsnd(msgId, &msg, 1, 0);
msg.type = 2; msg.data[0] = 'c'; msgsnd(msgId, &msg, 1, 0);
msg.type = 1; msg.data[0] = 'd'; msgsnd(msgId, &msg, 1, 0);
msgrcv(msgId, &msg, 1, 2, 0); putchar(msg.data[0]);
msgrcv(msgId, &msg, 1, 0, 0); putchar(msg.data[0]);
msgrcv(msgId, &msg, 1, 1, 0); putchar(msg.data[0]);
……
```

8. Что будет выведено на экран? Если возможны несколько вариантов – привести все. Предполагается, что обращение к функции вывода на экран прорабатывает атомарно и без

буферизации. Все системные вызовы прорабатывают успешно. Подключение заголовочных файлов опущено.

```
int main()
{
	pid_t pid;
	int fd[2];
	int x = 3;
	pipe(fd);
	if( (pid = fork()) > 0 ) { read(fd[0], &x, sizeof(int));  kill(pid, SIGKILL);
wait(NULL);
	}
	else { printf("%d", x); x = 2; write(fd[1], &x, sizeof(int)); x = 1;
	}
	printf("%d", x);
	return 0;
}
```