7-165

М.П.

Подпись студента (курсанта) _____

_____ 20____ г.
(дата выдачи зачетной книжки)

1

(учредитель)
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МГТУ ИМ. Н.Э. БАУМАНА

(полное наименование организации, осуществляющей образовательную деятельность)

ЗАЧЕТНАЯ КНИЖКА № 189976

Турсунов Жасурбек Рустамович
(фамилия, имя, отчество (последнее – при наличии) студента (курсанта))

Код, направление подготовки (специальность) 09.03.04 - 01

Структурное подразделение ИУ

Зачислен приказом от _____ 20____ г. № _____

Руководитель организации, осуществляющей образовательную деятельность или иное уполномоченное им должностное лицо _____ М.П. А.В.Пролетарский
(подпись) (фамилия, имя, отчество (последнее – при наличии))

Руководитель структурного подразделения А.В.Пролетарский
(подпись, фамилия, имя, отчество (последнее – при наличии))

2

Экзаменационный лист
22 января 2021 г.
Начало: 16:27
Окончание:
Оценка:
Дисциплина: Операционные системы
Студент: Турсунов Ж. Р.
Экзаменатор: Рязанова Н. Ю.
Группа: ИУ7-56Б
Билет: 9

1. Виртуальная память: управление памятью страницами по запросу - три схемы преобразования виртуального адреса к физическому. Реализация страничного преобразования в компьютерах на базе процессоров Intel (x86): стандартное преобразование и PAE в защищенном режиме - схемы, размеры таблиц и их кол-во на каждом этапе преобразования.

Виртуальная память - память, размер которой превышает размер реального физического пространства. Используется адресное пространство диска как область свопинга или пейджинга т.е. для временного хранения областей памяти.

Подходы к реализации управления виртуальной памятью:
1) управление памятью страницами по запросам
2) сегментами по запросам
3) сегментами, делящимися на страницы по запросам.

Страница - является единицей физического деления памяти. Её размер устанавливается системой. У страницы размер необязательно 4 Кб. Просто такой размер оптимален по кол-ву страничных прерываний.

Сегмент - является единицей логического деления памяти. Её размер определяется объемом кода. Любой сегмент создается такого размера, который требуется программе.

Запрос - страничное прерывание, которое возникает при попытке доступа к незагруженной в память странице. При его обработке происходит пейджинг.

Пейджинг - загрузка с диска новых и замещение старых страниц

Методы преобразования адресов:

1) Прямое отображение таблиц страниц.

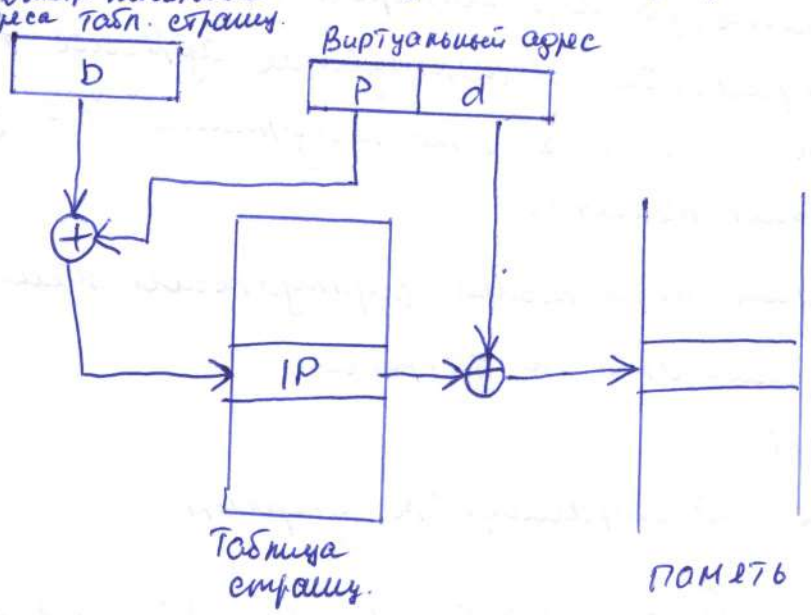
Каждый процесс должен иметь собственную таблицу страниц. Главная таблица - таблица (современные ОС оперируют связными списками) процессов.

Таблица состоит из дескрипторов процессов.

В прямом отображении таблиц страниц - системные таблицы хранятся в оперативной памяти в системной области.

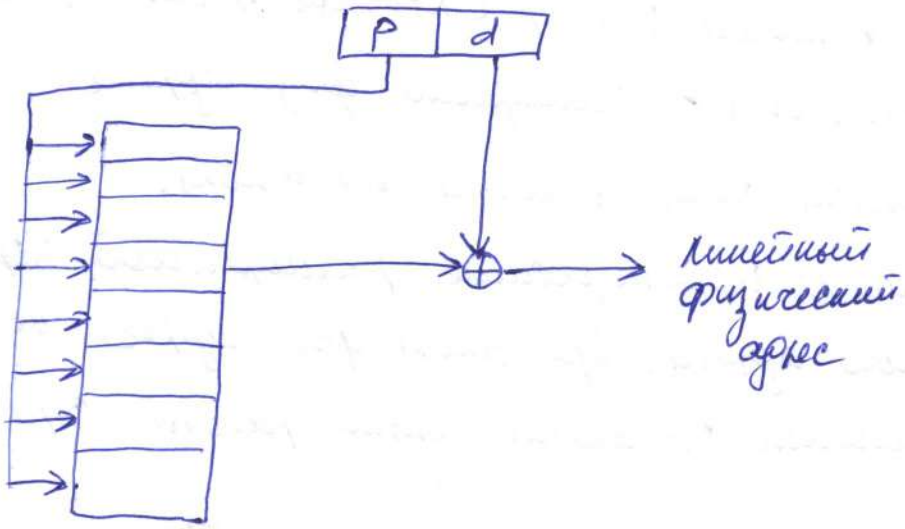
В процессоре должен быть регистр начального адреса таблицы страниц.

Если страница загружена в физ. память, то она имеет физ. адрес



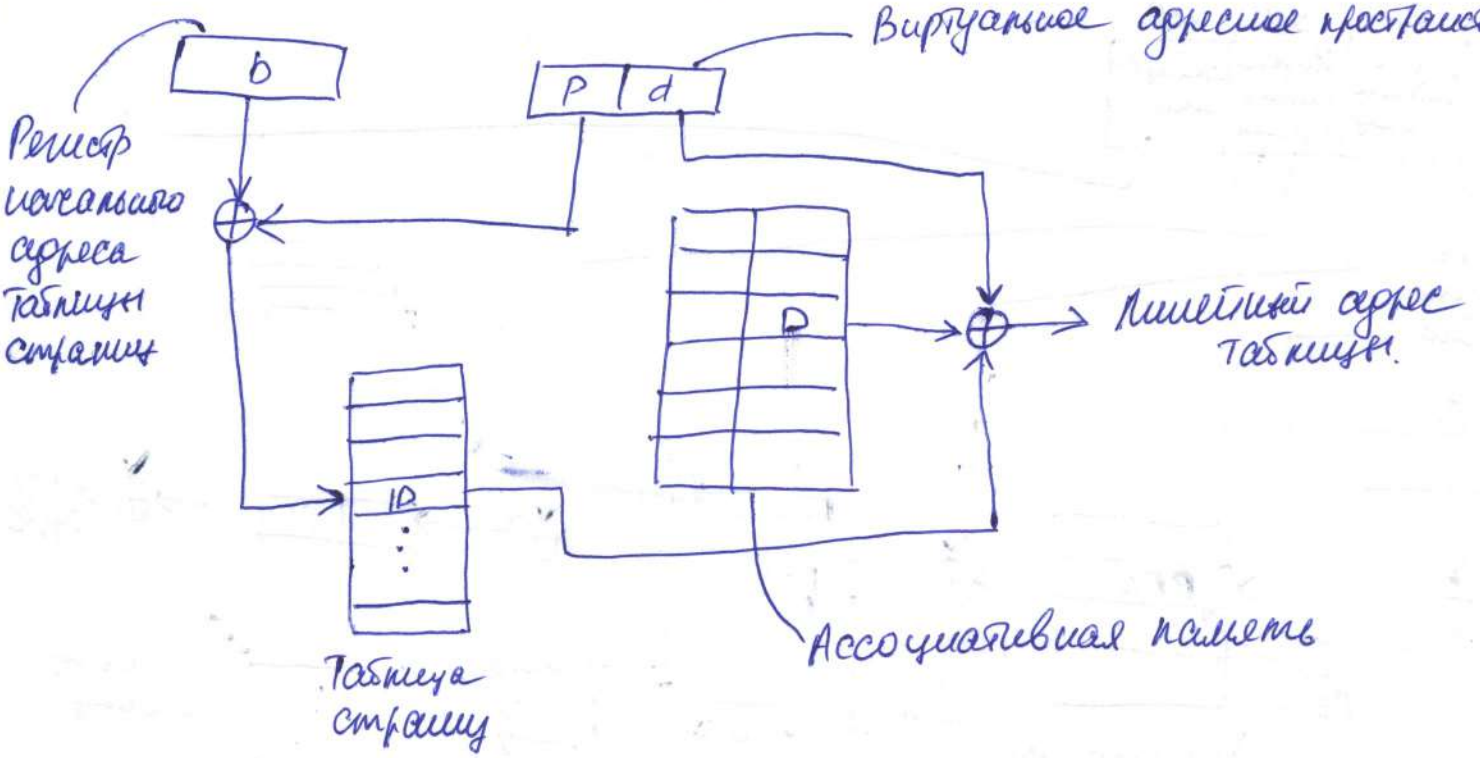
Преобразование поддерживается аппаратно.

2) Ассоциативное отображение (для сокращения обращений к памяти)
Использование ассоциативной памяти - память, которая обеспечивает выборку по ключу за 1 такт



Дорогая память, всегда регистровая количество схем увеличивается, много соединений. Практически не используется.

3) Ассоциативно прямое отображение.



Ассоциативный кэш (последние обращения) - хранятся физические адреса страниц, используемых недавно.

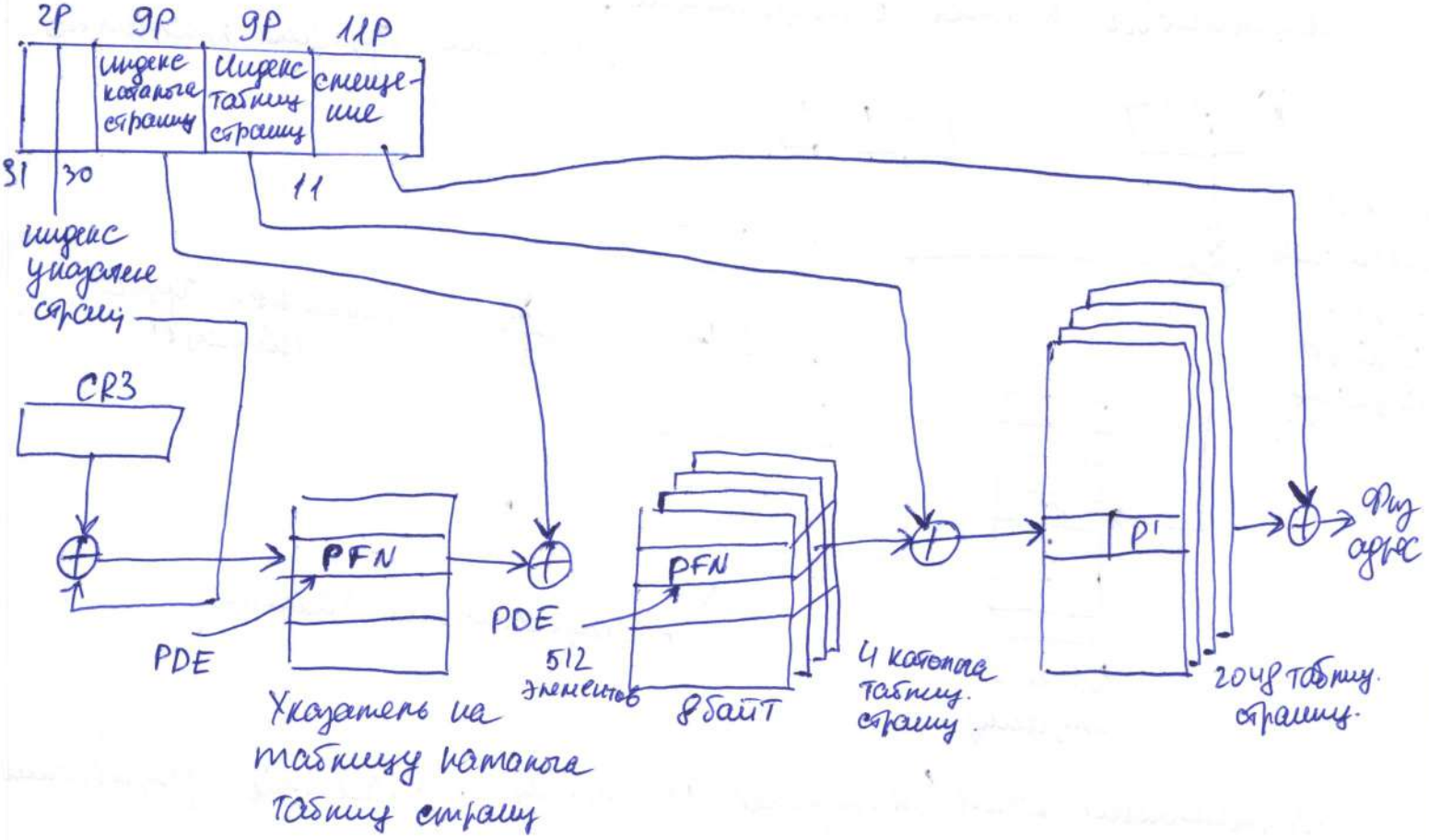
Существует небольшая по объему ассоциативная память

Эвристическое предпочтение. Если было обращение к странице, то следующее будет к той же странице.

Системы обеспечивают скоростные показатели 90-% и более в отличие от полностью с прямым отображением

Начиная с Pentium Pro включен регистр CR4, где 5 бит это PAE (physical adress extension) - расширение физ. адреса.
В режиме PAE виртуальный адрес делится на 4 поля.
Если PAE = 1, то разрешено использование расширенной 36рр, вместо 32 рр физического адреса. При этом физ. адрес остается 32 рр, все изменения касаются только работы страничного механизма.

PFN - Page Frame Number
PDE - Page Describe Entry



② Unix: концепция процессов; иерархия процессов, процессы - "сироты", процессы "зомби", демоны; примеры из л/р (2 Unix)

Процесс в Unix является базовой. Каждый процесс имеет собственное адресное пространство, в котором находятся все сегменты процесса, код, данные, стек. При этом с точки зрения Unix процесс часть времени выполняет собственный

код и тогда он выполняется в режиме задач (use mode), а часть времени код выполняет резидентный код ОС и тогда это режим ядра (Kernel mode).

В Unix любой процесс создается системным вызовом fork. Любой процесс может создать любое кол-во процессов (вызовов fork)

В результате создается иерархия процессов в отношении предок-потомок (parent-child).

В Unix процесс, все его дочерние процессы и более отдаленные потомки образуют группу процессов. Когда пользователь отправляет сигнал с клавиатуры, тот достигает всех участников этой группы процессов, связанных на тот момент времени с клавиатурой. Каждый процесс по отдельности может захватить сигнал, игнорировать его или совершить действие по умолчанию, которое должно быть уничтожено сигналом.

В качестве другого примера, поясняющего ту роль, которую играет иерархия процессов, рассмотрим, как UNIX инициализирует саму себя при запуске. В загрузочном образе присутствует служ. процесс, называемый init. В начале своей работы он считывает файл, сообщающий о кол-ве терминалов. Затем он разветвляется, порождая по одному процессу на каждый терминал. Эти процессы ждут, пока кто-нибудь не зарегистрируется в системе. Если регистрация проходит успешно, процесс регистрации порождает оболочку, для приема команд. Эти команды могут породить другие процессы, и т.д. Таким образом, все процессы во всей системе принадлежат единому дереву, в корне которого находится процесс init.

Процессы демоны - процессы которые не имеют родителей, они существуют сами и не входят ни в какие группы. Демон - процесс, выполняющий какую-то фоновую ~~работу~~ задачу.

не имеющий управляющего терминала и, как следствие, обычно не интерактивный по отношению к пользователю.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define ANSI_COLOR_GREEN "\x1b[32m"
#define ANSI_COLOR_RESET "\x1b[0m"

int main()
{
  int status;
  pid_t childpid1, childpid2;
  childpid1 = fork();
  if (childpid1 == -1)
  {
    perror("Can't fork.\n");
    return 1;
  }
  else if (childpid1 == 0)
  {
    if (execl("/bin/ls", "ls", "-lah", 0) == -1)
    {
      perror("Can't exec.\n");
      exit(1);
    }
  }
  childpid2 = fork();
  if (childpid2 == -1)
  {
    perror("Can't fork.\n");
    return 1;
  }
  else if (childpid2 == 0)
  {
    if (execl("/bin/cat", "cat", "maketile", 0) == -1)
    {
      perror("Can't exec.\n");
      exit(1);
    }
  }
```

```
else
{
    wait(&status);
    if (WIFEXITED(status))
    {
        printf(ANSI_COLOR_GREEN "child process exit
            success\n" ANSI_COLOR_RESET);
    }
}
return 0;
}
```

Процесс-сирота - процесс, у которого завершился предок.

Предок завершился, если не выполнил действий иерархия будет нарушена. Процесс сирота усыновляется терминальным процессом.

При завершении любого процесса система проверяет не осталось ли у него незавершенных потомков (по дескриптору процеса, где есть указатели на потомков). Если такие остались, то выполняется процесс усыновления, фактически путем указателя процесс-потомок получает указатель на нового процесса-пре

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    pid_t childpid1, childpid2;
    childpid1 = fork();
    if (childpid1 == -1)
    {
        perror("Can't fork.\n");
        return 1;
    }
    else if (childpid1 == 0)
    {
        printf("CHILD1 \npid: \%d; ppid: \%d; gid: \%d\n\n
            getpid(), getppid(), getgid());
```

```
getchar();
printf("CHILD1 \npid : %d ; ppid : %d ; gid: %d \n\n",
        getpid(), getppid(), getgid());
return 0;
}
if (childpid2 == -1)
{
  perror("Can't fork. \n");
  return 1;
}
else if (childpid2 == 0)
{
  printf("CHILD2 \npid : %d ; ppid : %d ; gid : %d \n\n",
  getpid(), getppid(), getgid());
  getchar()
  printf("CHILD2 \npid : %d ; ppid : %d ; gid : %d \n\n",
  getpid(), getppid(), getgid());
  return 0;
}
else
{
  printf("PARENT \npid : %d ; ppid : %d ; gid : %d \n\n",
        getpid(), getppid(), getgid());
  getchar();
}
return 0;
}
```