Занятие 11 сентября 2024

Разбор коллоквиума от 07.12.24

Задача 1

В 32-разрядной вычислительной системе используется 16-сегментная модель организации памяти.

Описать необходимые структуры данных и реализовать на языке Си функцию, реализующую преобразование виртуального адреса, задаваемого первым параметром, в физический (возвращается как результат функции).

Таблица сегментов в виде массива структур передается во втором параметре.

Размер виртуального и физического адреса равен 32 бита.
При обнаружении ошибки преобразования адреса завершить программу с кодом 13.

Решение задачи 1

Виртуальный адрес – это номер сегмента и смещение в сегменте. Поскольку используется 16-сегментная организация памяти, то под номер сегмента отводится 4 первых бита в виртуальном адресе, а под смещение (offset) – оставшиеся 28 бит.

Номер строки таблицы сегментов – это номер сегмента.

Содержимое таблицы сегментов – это размер сегмента (size) и адрес начала сегмента (SegAdr).

Физический адрес – это адрес начала сегмента (SegAdr) + смещение в сегменте (offset).

Прерывание происходит, если смещение больше либо равно размеру сегмента.

Представим элементы таблицы как структуры вида (могут быть в другом порядке):

```
typedef  struct Seg {        unsigned  size;
                             unsigned  SegAdr;
} segment;
unsigned int VirtIntoPhys (unsigned int VirtAdr,  segment * SegTable) {
        unsigned int SegNum = VirtAdr >> 28;
        unsigned int offset = VirtAdr & 0x0fff  ffff;
        if (SegTable [SegNum].size > offset){ return SegTable [SegNum].SegAdr + offset; }
        exit(13); }
```

За использование int в вычислениях, приводящих к ошибкам (побитовые сдвиги, сложения) –НЕ ЗАСЧИТЫВАЕМ.

Задача 2

В файловой системе учет свободных блоков реализован на основе битового массива, общее число блоков ФС равно NUM_BLOCKS.

Реализовать на языке Си функцию, возвращающую номер свободного блока ФС, ближайшего к номеру блока, заданного в виде первого параметра функции.

Второй параметр функции – указатель на битовый массив, представленный в виде массива байт.

В случае отсутствия свободных блоков возвращается -1.

Решение задачи 2

```
int find_near (unsigned int  num, unsigned char * bitBlocks){
        int size = (NUM_BLOCKS + 7) / 8;  // кол-во байтов в массиве
        int dist = NUM_BLOCKS,  near = -1,  cur_pos = 0;
        for (int i = 0; i < size; i++) {
                unsigned int  tmp = bitBlocks [i];
                for (int j = 0; j < 8; j++){
                        cur_pos = i * 8 + 7 - j;
                        if (cur_pos == NUM_BLOCKS) { return near; }
                        if ((tmp & 1U) == 1){
                                if (abs(cur_pos - num) < dist){
                                        near = cur_pos;
                                        dist = abs(cur_pos - num); } }
                        tmp = tmp >> 1; }
        } return near;  }
```

Основное требование – обработка массива как набора бит (т.е. каждый байт отвечает за 8 блоков ФС). Засчитываем обработку как слева направо, так и справа налево. Возможны вариации – например, проверка на то, что переданный номер блока может уже быть свободным (расстояние =0), и др. Если решение без неё – тоже засчитываем. int (если не приводит к ошибкам) – засчитываем

Задача 3

Реализовать на языке Си программу «двухпроцессный будильник».

Процесс порождает один процесс и затем:

Родительский процесс – через заданные константой промежутки времени отсылает сыновьему процессу сигнал таймера.

Процесс - потомок– ожидает ввода непустой строки со стандартного ввода, печатая при этом на стандартный вывод напоминание «жду ввода:» при поступлении каждого сигнала таймера; после ввода непустой строки печатает подтверждение, посылает родительскому процессу сигнал завершения и завершается сам.

Решение задачи 3

```
void alr (int s) {  printf  ("Жду ввода\n");   } //обработчик сигнала SIGALRM
#define   DELTA  5
int main(void) {
        int pid, buf_size = 80;
        signal (SIGALRM, alr);
        if (pid = fork()){
                while (1){   sleep(DELTA);
                             kill(pid, SIGALRM); }
        } else {
                char s[buf_size];
                while (1){ if (fgets(s, buf_size, stdin) != NULL){ break; } }
                printf  ("Ok\n");
                kill (getppid(), SIGKILL); }
        return 0; }
```

Возможны варианты (signal устанавливается сразу в сыне, считывание через gets и другие модификации, в целом удовлетворяющие условию) – засчитываем. Перепутали/забыли константу SIGALRM – засчитываем Считывание (gets/fgets/scanf…) БЕЗ цикла вокруг - НЕ ЗАСЧИТЫВАЕМ.

Задача 4

Что будет выведено на экран? Привести все возможные варианты. Кратко обосновать ответ.

```
pthread_mutex_t  mt = PTHREAD_MUTEX_INITIALIZER;

void * fn (void * arg) {  pthread_mutex_unlock ( & mt);
                          printf  (“%d\n”, *(int *) arg);
                          pthread_mutex_lock ( & mt);  return 0; }


int main () {      int i = 0;
                   pthread_t  th_id;
                   pthread_mutex_lock ( & mt);
                   pthread_create ( &th_id, NULL, fn, &i);
                   i++;
                   pthread_create ( &th_id, NULL, fn, &i);
                   return 0; }
```

Решение задачи 4

Поведение программы в общем случае **не определено**, так как мьютекс захватывается одной нитью, а освобождается другой.

Если данный факт указан, но приведен и вывод на экран – засчитывать.

Задача 5

В вычислительной системе используется оперативная память с характеристиками $\mathbf{t_{access}}$ (время доступа) и $\mathbf{t_{cycle}}$ (длительность цикла памяти).

Каковы будут оценки времени, необходимого для чтения 5 последовательно идущих ячеек памяти, для каждой из трех моделей реализации ОЗУ:

а) без расслоения памяти;

б) с расслоением памяти на 4 банка и одним общим контроллером;

в) с расслоением памяти на 4 банка и независимыми контроллерами каждого банка.

Решение задачи 5

а) 5 * t_cycle; (вариант: 4 * t_cycle + t_access засчитываем)

б) 4 * t_access + t_cycle;

в) t_access + t_cycle

Задача 6

В 32-разрядной вычислительной системе по адресу A размещено целое число (размером 4 байт) со значением, составляющим последовательность символов в виде константы UNIX.

Указать побайтовое содержимое машинного слова для архитектуры с прямым порядком байт (big-endian) и архитектуры с обратным порядком (little-endian).

Решение задачи 6

Big Endian: UNIX

‘U’	‘N’	‘I’	‘X’

Little Endian: XINU

‘X’	‘I’	‘N’	‘U’

Задача 7

На примере файловой системы Ext4 Linux описать пошаговый алгоритм поиска индексного дескриптора для существующего файла с абсолютным путём «/Dir1/Dir2/File».

Проверка прав доступа опущена.

Решение задачи 7

/Dir1/Dir2/File

1) Считываем ИД номер 2 (предопределенный номер ИД – корневой каталог «/») из области индексных дескрипторов.
2) получаем доступ к содержимому корневого файла-каталога (используя массив адресов блоков ИД 2), осуществляем поиск записи с именем «Dir1» в файле корневого каталога, номер индексного дескриптора в этой записи => ИД_Dir1.
3) Считываем содержимое ИД_Dir1, получаем доступ к содержимому файла-каталога /Dir1 (используя массив адресов блоков ИД_Dir1), осуществляем в нём поиск записи «Dir2», номер индексного дескриптора в этой записи => ИД_Dir2.
4) Считываем содержимое ИД_Dir2, получаем доступ к содержимому файла-каталога /Dir1/Dir2 (используя массив адресов блоков ИД_Dir2) и осуществляем поиск записи с именем «File», номер индексного дескриптор в этой записи – искомый ИД_File.

При оптимизации («склеивание» повторяющихся действий и т.п.), но корректном изложении алгоритма – засчитываем. Вместо значения 2 достаточно указания, что это константа. При отсутствии указания, что ИД корневого каталога всегда известен (константа) - НЕ ЗАСЧИТЫВАЕМ