

Вопросы к экзамену по «Управлению ресурсами».

1. Прерывание - это
 1. сигнал, прерывающий работу внешнего устройства и сообщающий о необходимости выполнить некоторую работу;
 - ✓ 2. сигнал, прерывающий работу центрального процессора и сообщающий о необходимости выполнить некоторую работу;
 3. сигнал, генерируемый аппаратурой;
 4. реакция на выполнение команды `int`.
2. Таблица векторов прерываний - это
 - ✓ 1. область, где хранятся адреса программ обработки прерываний;
 2. область старших адресов оперативной памяти;
 3. часть сегмента данных;
 4. область префикса программного сегмента;
 5. область, где хранятся параметры программ обработки прерываний.
3. Длина таблицы прерываний
 1. 512 б;
 - ✓ 2. 1024 б;
 3. 256 б;
 4. 4096 б.
4. Функция `25h` прерывания `21h` служит для
 1. защиты таблицы векторов прерываний;
 - ✓ 2. установки вектора прерываний;
 3. определение критической секции программы;
 4. чтение вектора прерывания;
 5. задание режима обработки прерываний.
5. Функция `35h` прерывания `21h` служит для
 1. защиты таблицы векторов прерываний;
 2. установки вектора прерываний;
 3. определение критической секции программы;
 - ✓ 4. чтение вектора прерывания;
 5. задание режима обработки прерываний.
6. Прерывание `9h` генерируется *(встроенным устройством)*
 1. любым периферийным устройствам;
 2. центральным процессором;
 3. программой пользователя;
 - ✓ 4. клавиатурой;
 5. операционной системой.
7. Анализ скан-кода выполняется
 1. прерыванием `16h`;
 2. прерыванием `21h`;
 3. любым прерыванием;
 - ✓ 4. прерыванием `9h`.
8. Код ASCII имеет длину
 1. 4 байта;
 2. 8 байт;
 - ✓ 3. 1 байт;
 4. 2 байта;
 5. переменная длина.
9. Расширенный код имеет длину
 1. 4 байта;
 2. 8 байт;
 3. 1 байт;
 - ✓ 4. 2 байта;
 5. переменная длина.
10. Каждому введенному символу в буфере клавиатуры соответствует

1. 1 байт;
 - ✓ 2. 2 байта;
 3. 1 или 2 байта;
 4. число байт, зависящее от вводимого символа.
11. Информация о первом введенном символе записывается в буфер клавиатуры по адресу 0040:001E, о втором - по адресу 0040:0020, . . . , о пятнадцатом - по адресу 0040:003A, о шестнадцатом - по адресу
1. 0040:003B;
 - ✓ 2. 0040:001E;
 3. 0040:001A;
 4. 0040:003C;
 5. 0040:001C.
12. Неравенство значений по адресам 0040:001A и 0040:001C свидетельствует о
1. отсутствии символа в буфере клавиатуры;
 2. наличии символа в памяти клавиатуры;
 3. отсутствии символа в памяти клавиатуры;
 - ✓ 4. наличии символа в буфере клавиатуры.
13. Длина префикса программного сегмента (PSP) составляет
- ✓ 1. 100h байт;
 2. 100 байт;
 3. 128 байт;
 4. 128h байт.
14. Область DTA в PSP содержит
1. номер параграфа строки среды;
 2. область параметров для метода FCB;
 - ✓ 3. командную строку программы;
 4. нечто иное.
15. Минимальный объем динамически запрашиваемой памяти равен
- ✓ 1. 16 байт;
 2. 1 байт;
 3. 128 байт;
 4. 256 байт.
16. Функция 4Ah прерывания 21h служит для
1. выделения блока памяти;
 - ✓ 2. изменения размера памяти, отведенного программе;
 3. освобождения блока памяти;
 4. для запуска одной программы из другой.
17. Строка среды в блоке параметров при динамическом вызове одной программы из другой
- ✓ 1. спецификации, используемые в файле config.sys;
 2. сведения из PSP;
 3. команды, используемые в файле autoexec.bat;
 4. параметры функции 4Bh;
 5. нечто иное.
18. Для передачи командной строки в динамически вызываемую программу используется
1. поле из PSP;
 - ✓ 2. поле блока параметров;
 3. строка с полным именем запускаемой программы;
 4. нечто иное.
19. В текстовом режиме каждой позиции экрана соответствует в памяти
1. 1 бит;
 2. 2 бита;

- 3. 4 бита;
- 4. 1 байт;
- 5. 4 байта;
- ✓ 6. 2 байта.

20. Среди указанных ниже операций единственной операцией, выполняемой в графическом режиме является

- 1. вывод строки символов;
- 2. перемещение курсора в заданную точку;
- 3. чтение положения курсора;
- 4. задание новой активной страницы;
- 5. прокрутка активной страницы;
- 6. чтение символа из текущей позиции курсора и его атрибута;
- 7. запись символа и его атрибута в текущую позицию курсора;
- 8. запись символа без изменения атрибута в текущую позицию курсора;

✓ 9. задание видеорежима.

21. Среди указанных ниже операций единственной операцией, выполняемой средствами DOS является

- ✓ 1. 1. вывод строки символов;
- 2. перемещение курсора в заданную точку;
- 3. чтение положения курсора;
- 4. задание новой активной страницы;
- 5. прокрутка активной страницы;
- 6. чтение символа из текущей позиции курсора и его атрибута;
- 7. запись символа и его атрибута в текущую позицию курсора;
- 8. запись символа без изменения атрибута в текущую позицию курсора;
- 9. задание видеорежима.

22. Один элемент FAT-таблицы соответствует

- ✓ 1. одному кластеру;
- 2. одному блоку;
- 3. одному элементу оглавления;
- 4. одному файлу;
- 5. одному сектору.

23. Прерывания 13h, 25h, 26h используются

- 1. в методе FCB;
- 2. в методе дескриптора файла;
- ✓ 3. для работы с отдельными секторами;
- 4. при работе с физической нумерацией диска;
- 5. при работе с логической нумерацией диска.

24. Метод FCB (блок управл-я файлами)

- 1. вызывается через прерывание BIOS;
- 2. работает с отдельными секторами;
- 3. использует идентификационный номер файла;
- ✓ 4. работает только с файлами текущей директории;
- 5. служит для работы с любыми периферийными устройствами.

25. Метод дескриптора файла

- 1. вызывается через прерывание BIOS;
- 2. работает с отдельными секторами;
- ✓ 3. использует идентификационный номер файла;
- 4. работает только с файлами текущей директории;
- 5. служит для работы с любыми периферийными устройствами.

26. В системной фазе могут выполняться

- 1. только процессы ядра ОС UNIX;
- ✓ 2. любые процессы ОС UNIX;
- 3. только пользовательские процессы;
- 4. только процессы интерпретатора команд.

27. Критическая секция служит для

1. обеспечения целостности данных пользователя;
2. реализации механизма событий (семафора);
3. учета процессов;
4. реализации системных вызовов;
- ✓ 5. обеспечения целостности данных ядра.

28. Процесс - это

1. объект, созданный в результате выполнения системного вызова `exec()`;
2. объект, созданный интерпретатором команд;
3. объект, созданный процессом ядра;
- ✓ 4. объект, созданный в результате выполнения системного вызова `fork()`;
5. нечто иное.

29. Мультипрограммирование в ОС UNIX - это

1. управление последовательностью выполнения свопинга;
2. управление файловой системой;
- ✓ 3. управление последовательностью выполнения процессов и последовательностью выполнения свопинга;
4. управление последовательностью выполнения свопинга и файловой системой;
5. управление последовательностью выполнения процессов.

30. Свопинг - это

- ✓ 1. перемещение процессов из оперативной памяти на диск и ввод их по мере необходимости обратно;
2. управление процессами в оперативной памяти;
3. управление процессами во внешней памяти;
4. управление внешней и оперативной памятью;
5. нечто иное.

31. Описатель файла содержит, в частности, информацию о

- ✓ 1. типе файла, его размере и адресах блоков данных, составляющих файл;
2. количестве свободных блоков файловой системы;
3. типе файла, его размере и идентификатор;
4. типе файла, его размере и имени.

32. При запросе последнего блока из списка в суперблоке (`s_free`)

1. в данный блок переписывается содержимое массива `s_free`, выполняется сортировка и блок включается в цепочку;
- ✓ 2. содержимое этого блока переписывается в массив `s_free`;
3. просматривается таблица блоков данных для поиска свободных;
4. в данный блок переписывается содержимое массива `s_free` и он включается в цепочку;
5. содержимое этого блока переписывается в массив `s_free` и выполняется сортировка.

33. При освобождении блока в случае заполненности списка в суперблоке

1. в данный блок переписывается содержимое массива `s_free`, выполняется сортировка и блок включается в цепочку;
2. содержимое этого блока переписывается в массив `s_free`;
3. просматривается таблица блоков данных для поиска свободных;
- ✓ 4. в данный блок переписывается содержимое массива `s_free` и он включается в цепочку;
5. содержимое этого блока переписывается в массив `s_free` и выполняется сортировка.

34. После открытия первым процессом файлов `file1`, `file2`, `file3`, `file4`, вторым процессом - файлов `file1`, `file2`, третьим - файла `file2`, число записей в таблице файлов равно

1. 3;
2. 4;
- ✓ 3. 7;
4. некоторому другому значению.

35. После открытия первым процессом файлов `file1`, `file2`, `file3`, `file4`, вторым процессом - файлов `file1`, `file2`, третьим - файла `file2`, число записей в таблице описателей файлов равно

1. 3;

v 2. 4;

3. 7;

4. некоторому другому значению.

36. После открытия процессом двух файлов число записей в таблице открытых файлов процесса равно (2 откр-ых и 3 закр-ых)

1. 2;

2. 0;

3. 3;

v 4. 5;

5. некоторому другому значению.

37. После открытия первым процессом файлов file1, file2, file3, file4, вторым процессом - файлов file1, file2, третьим - файла file2 и последующим закрытием вторым процессом всех открытых ранее им файлов, максимальное значение счетчика в таблице описателей файлов равно

1. 1;

2. 3;

v 3. 2;

4. 4;

5. 5;

6. 7.

38. После открытия процессом двух файлов и создания канала число записей в таблице открытых файлов равно

v 1. 7;

2. 6;

3. 4;

4. 3;

5. некоторому другому значению.

39. После открытия процессом двух файлов и использования системного вызова dup() с параметром, являющимся номером дескриптора одного из открытых ранее файлов, число записей в таблице открытых файлов равно

1. 3;

v 2. 6;

3. 4;

4. 7;

5. некоторому другому значению.

40. Исходный файл содержит последовательно по 128 значений "a", "b", "c", "d", "e", "f" и т.д. После открытия файла и получения копии дескриптора файла по системному вызову dup() с использованием оригинального дескриптора файла выполнено чтение двух записей по 128 байт, а с использованием копии дескриптора файла выполнено чтение трех записей по 128 байт. Последним прочитанным из файла символом является

1. "a";

2. "b";

3. "c";

4. "d";

v 5. "e";

6. "f".

41. Для обмена двух процессов данными через программный канал минимальный набор системных вызовов составляет

1. open(), read(), write(), close();

2. pipe();

v 3. pipe(), read(), write();

4. pipe(), dup(), read(), write();

5. dup(), read(), write().

42. Обработка сигналов выполняется

- ✓ 1. при переходе из режима ядра в режим задачи;
- 2. при переходе в режим приостанова;
- 3. при выходе из режима приостанова;
- 4. при переходе из режима ядра в режим задачи и обратно;
- 5. при переходе из режима задачи в режим ядра.

43. Исходный файл содержит последовательно 128 значений "a", "b", "c", "d", "e", "f" и т. д. Программа дважды открывает указанный файл и читает с использованием первого дескриптора две записи по 128 байт, а затем с использованием второго дескриптора три записи по 128 байт. Последним прочитанным символом из файла является

- 1. "a";
- 2. "b";
- ✓ 3. "c";
- 4. "d";
- 5. "e";
- 6. "f".

44. Из режима задачи возможен переход

- 1. в режим ядра и режим приостановки;
- ✓ 2. в режим ядра;
- 3. в режим ядра и режим готовности;
- 4. в любой другой режим.

45. Интерпретатор команд shell

- 1. является процессом, выполняющимся в режиме ядра;
- 2. для выполнения любой команды создает новый процесс;
- 3. осуществляет ввод командной строки, не пользуясь услугами ядра;
- 4. не пользуется системными вызовами;
- ✓ 5. является процессом, выполняющимся в режиме задачи.

46. Связывание обычных файлов системным вызовом link() может быть выполнено

- 1. только процессом, созданным интерпретатором shell;
- ✓ 2. любым процессом;
- 3. только процессом, принадлежащим суперпользователю;
- 4. только процессом, созданным интерпретатором shell, либо ядром.

47. Связывание каталогов файловой системы системным вызовом link() может быть выполнено

- 1. только процессом, созданным интерпретатором shell;
- 2. любым процессом;
- ✓ 3. только процессом, принадлежащим суперпользователю;
- 4. только процессом, созданным интерпретатором shell, либо ядром.

48. Компонента ОС "demand paging"

1. выполняет замещение страницы, путём выгрузки её во внешнюю память по некоторому алгоритму;

- 2. предотвращает возникновение ситуации пробуксовки (trashing);
- 3. занимается свопингом процессов;
- ✓ 4. выделяет страницу ОП, перемещая в нее копию страницы из внешней памяти;
- 5. выполняет откачку страниц во внешнюю память;
- 6. оперирует с понятием рабочего набора.

49. Выполнение P-операции P(S) над классическим семафором

- 1. ведет к уменьшению значения аргумента на 1;
- 2. равносильно операции $S=S+1$;
- ✓ 3. неделимая операция, уменьшающая положительное значение аргумента на 1;
- 4. ведет к увеличению значения аргумента на 1;
- 5. равносильно операции $S=S-1$;

- 6. производится над любым целочисленным аргументом;
- 7. подразумевает нечто иное.;
- 50. Выполнение V-операции V(S) над классическим семафором
 - 1. ведет к уменьшению значения аргумента на 1;
 - 2. равносильно операции $S=S+1$;
 - 3. ведет к увеличению значения аргумента на 1 для любого целочисленного аргумента;
 - 4. равносильно операции $S=S-1$;
 - 5. производится над любым целочисленным аргументом;
 - ✓ 6. неделимая операция, увеличивающая неотрицательное значение аргумента на 1.
- 51. Массовые операции над семафорами в UNIX (набор семафоров) введены с целью
 - 1. расширения понятия классического семафора;
 - 2. увеличения числа выполняемых операций над семафором;
 - ✓ 3. уменьшить вероятность возникновения тупиковых ситуаций;
 - 4. увеличения числа процессов, одновременно использующих семафоры.
- 52. Механизм очередей сообщений служит для обмена сообщениями
 - 1. родственных процессов;
 - 2. процессов, не связанных отношением родства;
 - 3. процессов, имеющих общего предка;
 - ✓ 4. любых процессов;
 - 5. процессов, имеющих общего владельца.
- 53. Системный вызов `msgget()` набора системных средств IPC позволяет
 - 1. получить сообщение из очереди сообщений;
 - 2. послать сообщение в очередь сообщений;
 - 3. образовать новую очередь сообщений;
 - 4. получить дескриптор существующей очереди;
 - ✓ 5. образовать новую очередь сообщений и получить дескриптор существующей очереди.
- 54. Системный вызов `shmget()` набора системных средств IPC позволяет
 - 1. подключить сегмент разделяемой памяти к виртуальной памяти процесса;
 - 2. отключить сегмент разделяемой памяти от виртуальной памяти процесса;
 - 3. образовать новый сегмент разделяемой памяти;
 - 4. найти сегмент разделяемой памяти по ключу;
 - ✓ 5. образовать новый сегмент разделяемой памяти или найти сегмент разделяемой памяти по ключу.
- 55. Набор программных средств IPC является средством взаимодействия
 - 1. родственных процессов;
 - 2. процессов, не связанных отношением родства;
 - 3. процессов, имеющих общего предка;
 - ✓ 4. любых процессов;
 - 5. процессов, имеющих общего владельца.
- 56. При страничной организации памяти
 - 1. виртуальная память процесса не может превышать размера оперативной памяти;
 - 2. каждый блок виртуальной памяти может иметь произвольный размер;
 - 3. имеет место двухуровневая трансляция виртуального адреса в физический;
 - ✓ 4. каждый блок виртуальной памяти имеет одинаковый размер;
 - 5. сумма виртуальных пространств всех процессов не может превышать размера оперативной памяти.
- 57. При сегментной организации памяти
 - 1. виртуальная память процесса не может превышать размера оперативной памяти;
 - ✓ 2. каждый блок виртуальной памяти может иметь произвольный размер;
 - 3. имеет место двухуровневая трансляция виртуального адреса в физический;
 - 4. каждый блок виртуальной памяти имеет одинаковый размер;

5. сумма виртуальных пространств всех процессов не может превышать размера оперативной памяти.

58. При сегментно-страничной организации памяти

1. виртуальная память процесса не может превышать размера оперативной памяти;
2. каждый блок виртуальной памяти может иметь произвольный размер;
- ✓ 3. имеет место двухуровневая трансляция виртуального адреса в физический;
4. каждый блок виртуальной памяти имеет одинаковый размер;
5. сумма виртуальных пространств всех процессов не может превышать размера оперативной памяти.

59. Произвольный алгоритм подкачки

- ✓ 1. выполняет замещение страницы, путём выгрузки её во внешнюю память по некоторому алгоритму;
2. предотвращает возникновение ситуации пробуксовки (trashing);
3. занимается свопингом процессов;
4. выделяет страницу ОП, перемещая в нее копию страницы из внешней памяти;
5. выполняет откачку страниц во внешнюю память;
6. оперирует с понятием рабочего набора.

60. Принцип локальности ссылок

1. выполняет замещение страницы, путём выгрузки её во внешнюю память по некоторому алгоритму;
2. предотвращает возникновение ситуации пробуксовки (trashing);
3. занимается свопингом процессов;
4. выделяет страницу ОП, перемещая в нее копию страницы из внешней памяти;
5. выполняет откачку страниц во внешнюю память;
- ✓ 6. оперирует с понятием рабочего набора.

61. Системный процесс - stealer

1. выполняет замещение страницы, путём выгрузки её во внешнюю память по некоторому алгоритму;
2. предотвращает возникновение ситуации пробуксовки (trashing);
3. занимается свопингом процессов;
4. выделяет страницу ОП, перемещая в нее копию страницы из внешней памяти;
- ✓ 5. выполняет откачку страниц во внешнюю память;
6. оперирует с понятием рабочего набора.

62. ОС UNIX - является мобильной ОС, поскольку

1. позволяет легко работать в сети;
2. обладает средствами восстановления после возникновения сбоев в системе;
- ✓ 3. допускает перенос в текстах на различные платформы;
4. построена по архитектуре "Клиент-сервер".

63. Критическая секция

- ✓ 1. предотвращает использование несколькими процессами критичных данных;
2. обеспечивает целостность данных пользователя;
3. создается в процессе обработки прерываний;
4. служит для предотвращения использования несколькими пользователями критичных данных;
5. доступна только процессам, созданным ОС.

64. Процесс - это

- ✓ 1. единица работы, управления и потребления ресурсов;
2. последовательность команд программы;
3. объект, созданный интерпретатором команд;
4. объект, созданный процессом ядра;
5. нечто иное.

65. Процесс обязательно включает

1. секции текста, стека, данных;
- ✓ 2. секции текста, стека;
3. секцию текста;
4. секции текста, данных;
5. секции стека, данных.

66. Ядро ОС UNIX

1. является полностью машинно-независимой частью ОС;
2. включает секцию управляющих структур, программную секцию и технические средства;
3. непосредственно взаимодействует с программами пользователя;
- v 4. выполняет диспетчерские функции;
5. является самым нижним уровнем в архитектуре ОС.

67. В режиме ядра

1. выполняются только процессы, созданные ОС;
- v 2. выполняется код ядра ОС;
3. процесс не может быть прерван;
4. недоступен аппарат системных вызовов;
5. нечто иное.

68. Промежуточная таблица областей процессов

- v 1. обеспечивает совместное использование областей независимыми процессами;
2. обеспечивает ссылки к таблице процессов;
3. содержит управляющую информацию о состоянии процесса;
4. указывает, где размещены сегменты текста, стека и данных.

69. Таблица областей процессов

1. обеспечивает совместное использование областей независимыми процессами;
2. обеспечивает ссылки к таблице процессов;
3. содержит управляющую информацию о состоянии процесса;
- v 4. указывает, где размещены сегменты текста, стека и данных.

70. Таблица процессов

1. обеспечивает совместное использование областей независимыми процессами;
2. обеспечивает ссылки к таблице процессов;
- v 3. содержит управляющую информацию о состоянии процесса;
4. указывает, где размещены сегменты текста, стека и данных.

71. Реентерабельная программа

1. оптимальна по времени выполнения;
- v 2. допускает совместное свое использование;
3. допускает совместное свое использование в системной фазе;
4. допускает совместное свое использование в пользовательской фазе.

72. При нехватке ОП кандидатом на выгрузку является

1. процесс, находящийся в пользовательской фазе;
2. процесс, запущенный в режиме ядра;
- v 3. процесс, находящийся в системной фазе;
4. процесс, запущенный в режиме пользователя.

73. Приоритет процесса является

- v 1. функцией от времени с момента последней загрузки в ОП;
2. функцией от времени с момента предоставления процессора;
3. функцией от времени использования процессора;
4. функцией от времени нахождения в системной фазе;
5. функцией от времени нахождения в пользовательской фазе.

74. Свопингу

1. менее подвергаются процессы с большим приоритетом;
2. более подвержены процессы, находящиеся в системной фазе;
- v 3. не подвергаются процессы, созданные в режиме ядра;
4. более подвержены процессы, находящиеся в пользовательской фазе.

75. Таблица файлов содержит

1. сведения о типе файла, правах доступа к нему, размере файла, а также счетчик ссылок на запись таблицы;
- v 2. информацию о режиме открытия файла, указатель чтения/записи и число ссылок на запись таблицы;
3. идентификатор (дескриптор) файла;
4. номера блоков, составляющих файл;
5. номер процесса.

76. Таблица описателей файла содержит

✓ 1. сведения о типе файла, правах доступа к нему, размере файла, а также счетчик ссылок на запись таблицы;

2. информацию о режиме открытия файла, указатель чтения/записи и число ссылок на запись таблицы;

3. идентификатор (дескриптор) файла;

4. номера блоков, составляющих файл;

5. номер процесса.

77. Таблица открытых файлов процесса содержит

1. сведения о типе файла, правах доступа к нему, размере файла, а также счетчик ссылок на запись таблицы;

2. информацию о режиме открытия файла, указатель чтения/записи и число ссылок на запись таблицы;

✓ 3. идентификатор (дескриптор) файла;

4. номера блоков, составляющих файл;

5. номер процесса.

78. Кэш-память

1. ускоряет работу с байт-ориентированными устройствами;

✓ 2. ускоряет работу с блок-ориентированными устройствами;

3. ускоряет работу с любыми устройствами;

4. является аппаратно реализованным механизмом.

79. Подсистема управления процессами

1. является частично машинно-зависимой;

2. распознает системные вызовы `fork()`, `exit()` и пр.;

✓ 3. функционирует на уровне ядра;

4. функционирует на уровне аппаратуры.

80. Подсистема управления файлами

1. является частично машинно-зависимой;

2. распознает системные вызовы `read()`, `write()` и пр.;

✓ 3. функционирует на уровне ядра;

4. функционирует на уровне аппаратуры.

81. После открытия первым процессом файлов `file1`, `file2`, `file3`, `file4`, вторым процессом - файлов `file1`, `file2`, третьим - файла `file2` и последующим открытием вторым процессом файла `file3`, максимальное значение счетчика в таблице описателей файлов равно

1. 1;

✓ 2. 3;

3. 2;

4. 4;

5. 5;

6. некоторому другому значению.

82. После открытия процессом файла и создания канала число записей в таблице открытых файлов процесса равно

1. 7;

✓ 2. 6;

3. 4;

4. 3;

5. некоторому другому значению.

83. Образ процесса состоит из

1. процедурного сегмента, сегмента данных и сегмента стека;

2. процедурного сегмента и сегмента стека;

✓ 3. процедурного сегмента и сегмента данных;

4. процедурного сегмента, сегмента данных и сегмента стека и U-области;

5. процедурного сегмента, сегмента стека и U-области.

84. Исходный файл содержит последовательно по 128 значений "a",

"b", "c", "d", "e", "f" и т. д. После открытия файла и получения копии дескриптора файла по системному вызову dup() с использованием оригинального дескриптора файла выполнено чтение двух записей по 64 байта, а с использованием копии дескриптора файла выполнено чтение 3-х записей по 64 байт. Последним прочитанным из файла символом является

1. "d";
- ✓ 2. "c";
3. "b";
4. "a";
5. "f";
6. "e".

85. После открытия процессором 2-х файлов и создания потомка по системному вызову fork() общее число записей в таблице файлов равно

- ✓ 1. 2;
2. 5;
3. 8;
4. 10;
5. некоторому другому значению.

86. Контекст процесса - это

1. адресное пространство процесса;
- ✓ 2. состояние процесса в любой момент времени;
3. образ процесса в любой момент времени;
4. процедурный сегмент, сегмент данных и сегмент стека.

87. Смена контекста выполняется

1. при переходе из режима ядра в режим задачи;
2. при переходе из режима ядра в заблокированное состояние;
3. при переходе из режима задачи в режим ядра;
4. при переходе из заблокированного состояния в режим готовности;
- ✓ 5. при любой смене режима.

88. При вызове из функции main() функции printf() в момент вывода данных число записей активации равно

1. 0;
2. 1;
3. 2;
- ✓ 4. 3;
5. другому значению.

89. Обработка сигналов выполняется

1. при переходе из режима задачи в режим ядра;
- ✓ 2. при переходе из режима ядра в режим задачи;
3. при переходе в режим приостанова;
4. при выходе из режима приостанова;
5. при переходе из режима ядра в режим задачи и обратно.

90. Суперблок помимо прочего содержит

1. указатели на описатели файлов файловой системы;
- ✓ 2. список свободных описателей файлов;
3. указатель на таблицу описателей файлов;
4. списковую структуру с номерами описателей файлов;
5. счетчик числа используемых описателей файлов.

91. Два родственника процесса, выполняющие ввод-вывод данных, используют системный вызов pipe() для обмена данными между собой. Первая программа пишет в канал, вторая - читает.

В момент, когда первая программа поместила в канал n записей длиной 128 байт, вторая программа может обнаружить в канале

1. $128 \cdot n$ байт;

- 2. $128 * n + 2$ байта;
- 3. $128 * n - 2$ байта;
- 4. $128 * n + 1$ байт;
- ✓ 5. любое число байт.

92. Исходный файл содержит последовательно 128 байт значений "a", "b", "c", "d", "e", "f" и т.д. Программа создает новый процесс и в рамках порожденного процесса дважды открывает указанный файл и читает с использованием первого дескриптора две записи по 64 байта, а затем с использованием второго дескриптора три записи по 64 байта. Последним прочитанным символом из файла является

- 1. "f";
- 2. "e";
- 3. "d";
- 4. "c";
- ✓ 5. "b";
- 6. "a".

93. Результатом нормального выполнения системного вызова wait() является

- 1. нулевой код завершения;
- ✓ 2. идентификатор завершившегося процесса;
- 3. идентификатор ожидаемого процесса;
- 4. статус завершения.

94. Проверка поступления сигналов выполняется

- 1. при переходе из режима задачи в режим ядра;
- ✓ 2. при переходе в режим приостанова, режим ядра и возврата из режима готовности;
- 3. при переходе из режима ядра в режим задачи;
- 4. при переходе из режима заблокировано в режим готовности.

95. Сразу после обработки прерывания режимом процесса является

- ✓ 1. режим ядра;
- 2. режим задачи;
- 3. режим ядра или режим задачи;
- 4. режим готовности.

96. В случае нулевого второго аргумента системного вызова Signal

- 1. процесс игнорирует все последующие получения сигнала;
- ✓ 2. по получению сигнала процесс завершается;
- 3. сигнал посылается всем процессам, входящим с данным процессом в одну группу;
- 4. сигнал посылается всем процессам, у которых код иден-ра пользователя совпадает с тем, под которым выполняется процесс;
- 5. процесс немедленно завершается.

97. Чтение потока символов с терминала интерпретатором shell

- 1. выполняется отдельным процессом в режиме задачи;
- 2. выполняется процессом-интерпретатором в режиме ядра;
- 3. выполняется процессом-интерпретатором в режиме задачи;
- ✓ 4. выполняется отдельным процессом в режиме ядра.

98. Системные вызовы, связанные со временем

- 1. доступны только в привилегированном режиме;
- ✓ 2. оперируют с глобальными переменными, определенными на уровне ядра;
- 3. работают с системными часами;
- 4. влияют на режим квантования времени;
- 5. отсчитывают время в машинных тиках.

99. Системный вызов mount() обеспечивает

- 1. связывание файлов;
- ✓ 2. связывание файловых систем;
- 3. связывание каталогов;
- 4. связывание дисковых устройств.

100. Блокировка описателя файла в алгоритме Link
- v 1. порождает тупиковые ситуации;
 - 2. предотвращает тупиковые ситуации;
 - 3. создает условия для конкуренции процессов;
 - 4. ведет к возникновению некорректных ситуаций.

101. Удаление связи с файлом после его открытия
- 1. приводит к аварийному завершению процесса;
 - v 2. оставляет возможность для работы с ним;
 - 3. приводит к автоматическому закрытию файла;
 - 4. приводит к блокировке этого файла.

102. Процесс открывает существующий файл длиной 500 байт в режиме O_WRONLY и записывает в него 10 байт. Какова длина файла после окончания записи?

1. 510 б;

~~v 2. 10 б;~~

✓ 3. 500 б;

4. 0 б;

5. некоторое другое значение.

103. Процесс открывает существующий файл длиной 500 байт в режиме O_WRONLY|O_APPEND и записывает в него 10 байт. Какова длина файла после окончания записи?

v 1. 510 б;

2. 10 б;

3. 500 б;

4. 0 б;

5. некоторое другое значение.

104. Процесс открывает существующий файл длиной 500 байт в режиме O_WRONLY и записывает в него 10 байт. После этого указанная операция записи 10 байт повторяется еще 50 раз. Какова длина файла после окончания записи?

v 1. 510 б;

2. 10 б;

3. 500 б;

4. 0 б;

5. некоторое другое значение.

105. При условии, что fildes - дескриптор файла, системный вызов lseek(fildes, (off_t)0, SEEK_END) позволит

1. выполнить переход в начало файла;

v 2. определить длину файла;

3. уменьшить длину файла до нуля;

4. перевести указатель в начало файла.

106. Системный вызов unlink("/temp/usedfile"), выполненный сразу после создания файла

1. добавляет ссылку к указанному файлу;

2. закрывает файл;

v 3. уничтожает файл;

4. запрещает доступ к файлу.

107. Каковы права доступа, при которых владелец может выполнять все операции (r, w, x), а все прочие пользователи - только читать?

1. 0666;

2. 0644;

3. 0555;

v 4. 0744.

108. Каковы права доступа, при которых владелец может читать и писать в файл, а все прочие пользователи - только читать?

1. 0666;
- v 2. 0644;
3. 0555;
4. 0744.

109. Сколько строк будет напечатано при выполнении программы, содержащей следующий контекст?

```
printf("One\n");  
fork();  
printf("Two\n");
```

1. 1;
2. 2;
- v 3. 3;
4. 4;

5. некоторое другое число.

110. Статус завершения процесса, переданный из процесса-потомка в родительский процесс, доступен

1. в родительском процессе в любом случае;
- v 2. только в случае, если в родительском процессе выдан системный вызов wait();
3. только в случае успешного завершения потомка;
4. только в случае аварийного завершения потомка.

111. Процесс последовательно создает три процесса-потомка, после чего ожидает их завершения тремя системными вызовами wait(). Завершающийся процесс в качестве статуса завершения возвращает свой идентификатор. С большей вероятностью первым будет получен

1. идентификатор первого процесса;
2. идентификатор второго процесса;
3. идентификатор третьего процесса;
- v 4. идентификатор любого из трех процессов.

112. Системный вызов alarm()

1. моментально посылает сигнал некоторому процессу;
2. моментально посылает сигнал вызвавшему его процессу;
- v 3. устанавливает интервал времени, через который данному процессу будет послан сигнал;
4. устанавливает интервал времени, через который некоторому процессу будет послан сигнал;
5. выполняет нечто иное.

113. Системный вызов raise()

- v 1. моментально посылает сигнал некоторому процессу;
2. моментально посылает сигнал вызвавшему его процессу;
3. устанавливает интервал времени, через который данному процессу будет послан сигнал;
4. устанавливает интервал времени, через который некоторому процессу будет послан сигнал;
5. выполняет нечто иное.

114. Именованные каналы

1. не работают по алгоритму FIFO;
- v 2. могут использоваться неродственными процессами;
3. уничтожаются после работы с ними;
4. полностью аналогичны файлам файловой системы.

115. Ключ объекта IPC

1. является уникальным в рамках программы пользователя;
2. является уникальным в рамках группы процессов, работающих с объектом;
- v 3. является уникальным в рамках вычислительной системы (ОС);
4. является уникальным в рамках вычислительной сети.

116. Номер семафоров (индекс) в наборе семафоров

1. должен быть меньше или равен числу семафоров;
- v 2. должен быть меньше числа семафоров;
3. должен быть положительным значением;
4. может быть больше числа семафоров.

117. В случае заполненности очереди сообщений и невозможности поместить в нее сообщение процесс, выдавший системный вызов `msgsnd()`,

1. заканчивается аварийно с соответствующим кодом завершения;
- ✓ 2. замораживается до появления возможности занести сообщение;
3. заносит сообщение в буфер и продолжает выполнение;
4. игнорирует занесение сообщения в очередь и продолжает выполнение.

118. По технологии "Клиент-сервер" не работает

1. сервер баз данных;
2. WWW-сервер;
- ✓ 3. файловый сервер;
4. сервер приложений;
5. почтовый сервер.

119. При использовании модели дейтаграмм в сравнении с моделью TCP-соединения не используется следующий этап серверного процесса

1. создание сокета (`socket`);
2. связывание адреса сервера с сокетом (`bind`);
- ✓ 3. установка соединения с клиентом (`accept`);
4. прием данных от клиента;
5. передача данных клиенту.

120. При использовании модели дейтаграмм в сравнении с моделью TCP-соединения не используется следующий этап клиентского процесса

1. преобразование и запись IP-адреса в структуру сокета (`inet_addr`);
2. создание сокета (`socket`);
- ✓ 3. подключение к сокету (`connect`);
4. прием данных от клиента;
5. передача данных клиенту.

121. Родительский процесс создал неименованный канал, получив дескрипторы `f[0]`, `f[1]`, после чего породил дочерний процесс. Оба процесса используют канал для двухсторонней передачи данных, при этом каждый из процессов использует дескриптор `f[0]` для чтения, а `f[1]` - для записи. Описанная схема

1. обеспечивает гарантированную двухстороннюю передачу данных;
- ✓ 2. дает непредсказуемый результат, включая возможные дедлоки;
3. соответствует спецификации системного вызова `pipe()`;
4. аналогична схеме использования именованного канала;

122. Родительский процесс создал неименованный канал, получив дескрипторы `f[0]`, `f[1]`, после чего породил дочерний процесс. Оба процесса используют канал для двухсторонней передачи данных, при этом и для чтения и для записи родительский процесс использует дескриптор `f[0]`, а порожденный процесс - дескриптор `f[1]`. Описанная схема

1. соответствует спецификации системного вызова `pipe()`;
2. дает непредсказуемый результат, включая возможные дедлоки;
- ✓ 3. обеспечивает гарантированную

двухстороннюю передачу данных

4. аналогична схеме использования
именованного канала;

123. Можно ли открыть именованный канал на чтение и на запись?

☒ 1. да;

2. нет;

3. в зависимости от ситуации;

124. RPC расшифровывается как

1. механизмы межпроцессного взаимодействия;

2. механизмы взаимодействия посредством сокетов;

☒ 3. удаленный вызов процедур;

4. принцип локальности ссылок;

125. IPC расшифровывается как

1. удаленный вызов процедур;

2. механизмы взаимодействия посредством сокетов;

3. принцип локальности ссылок;

☒ 4. механизмы межпроцессного взаимодействия;

126. Фрагмент программы:

```
...  
if(fork()==0)  
printf("text\n");
```

...
Каким процессом будет напечатан текст?

1. родителем;

☒ 2. потомком;

3. родителем и потомком;

127. Фрагмент программы:

```
...  
if(fork())  
printf("text\n");  
...
```

Каким процессом будет напечатан текст?

☒ 1. родителем;

2. потомком;

3. родителем и потомком;

128. Фрагмент программы:

```
...  
if((i=fork())==0)  
printf("text\n"  
);
```

...
Каким процессом будет напечатан текст?

- 1. родителем;
- ☒ 2. потомком;
- 3. родителем и потомком;

129. Фрагмент программы:

```
...
if(i=fork()==0)
printf("text\n");
```

...
Каким процессом будет напечатан текст?

- 1. родителем;
- ☒ 2. потомком;
- 3. родителем и потомком;

130. Фрагмент программы:

```
...
if(fork()!=0)
printf("text\n");
```

...
Каким процессом будет напечатан текст?

- ☒ 1. родителем;
- 2. потомком;
- 3. родителем и потомком;

131. Фрагмент программы:

```
...
if(i=fork()!=0)
printf("text\n");
```

...
Каким процессом будет напечатан текст?

- ☒ 1. родителем;
- 2. потомком;
- 3. родителем и потомком;

132. marshaling - это

- 1. процедура пересылки данных между клиентом и сервером;
- ☒ 2. процедура упаковки данных в сетевое сообщение;
- 3. процедура пробуксовки;
- 4. процедура распаковки данных из сетевого сообщения;

133. клиентский суррогат (client stub)

- 1. передает параметры реальной функции сервера;
- ☒ 2. упаковывает данные в сетевое сообщение;
- 3. распаковывает полученные от клиентского процесса данные;
- 4. подготавливает файл спецификаций RPC;

134. серверный суррогат (server stub)

1. подготавливает файл спецификаций RPC;
- ✓ 2. распаковывает полученные от клиентского процесса данные;
3. упаковывает данные в сетевое сообщение и передает его стабу сервера;
- ✓ 4. передает параметры реальной функции сервера; S.D.

135. Файл спецификаций RPC

1. подготавливается клиентским суррогатом;
2. подготавливается серверным суррогатом;
- ✓ 3. подготавливается пользователем;
4. генерируется утилитой rpcgen;

136. IP-адрес

1. назначается производителем оборудования и является уникальным адресом;
- ✓ 2. состоит из четырех байт и используется на сетевом уровне;
3. назначается администратором и состоит из нескольких частей, разделенных точками;
4. состоит из четырех байт и используется на физическом уровне;

137. Ethernet-адрес

1. состоит из четырех байт и используется на физическом уровне;
2. назначается администратором и состоит из нескольких частей, разделенных точками;
3. состоит из четырех байт и используется на сетевом уровне;
- ✓ 4. назначается производителем оборудования и является уникальным адресом;

138. Символьный адрес

1. состоит из четырех байт и используется на сетевом уровне;
- ✓ 2. назначается администратором и используется на прикладном уровне;
3. состоит из четырех байт и используется на физическом уровне;

4. назначается администратором и используется на сетевом уровне;

139. 112.255.255.112 - IP-адрес

- ✓ 1. класса A;
- 2. класса B;
- 3. класса C;
- 4. класса D;

140. 144.255.255.144 - IP-адрес

- 1. класса A;
- ✓ 2. класса B;
- 3. класса C;
- 4. класса D;

141. 193.155.55.65 - IP-адрес

- 1. класса A;
- 2. класса B;
- ✓ 3. класса C;
- 4. класса D;

142. Максимальное число хостов в классе A

- ✓ 1. 16777214; $2^{24} = 16777216$
- 2. 65534;
- 3. 254;

143. Максимальное число хостов в классе B

- 1. 16777214;
- 2. 254;
- ✓ 3. 65534; $2^{16} = 65536$

144. Максимальное число хостов в классе C

- ✓ 1. 254; $2^8 = 256$
- 2. 65534;
- 3. 16777214;

144. Ядро состоит из

- 1. взаимосвязанных системных таблиц;
- 2. раздела управляющих структур и машинно-независимой частью программной секции;
- 3. раздела управляющих структур;
- 4. раздела управляющих структур и взаимосвязанных системных таблиц;
- ✓ 5. раздела управляющих структур и программной секции;

145. Обслуживанием программ пользователей занимается

- ✓ 1. ядро; ✓
- ✓ 2. утилиты; D.O.
- 3. пользовательский процесс;
- 4. технические средства ОС;
- 5. ничего из перечисленного

неверно;

145. Процессу, выполняющемуся в системной фазе соответствует

	Класс N°	Класс N
A	1.0.0.0	126.0.0.0
B	128.0.0.0	191.255.0.0
C	192.0.1.0	223.255.255.
D	224.0.0.0	239.255.255
E	240.0.0.0	247.255.255

1. исполняемый код пользовательской программы;
- ✓ 2. реентеральный код ядра;
3. один или несколько процессов;
4. ничего из перечисленного

неверно;

146. Процессу, выполняющемуся в пользовательской фазе соответствует

- ✓ 1. исполняемый код пользовательской программы;
2. реентеральный код ядра;
3. один или несколько процессов;
4. ничего из перечисленного

неверно;

146. Для синхронизации процессов в системной фазе используется

1. аппарат прерываний;
- ✓ 2. аппарат семафоров (аппарат событий);
3. аппарат блокировки;
4. аппарат

сохранения/восстановление контекста пользователя;

5. ничего из перечисленного

неверно;

147. Обработку прерываний от внешних устройств регламентирует

- ✓ 1. аппарат прерываний;
2. аппарат семафоров (событий);
3. аппарат блокировки;
4. аппарат

сохранения/восстановление контекста пользователя;

5. ничего из перечисленного

неверно;

147. Какой из процессов диспетчерский процесс, занимающийся свопингом, выгрузит из ОП в первую очередь при нехватке места для загрузки образа процесса, находящегося во внешней памяти

1. процесс, имеющий самый меньший приоритет среди всех;
2. процесс, дольше всех находящийся в ОП;
3. процесс, занимающий в ОП наибольшее место;

4. процесс, находящийся в системной фазе и дольше всех находящийся в ОП;

5. процесс, находящийся в системной фазе;

- ✓ 6. процесс, находящийся в системной фазе и занимающий в ОП наибольшее место;

148. Машинно-зависимой является следующая подсистема ОС

UNIX

- 1. подсистема управления файлами;
- ~~2. подсистема управления процессами;~~
- ✓ 3. подсистема управления устройствами;
- 4. все из перечисленных подсистем являются машинно-независимыми;

148. Механизм квантования времени реализует

- 1. подсистема управления файлами;
- ✓ 2. подсистема управления процессами;
- 3. подсистема управления устройствами;
- 4. ни одна из перечисленных подсистем;

149. ОП управляет

- 1. подсистема управления файлами;
- ✓ 2. подсистема управления процессами;
- 3. подсистема управления устройствами;
- 4. ни одна из перечисленных подсистем;

149. Что из перечисленного является библиотечной функцией

- ✓ 1. `Printf()`;
- 2. `Open()`;
- 3. `Creat()`;
- 4. `Read()`;
- 5. среди перечисленных нет ни одной библиотечной функции;

150. Какая из перечисленных функций возвращает дескриптор файла

- 1. `Read()`;
- 2. `Lseek()`;
- ~~✓ 3. `Chmod()`;~~
- ✓ 4. `Pipe()`;

150. Если первый аргумент системного вызова `kill()` меньше нуля

- и не равен -1, то
- 1. ядро посылает сигнал процессу с идентификатором, равным модулю аргумента;
- 2. ядро посылает сигнал всем процессам, входящим в одну группу с этим процессом;
- ✓ 3. ядро посылает сигнал всем процессам, идентификатор группы которых равен этому модулю этого аргумента;
- 4. ничего из предложенного

неверно;

- 151 В семафор не входит
1. значение семафора;
 2. идентификатор процесса, который хронологически последним работал с семафором;
 3. число процессов, ожидающих увеличения семафора;
 4. число процессов, ожидающих нулевого значения семафора;
 - ✓ 5. число процессов, ожидающих уменьшения семафора;

- 1) Структурный механизм работы с памятью истр-ся
для - управления графиком режима вывода и кер-ии
- 2) Структура, в частности, содержит - информацию о
кол-ве выданных блоков файловой сист.
- 3) Системный вызов exec() служит для - загрузки
пр-мы из любого процесса
- 4) Завершение процесса вып-ся в пр-ме - сист-ной функ-
ции exit() либо при завершении посл-го оп-ра главной
ф-ии main()
- 5) Сист-ной вызов kill() служит - для посылки сигнала
любому процессу или группе
- 6) Сист-ной вызов signal() служит - для задания режима
обработки сигнала.
- 7) Выполнение сист-но вызова - связано с переходом из
пользов-ой фазы в сист-ную
- 8) В ядре есть команда - kill - вып-е асинхронно при-

TCP - Тр-ур

FTP - пр-я