ВСЕ ВОПРОСЫ

1) Какие программные системы называют приложениями реального времени и особенности их функционирования?
2) Опишите состав и функционирование вычислительного ядра ПРВ.
3) Охарактеризуйте понятие «темпоральные данные».
4) Охарактеризуйте понятие "период репрезентативности" темпорального данного.
5) Темпоральные данные и их характеристики.
6) Охарактеризуйте понятия системного времени и реального времени ПРВ.
7) Охарактеризуйте понятие «одномоментность» в реальном времени ПРВ.
8) Опишите состав базы темпоральных данных вычислительного ядра ПРВ.
9) Опишите формальную модель темпоральных данных.
10) Какие существуют классы темпоральных данных и их семантика?
11) Охарактеризуйте темпоральные данные класса импульс.
12) Охарактеризуйте темпоральные данные класса датум.
13) Охарактеризуйте темпоральные данные класса мода.
14) Какие типы каналов предоставляет БТД для связи с периферийными подсистемами ПРВ.
15) Что называется темпоральным прецедентом БДТ и причины его возникновения?
16) Что понимается под режимами жёсткого и мягкого реального времени работы СРВ?
17) Формальный метод оценки валидности темпоральных данных в режиме мягкого реального времени.
18) Формальный метод оценки валидности темпоральных данных в режиме жёсткого реального времени.
19) Если на компьютере с именем comp выполнена команда shell:
#mount –T io-net /lib/dll/npm-qnet.so,
то в файловой системе компьютера comp …
20) Какой процесс будет считаться родительским при запуске процесса функцией system()?
21) Какие функции позволяют запустить процесс на удалённом узле?
22) Являются ли вызовы MsgReply(),MsgRead(),MsgWrite() блокирующими?
23) Какой вид имеет объявление функции, используемой для запуска нитей?
24) Может ли нить main() быть обособленной?
25) Может ли нить main() быть присоединяемой?
26) Какие параметры стека создаваемой нити можно задать одновременно?
27) Какие значения приоритета можно назначить нити в QNX?
28) Активная нить выполнила функцию освобождения процессора - sched_yield(), какая из готовых к выполнению нитей начнёт выполняться?
29) Что понимается под инверсией приоритетов нитей при выполнении ПРВ?
30) Какие функции связывают продолжение выполнения одной нити с моментом завершения существования другой нити?
31) Рекурсивный мутекс позволяет нити…
32) Если нити требуется исключительный доступ к некоторому ресурсу, но использовать ресурс нить может только при выполнении определённых условий, то каким из приведённых ниже механизмов синхронизации можно воспользоваться наиболее эффективно?
33) Какой объект управления синхронизацией нитей создаётся функцией:
int sem_init(sem_t *sem, int pshared, unsigned value) ?
34) Какая функция управления семафором приводит к увеличению значения счётчика семафора на 1?
35) Механизм отображения адресов системных областей памяти в адресное пространство процесса используется для доступа процесса к …
36) Если процессу приходит сигнал, а диспозиция сигнала явно не установлена, то как это отразится на процессе?

37) Если сигнал генерируется функцией kill(pid, sig), где pid>0, то кому адресуется сигнал?
38) Какие функции позволяют послать сигнал процессу, находящемуся на удалённом узле локальной сети?
39) Какие функции позволяют установить обработчик сигналов в режиме, который предусматривает возможность образования очереди сигналов?
40) Укажите имя пользователя, которое автоматически регистрируется при установке ОС QNX.
41) Клиент посылает 120 Кб данных серверу на удалённом узле. Какой объем данных будет получен сервером при выполнении функции MsgReceive()?
42) Родительский процесс, находящийся на узле с именем compA, запустил дочерний процесс на удалённом узле с именем compB и передал ему необходимые параметры. Какая последовательность действий дочернего процесса является верной для установления соединения с родительским процессом?
43) Какова особенность запуска дочернего процесса с помощью функции fork()?
44) Какова особенность запуска дочернего процесса с помощью функции vfork()?
45) В каком состоянии окажется процесс-сервер, выполнивший функцию MsgReceive(), если процесс-клиент был в Send-блокированном состоянии?
46) Что обеспечивает механизм векторов ввода/вывода при обмене сообщением между клиентом и сервером?
47) Могут ли в приложении быть запущены нити с разными дисциплинами диспетчеризации?
48) В каком состоянии окажется процесс-клиент, выполнивший функцию MsgSend(), если процесс-сервер был в Receive-блокированном состоянии?
49) Если для управления доступом нитей к некоторому разделяемому программному ресурсу создан мутекс, то что предпримет ядро по отношению к нити, которая попытается осуществить доступ к этому ресурсу, не захватывая мутекс?
50) Что произойдёт с нитью, если она по отношению к блокировке чтения/записи выполнит функцию pthread_rwlock_wrlock() или pthread_rwlock_rdlock(), а блокировка чтения/записи уже захвачена на чтение?
51) Что произойдёт с нитью, если она по отношению к блокировке чтения/записи выполнит функцию pthread_rwlock_wrlock() или pthread_rwlock_rdlock(), а блокировка чтения/записи уже захвачена на запись?
52) Какой объект синхронизации нитей создаётся функцией:
sem_t* sem_open(char* sem_name,int oflags,mode_t mode,unsigned value); ?
53) Как изменяется значение счётчика семафора в результате успешного выполнения функции sem_post()?
54) Как изменяется значение счётчика семафора в результате успешного выполнения функции sem_wait()?
55) Какой результат ожидает процесс при выполнении функции shm_open().
56) Какой результат ожидает процесс при выполнении функции mmap().
57) Если сигнал генерируется функцией kill(pid, sig), где pid=0, то кто является адресатом этого сигнала?
58) Если для посылки сигнала процесс использует функцию SignalKill(), то кто может быть адресатом сигнала?
59) Условия адекватного применения процессом функции SignalWaitinfo() для работы с сигналами.
60) Что необходимо выполнить удалённому процессу-клиенту с результатом выполнения функции nd=netmgr_strtond("/net/comp",NULL), используемого для организации соединения с процессом-сервером на удалённом узле с именем comp?
61) Какие и чьи права доступа будет иметь файл, созданный при выполнении функции open("file.dat", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR)?
62) Как завершится выполнение функции open("file.dat", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR); если в текущем каталоге существует файл с именем file.dat?
63) Что означает для нити, запущенной в процессе, свойство обособленности?

64) Если завершился квант процессорного времени текущей нити с приоритетом pr и дисциплиной диспетчеризации RR, но выполнение нити ещё не завершилось, то ожидающей нити с каким приоритетом будет предоставлен очередной квант процессорного времени?
65) Что означает механизм наследования приоритетов для борьбы с инверсией приоритетов?
66) Какой атрибут устанавливается процессом в атрибутной записи запускаемой нити, чтобы она была обособленной нитью?
67) Что произойдёт с нитью, если она по отношению к некоторой блокировке чтения/записи выполнит функцию pthread_rwlock_trywrlock() или pthread_rwlock_tryrdlock(), а блокировка чтения/записи уже захвачена?
68) Какой программный объект создаётся функцией:
int shm_open(const char *name, int oflag, mode_t mode)?
69) Выполнение какой функции приводит к уменьшению на 1 значения счётчика семафора?
70) Для чего процесс применяет функцию mmap()?
71) Если сигнал генерируется функцией kill(pid, sig), где pid<0, то кому адресуется сигнал?
72) Какая из функций kill() или SignalKill() позволяют послать сигнал, адресуемый конкретной нити?
73) Если пришёл сигнал, адресованный нити, которая замаскировала этот сигнал, то что будет с этим сигналом?
74) Если для управления доступом нитей к некоторому разделяемому программному ресурсу создан мутекс, что предпримет ядро по отношению к нити, которая попытается осуществить доступ к этому ресурсу, не захватывая мутекс?
75) Как в ОСРВ QNX6 задаётся абсолютное значение реального времени для таймеров?
76) Как в ОСРВ QNX6 задаётся значение интервала реального времени?
77) Какой тип уведомления для таймера формирует вызов SIGEV_PULSE_INIT()?
78) Какой тип уведомления формирует для таймера вызов SIGEV_SIGNAL_INIT()?
79) Какой тип уведомления формирует для таймера вызов SIGEV_SIGNAL_CODE_INIT()?
80) Какой тип уведомления таймера формирует вызов SIGEV_SIGNAL_THREAD_INIT()?
81) Какой тип уведомления таймера формирует вызов SIGEV_THREAD_INIT()?
82) Какой режим срабатывания таймера планируется вызовом timer_settime(), если поля переменной struct itimerspec timer равны:
timer.it_value.tv_sec=3600;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=0;
timer.it_interval.tv_nsec=0;
?
83) Как сработает таймер, запланированный вызовом timer_settime(), если поля переменной struct itimerspec timer равны:
timer.it_value.tv_sec=0;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=1;
timer.it_interval.tv_nsec=500000000;
?
84) Поля переменной struct itimerspec timer имеют значения:
timer.it_value.tv_sec=1;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=1;
timer.it_interval.tv_nsec=0.
Какой тип и режим срабатывания таймера имеет смысл с такими полями системного времени?
85) Какую роль играет функция Timer_Timeout()?

86) Какова реакция на тайм-аут ядра, установленного клиентом для REPLY блокированного состояния при передаче сообщения, если при создании канала сервером был установлен флаг _NTO_CHF_UNBLOCK?

ВОПРОСЫ

1. **Какие программные системы называют приложениями реального времени и особенности их функционирования?**

Приложения реального времени (ПРВ) являются особым классом программных приложений, которые обеспечивают работу систем реального времени. В самом общем виде функционирование ПРВ выражается в управлении потоками данных между ПРВ и его окружением в реальном времени.

Особенностью функционирования ПРВ является их непрерывный во времени обмен данными с параметрами физических объектов, с которыми они связаны специальными аппаратно-программными интерфейсами связи, обеспечивающими получение значений параметров физических объектов для контроля и формирования данных для управления их состоянием, а также устройствами регистрации, хранения, отображения или коммуникации данных

2. **Опишите состав и функционирование вычислительного ядра ПРВ.**

Принципиальная схема ПРВ представляет собой вычислительное ядро, взаимодействующее с периферийными подсистемами, обеспечивающими в реальном времени обмен данными ПРВ с его внешним окружением: физическим объектом, человеко-машинным интерфейсом оператора, базой данных СРВ и сетью передачи данных. Таким образом, в общем виде любая СРВ может быть представлена в виде взаимодействующих между собой ПРВ и его внешнего окружения, а ПРВ, в свою очередь, представляется в виде вычислительного ядра, взаимодействующего с его периферийными подсистемами посредством программных каналов, обеспечивающих транспонирование данных между внешним окружением ПРВ и базой темпоральных данных вычислительного ядра.

В состав вычислительного ядра ПРВ входят следующие компоненты:

- База темпоральных данных (БТД) – конечное множество темпоральных данных (программных объектов абстрактных типов), значения которых актуализируются в реальном времени.
- Система управления базой темпоральных данных (СУБТД) – служба, управляющая инициацией и параллельным выполнением заданного конечного набора вычислительных процедур, обеспечивающих обмен данными с периферийными подсистемами или актуализацию темпоральных данных БТД.
- Часы ПРВ – служба хронометрирования течения относительного реального времени в заданных тиках, используемая СУБТД для контроля валидности и управления актуализацией темпоральных данных БТД.
- Вычислительные процедуры – конечный набор параллельно выполняемых программных процедур, используемых СУБТД либо для взаимодействия с периферийными подсистемами (процедуры коммуникации данных), либо для обновления значений темпоральных данных (процедуры темпоральных вычислений).

Периферийные подсистемы, взаимодействующие одновременно с вычислительным ядром и внешним окружением ПРВ, выполняют следующие функции:

- Подсистема обмена данными с физическим объектом обеспечивает в обе стороны транспонирование значений между параметрами физического объекта СРВ и соответствующими темпоральными данными БТД.

- Подсистема обмена данными с оператором отображает текущие значения заданных темпоральных данных на терминал оператора СРВ или транспонирует вводимые оператором посредством человеко-машинного интерфейса значения в соответствующие темпоральные данные БТД;
- Подсистема регистрации и хранения данных сохраняет тренды заданных темпоральных данных БТД в базе данных СРВ (на внешних устройствах хранения информации);
- Подсистема сетевой репликации базы темпоральных данных осуществляет сетевую репликацию темпоральных данных фрагментов БТД распределенного ПРВ.

Транспонирование данных – конвертация (преобразование) данных из аналоговой формы представления на физическом объекте в программный тип данных или наоборот.

Тренд – временной ряд значений, характеризующий хронологию изменения технологического параметра объекта мониторинга (например, давление, температура, обороты, объём, расход, контролируемое событие), формирующийся в результате периодической актуализации значения параметра в реальном времени.

3. **Охарактеризуйте понятие «темпоральные данные».**

Темпоральные данные — это данные, актуальность которых ограничена во времени и которые требуют периодического обновления для поддержания их актуальности. Примером могут служить медицинские показатели, которые сохраняют свою актуальность на протяжении определенного временного промежутка и требуют обновления после его завершения

4. **Охарактеризуйте понятие "период репрезентативности" темпорального данного.**

Период репрезентативности — это промежуток времени, в течение которого значение темпорального данного считается актуальным. Начинается он с момента получения данных и продолжается до окончания этого периода. Протяженность периода репрезентативности называется интервалом репрезентативности

Валидность — показатель актуальности или допустимости использования полученного ранее значения темпорального данного в темпоральных вычислениях в текущий момент времени

5. **Темпоральные данные и их характеристики.**

Темпоральные данные характеризуются ограниченной во времени актуальностью и необходимостью их обновления для поддержания актуальности. Они играют ключевую роль в разработке приложений реального времени, где помимо текущего значения важны период репрезентативности и валидность данных

Период репрезентативности — это промежуток времени, в течение которого значение темпорального данного считается актуальным. Начинается он с момента получения данных и продолжается до окончания этого периода. Протяженность периода репрезентативности называется интервалом репрезентативности

Валидность — показатель актуальности или допустимости использования полученного ранее значения темпорального данного в темпоральных вычислениях в текущий момент времени.

6. **Охарактеризуйте понятия системного времени и реального времени ПРВ.**

Тик - единица шкалы времени - интервал времени, используемый для измерения и отсчета временных промежутков (заданная временнАя протяжённость, по истечении которой показание скачком возрастает на 1).

Реальное время – время (дискретное), в котором показание времени, начиная с нуля, периодически скачком возрастает на 1 по истечении тика.

Системное время - время, реализуемое службой времени ОСРВ и используемое ПРВ для задания тика времени собственных часов (аналог непрерывного, абстрактного времени).

$$\tilde{t} = \left\lfloor \frac{t}{1_t} \right\rfloor.$$

, где t~ - часы ПРВ, t - часы системного времени.

Темпоральный прецедент потери данного – ситуация, когда данные (например, информация или результаты вычислений), связанные с определенным моментом времени, утрачивают свою актуальность в текущий момент времени.

7. **Охарактеризуйте понятие «одномоментность» в реальном времени ПРВ.**

Одномоментность – фундаментальное понятие, которое означает способность ПРВ актуализировать (обновлять) БТД в течение тика.

Актуализация темпоральных данных в ПРВ не может быть осуществлена мгновенно, поэтому все события и данные, полученные в течение текущего тика считаются одномоментными, и им присваивается одна и та же метка времени, полученная от часов ПРВ.

Протяжённость тика в системном времени существенно влияет на способность системы к одномоментной актуализации данных и предотвращению темпоральных прецедентов.

Темпоральный прецедент потери данного – ситуация, когда данные (например, информация или результаты вычислений), связанные с определенным моментом времени, утрачивают свою актуальность в текущий момент времени.

8. Опишите состав базы темпоральных данных вычислительного ядра ПРВ.

База темпоральных данных представляется множеством объектов хранения данных (темпоральные данные) и множеством объектов коммуникации данных (входные/выходные каналы).

Темпоральные данные - множество объектов хранения данных, которые представляют собой информацию, используемую вычислительными процедурами как входные или выходные данные.

Темпоральные данные могут быть как экзогенными (первичными, канальными), получаемыми от периферийных подсистем, так и эндогенными (вторичными, вычисляемыми), которые вычисляются внутри ПРВ.

Экзогенные данные являются выходными данными процедур коммуникации данных, и они актуализируются значениями, получаемыми по входным каналам от периферийных подсистем ПРВ.

Эндогенные данные вычисляются внутри ПРВ.

И экзогенные, и эндогенные данные БТД могут быть как интернальными, используемыми локально, так и экстернальными, передаваемыми по выходным каналам периферийным подсистемам.

Объекты коммуникации данных – это множество объектов, которые представляют собой входные и выходные каналы для обмена данными между базой темпоральных данных и периферийными подсистемами. Эти каналы транспонируют значения данных при их обмене.

Система управления базой темпоральных данных обеспечивает контроль валидности и поддержание актуального состояния всех темпоральных данных в каждом тике (моменте) реального времени часов ПРВ. Если какие-либо темпоральные данные утратили свою актуальность, система инициирует транзакцию их актуализации, используя соответствующую вычислительную процедуру и необходимые входные и выходные данные.

9. Опишите формальную модель темпоральных данных.

Формальная модель темпоральных данных описывает структуру и поведение темпоральных данных в системах реального времени. В этой модели данные описываются тремя основными аспектами: значением данных, временным интервалом, когда данные актуальны, и индикатором валидности. Основная идея модели заключается в том, что данные в определенный момент времени могут быть актуальными или неактуальными в зависимости от их периода репрезентативности. Это позволяет системе определять, когда данные нуждаются в обновлении, чтобы оставаться релевантными для текущего состояния системы реального времени. Модель также учитывает возможные задержки в актуализации данных и позволяет различать различные состояния данных в зависимости от их актуальности в каждый конкретный момент времени.

Непрерывное (динамическое) данное — это вид темпорального данного, которое изменяется во времени без привязки к конкретным событиям. В контексте ПРВ оно требует постоянного мониторинга и обновления.

Событийное (статическое) данное — это темпоральное данное, изменение которого происходит только в ответ на определённые события. Между этими событиями значение данного остаётся неизменным.

Система жесткого реального времени: в этом контексте, темпоральные предикаты воспринимаются как фатальные ошибки ПРВ, что означает, что любое нарушение временных ограничений приводит к критическим сбоям в системе.

Система мягкого реального времени: здесь временные ограничения не столь критичны, и редкие нарушения допустимы, при этом система продолжает функционировать.

10. Какие существуют классы темпоральных данных и их семантика?

Класс темпоральных данных "датум" представляет собой модель динамических данных, которые изменяются во времени. Эти данные могут быть экзогенными (непосредственно получаемыми от объекта

мониторинга) или эндогенными (полученными в результате преобразований). Примерами физических параметров, которые могут быть представлены в виде датумов, включают температуру, давление, обороты, координаты, скорость, расход топлива и другие.

Класс темпоральных данных "импульс" представляет собой модель для транспонирования сигналов, которые представляют собой контролируемые "продолжительные" события, которые могут возникать спорадически на объекте мониторинга. Эти события требуют своевременного выполнения определенных действий в системе в течение ограниченного периода времени (период репрезентативности импульса), после чего импульс перестает быть валидным. Примерами таких событий могут быть аварийные ситуации, срабатывание реле, датчиков замыкания/размыкания, оптических датчиков и другие.

Класс темпоральных данных "мода" представляет собой абстрактные статические параметры, которые могут изменяться спорадически во времени. Моды могут представлять собой как параметры конфигурации физического объекта, так и системные параметры, которые влияют на текущий режим работы системы. Значения моды могут устанавливаться до начала работы системы (например, оператором при загрузке), а также в процессе функционирования системы, когда изменения в реальном времени отражаются в соответствующей моде.

11. Охарактеризуйте темпоральные данные класса импульс.

Темпоральные данные класса "импульс" предназначены для представления и управления спорадически возникающими контролируемыми событиями на объекте мониторинга.

Определение класса в контексте темпоральных данных:

- Темпоральные данные класса "импульс" предназначены для отражения сигналов, которые представляют собой спорадически возникающие события на объекте мониторинга. Эти события требуют оперативной реакции в системе в течение ограниченного времени, после чего импульс перестает быть валидным.

Примеры:

- Примерами сигналов, которые могут быть представлены как импульсы, являются сигналы контроля аварийных ситуаций, срабатывание реле или датчиков замыкания/размыкания (концевиков), сигналы срабатывания оптических датчиков, фиксирующих событие распознавания контролируемого объекта, и другие аналогичные сигналы.

Интервал репрезентативности:

- Интервал репрезентативности для импульса ограничен временем, и по истечении этого интервала текущее значение импульса теряет валидность, оставаясь таким до момента следующего спорадически возникающего события, которое актуализирует импульс.

Обновление данных:

- Импульсы обновляются каждый раз, когда событие, на которое они реагируют, происходит. Это означает, что значение может изменяться при каждом новом событии, и интервал репрезентативности будет обновлен.

Процесс актуализации:

- Процесс актуализации импульса начинается с возникновения события, которое он отслеживает. В этот момент значение обновляется, а интервал репрезентативности устанавливается в значение, определенное априорно.

Завершение актуализации:

- Актуализация импульса завершается по истечении интервала репрезентативности. После этого импульс считается неактуальным до возникновения следующего события. Время завершения актуализации совпадает с моментом, когда интервал репрезентативности истекает.

12. Охарактеризуйте темпоральные данные класса датум.

Темпоральные данные класса "датум" представляют собой модель для отслеживания и представления непрерывно изменяющихся во времени параметров или состояний физических объектов или системы в целом.

Определение класса в контексте темпоральных данных:

- Темпоральные данные класса "датум" представляют собой динамические данные, которые непрерывно изменяются во времени. Эти данные могут быть получены непосредственно от

физических объектов мониторинга (экзогенный датум) или могут быть результатом преобразований (эндогенный датум).

Параметры:

- Данные класса "датум" отражают физические параметры объектов мониторинга, такие как температура, давление, обороты, координаты в пространстве, скорость перемещения, расход топлива, объем заполнения емкости и другие подобные параметры.

Интервал репрезентативности:

- Для данных класса "датум" интервал репрезентативности известен заранее. Этот интервал определяется порогом чувствительности к изменениям значений физических параметров во времени относительно предыдущего замера. Такой интервал называется апертурой.

Обновление данных:

- По истечении интервала репрезентативности, вычислительное ядро системы должно одномоментно обновить данные (датум) новыми значениями, полученными от периферийной подсистемы обмена данными с физическим объектом мониторинга.

Экзогенный и эндогенный датум:

- Экзогенный датум получает значения напрямую от физического объекта мониторинга, тогда как эндогенный датум может зависеть от данных класса "мода" и его интервал репрезентативности может завершиться раньше, чем запланировано для датума.

Процесс актуализации:

- Для корректной актуализации датума, обновление должно начаться и завершиться в момент потери валидности датумом. Это означает, что вычисления или получение новых данных должно происходить в течение интервала репрезентативности.

Завершение актуализации:

- По завершении актуализации текущее значение датума обновляется. Интервал репрезентативности заменяется априорно заданным значением, и индикатор валидности устанавливается в 1. Таким образом, актуализированный датум становится валидным и принимает новое значение.

13. Охарактеризуйте темпоральные данные класса мода.

Темпоральные данные класса "мода" представляют собой абстрактные статические параметры, которые могут изменяться во времени спорадически. Эти параметры могут относиться как к физическому объекту мониторинга, так и к системе в целом. Примерами таких параметров могут быть уставки границ изменения значений физических параметров или системные параметры, определяющие текущий режим работы.

Определение класса в контексте темпоральных данных:

- Темпоральные данные класса "мода" представляют собой статические параметры, которые могут изменяться во времени, но это изменение происходит спорадически. Эти параметры могут относиться как к физическому объекту мониторинга, так и к системе в целом. Это абстрактные значения, которые могут быть установлены до начала работы системы или в процессе ее функционирования.

Параметры:

- Значения параметров моды могут изменяться в реальном времени. Это может происходить, например, при изменении состояния переключателей режимов работы системы, или через вычислительные процедуры, которые контролируют актуальность моды и могут вычислять новые значения.

Интервал репрезентативности:

- Для данных класса "мода" интервал репрезентативности является неопределенным, то есть не существует фиксированного интервала, в течение которого мода считается актуальной. Мода может быть актуальной до момента ее изменения.

Контроль актуальности:

- Система управления должна контролировать актуальность моды в каждом временном тике. Если происходит событие, которое указывает на потерю актуальности моды, то мода должна быть обновлена.

Обновление данных:

- Когда мода теряет валидность, ее значение должно быть обновлено. Это обновление должно начаться и завершиться в момент потери валидности моды.

Процесс актуализации:

- Процедура актуализации моды может включать в себя вычисления или обновление значений параметров моды на основе различных факторов или событий.

Завершение актуализации:

- По завершении процедуры актуализации значение моды обновляется, и она становится актуальной с новым интервалом репрезентативности.

14. Какие типы каналов предоставляет БТД для связи с периферийными подсистемами ПРВ.

Каналы – это интерфейсные объекты БТД, используемые процедурами вычислительного ядра для связи экзогенных или экстернальных данных с периферийными подсистемами как абстрактными поставщиками или потребителями значений темпоральных данных, получаемых по каналу.

В соответствии с этим каналы делятся на входные (in), для приёма от периферийных подсистем значений для актуализации экзогенных данных БТД, и выходные (out) – для получения периферийными подсистемами из БТД текущих значений экстернальных данных.

Инициативный канал — это интерфейсный объект, у которого в качестве инициатора посылки в канал нового значения выступает периферийная подсистема, а соответствующая процедура вычислительного ядра ПРВ ожидает его прихода.

Пассивный канал — это интерфейсный объект, у которого в качестве инициатора посылки в канал нового значения выступает процедура вычислительного ядра ПРВ, а соответствующая периферийная подсистема ожидает запроса на его посылку.

Инициативный входной канал (in) с протоколом $proactive_{in}$:

- Определение: этот канал позволяет периферийным подсистемам инициировать отправку данных в БТД для актуализации экзогенных данных класса "импульс" или "мода" в реакции на возникновение события.
- Процесс актуализации: вычислительная процедура в ядре ПРВ находится в состоянии пассивного ожидания и ожидает поступления нового значения темпорального данного в канал от периферийной подсистемы, когда происходит спорадическое событие.

Пассивный входной канал (in) с протоколом $inactive_{in}$:

- Определение: этот канал используется для актуализации экзогенных данных класса "датум" в БТД.
- Процесс актуализации: периферийная подсистема ожидает запроса на отправку в канал нового значения физического параметра после истечения интервала репрезентативности.

Инициативный выходной канал (out) с протоколом $proactive_{out}$:

- Определение: этот канал инициирует отправку текущего значения экстернального данного класса "датум" из БТД.
- Процесс актуализации: когда в вычислительной процедуре ядра ПРВ возникает запрос от периферийной подсистемы на получение данных, она инициирует отправку текущего значения в выходной канал.

Пассивный выходной канал (out) с протоколом $inactive_{out}$:

- Определение: этот канал используется для отправки в выходной канал вновь актуализированного значения экстернального данного любого класса.
- Процесс актуализации: периферийная подсистема пассивно ожидает прихода в канал нового значения, а инициатором посылки является соответствующая процедура вычислительного ядра ПРВ, ожидающая очередной актуализации экстернального данного БТД.

15. Что называется темпоральным прецедентом БДТ и причины его возникновения?

Темпоральный прецедент потери данного – ситуация, когда данные (например, информация или результаты вычислений), связанные с определенным моментом времени, утрачивают свою актуальность в текущий момент времени.

Обусловлено тем, что может быть высока вероятность того, что начатые в текущий момент времени в ПРВ (функционирующего в заданных вычислительных ресурсах) необходимые для актуализации темпоральных данных (утративших валидность в текущий момент времени) параллельные вычисления не смогут все завершить своё выполнение в этот же момент времени системных часов ОСРВ. В итоге, результаты таких вычислений актуализируют темпоральные данные с опозданием (показание времени системных часов уже изменилось).

Темпоральный прецедент является следствием отложенного начала и/или запоздалого завершения процедуры обновления темпорального данного.

16. Что понимается под режимами жёсткого и мягкого реального времени работы СРВ?

Режимы жёсткого и мягкого реального времени работы систем реального времени (СРВ) относятся к различным уровням строгости соблюдения временных ограничений.

В жёстком режиме реального времени система должна выполнять задачи в строго определённые временные рамки. Любое нарушение временных ограничений рассматривается как критическая ошибка, что может привести к серьёзным последствиям. Для таких систем условие одномоментности актуализации темпоральных данных не может нарушаться в течение всего времени работы системы.

В мягком режиме реального времени система также стремится соблюдать временные ограничения, однако периодические и небольшие нарушения временных рамок допустимы и не ведут к катастрофическим сбоям. Система может продолжать работу, но с потенциально ухудшенной производительностью.

Система жесткого реального времени: в этом контексте, темпоральные предикаты воспринимаются как фатальные ошибки ПРВ, что означает, что любое нарушение временных ограничений приводит к критическим сбоям в системе.

Система мягкого реального времени: здесь временные ограничения не столь критичны, и редкие нарушения допустимы, при этом система продолжает функционировать.

17. Формальный метод оценки валидности темпоральных данных в режиме мягкого реального времени.

Формальный метод оценки валидности темпоральных данных в режиме мягкого реального времени основываются на уровне строгости соблюдения временных ограничений:

В режиме мягкого реального времени допускается кратковременная потеря валидности темпоральных данных.

Вводится вещественный коэффициент валидности $\tilde{\nu} \in [0,1]$, который характеризует степень условной валидности (практической пригодности) использования не валидного импульса или датума. Если коэффициент валидности $\tilde{\nu} = 1$, то темпоральное данное является валидным.

Если коэффициент валидности $\tilde{\nu} = 0$, то темпоральное данное не является валидным и не может использоваться в вычислениях.

Если коэффициент валидности $0 \ll \tilde{\nu} < 1$, то темпоральное данное является нечётким во времени, а величина коэффициента валидности (степень валидности) характеризует практическую значимость использования в вычислениях нечёткого во времени значения темпорального данного.

18. Формальный метод оценки валидности темпоральных данных в режиме жёсткого реального времени.

Формальный метод оценки валидности темпоральных данных в режиме жёсткого реального времени основываются на уровне строгости соблюдения временных ограничений:

В отличие от мягкого режима, жёсткий режим реального времени не позволяет допускать деградацию данных. В этом режиме временные ограничения строго соблюдаются, и нарушение этих ограничений может привести к аварийной остановке работы системы. Жёсткий режим требует более строгой синхронизации и актуальности данных, не допуская использование деградированных или условно валидных значений.

19. Если на компьютере с именем comp выполнена команда shell:
#mount –T io-net /lib/dll/npm-qnet.so,
то в файловой системе компьютера comp …

Появляется каталог "/net", который содержит в себе каталоги с именами, соответствующими примонтированным к сети узлам. Эти каталоги в "/net" позволяют доступ к ресурсам узлов сети по пути "/net/<имя_узла>/<имя_ресурса>".

Выполнение данной команды приводит к присоединению компьютера в качестве узла к локальной сети (монтирование сети) и запуску администратора сети (системного процесса npm-qnet, реализующего протокол Qnet для работы в сети)

20. Какой процесс будет считаться родительским при запуске процесса функцией system()?

Командный интерпретатор shell

Можно запустить процесс посредством не ручного, а программного вызова командного интерпретатора shell для выполнения любой команды и, в частности, команды запуска процесса в виде имени исполняемого модуля, на базе которого процесс будет порождаться:

int system(const char *command);

Поведение функции зависит от значения аргумента command. Если command равен NULL, то определяется, имеется ли в системе shell. Функция возвращает ноль, если shell отсутствует, или отличное от нуля значение, если присутствует. Если command не равен NULL, то запускается копия shell и ему, через указатель command, передаётся командная строка для обработки и возвращается результат выполнения shell. Если shell не смог загрузиться, то возвращается -1. При успешной загрузке shell возвращается статус завершения его выполнения, соответствующий выполненной команде.

ДОПОЛНИТЕЛЬНАЯ ОБЩАЯ ИНФОРМАЦИЯ

В ОС QNX разработка приложений базируется на использовании таких конструктивных элементов как исполняемые программные модули, процессы, нити.

В общем случае приложение реального времени в операционной системе QNX является многопроцессным. Процессы запускаются на основе предварительно созданных исполняемых программных модулей (файлов). Каждая загрузка на выполнение программного модуля рассматривается и реализуется системой как запуск некоторого нового процесса. Одновременно могут выполняться любое количество процессов. Процессы выполняются либо независимо, либо взаимодействуя друг с другом посредством обмена сообщениями или разделяемой системной памяти.

Процессы имеют каналы для приёма сообщений и соединения с каналами для посылки сообщений, а также имеют доступ к разделяемой системной памяти.

Исполнение процесса начинается с особого запуска операционной системой функции main() соответствующего программного модуля. Такой способ запуска функции main() интерпретируется как создание в процессе потока управления, для обозначения которого далее будет использоваться термин "нить" (thread).

Все нити, запущенные во всех процессах, выполняются параллельно. При этом каждая нить уже не имеет внутреннего программного параллелизма и рассматривается как программная процедура последовательно выполняемых инструкций. В итоге, исполнение программы в процессе в общем случае выглядит как запуск нити main() и последующее порождение и параллельное выполнение совокупности нитей, которые могут асинхронно создаваться и выполняться, разделяя общие ресурсы, а также асинхронно завершаться.

Механизм процессов обеспечивает при разработке логической структуры ПРВ структурный параллелизм (модульность) ПРВ, а механизм нитей - функциональный параллелизм ПРВ.

Процессы

Процесс выступает в качестве обособленной вычислительной среды, обеспечивающей в общем случае параллельное и асинхронное выполнение запущенных в нём вычислительных процедур (функций) или так называемых – нитей.

Жизненный цикл процесса в операционной системе включает четыре основных этапа.

1) Создание: Процесс может быть создан только другим (родительским) процессом. При этом администратор процессов формирует для него необходимую среду выполнения и создаёт у себя системную структуру атрибутов – дескриптор, характеризующих свойства процесса, используемых для управления созданным процессом.
2) Загрузка: Код и данные заданного программного модуля загружаются в выделенную процессу память и запускается нить main().
3) Выполнение: В процессе, в общем случае, параллельно и асинхронно выполняются создаваемые нити
4) Завершение: Завершение процесса проходит две стадии. На первой стадии происходит освобождение ресурсов, связанных с процессом (страницы ОЗУ, открытые файловые дескрипторы и т.п.). На второй стадии код возврата завершаемого процесса передаётся его процессу-родителю. При этом, в

зависимости от режима создания дочернего процесса возможны следующие варианты поведения процесса-родителя:

- процесс-родитель блокируется в ожидании получения кода завершения дочернего процесса. Код завершения доставляется процессу-родителю, он выходит из блокированного состояния, а дочерний процесс терминируется;
- процесс-родитель не блокируется в ожидания и отказывается от получения кода завершения дочернего процесса. В этом случае дочерний процесс становится независимым от родительского процесса;
- процесс-родитель не блокируется в ожидании, но не отказывается от получения кода завершения дочернего процесса. Дочерний процесс при завершении становится DEAD-блокированным процессом или "зомби". Для такого процесса администратор процессов сохраняет минимум управляющей информации, необходимой только для того, чтобы доставить код завершения родительскому процессу, когда он выполнит переход в ожидание завершения дочернего процесса.

Типы процессов:

- Системные процессы: Системные процессы являются процессами, порождаемыми ядром или специальными системными программами ОС. Системные процессы, порождаемые ядром, не имеют исполняемых модулей и запускаются особым образом при инициализации ядра системы. Выполняемые инструкции и данные этих процессов находятся в ядре системы, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов.
- Процессы демоны: Демоны — это не интерактивные процессы, которые запускаются обычным образом - путём загрузки в память соответствующих им программных модулей (исполняемых файлов), и выполняются в фоновом режиме. Обычно демоны запускаются при инициализации системы и обеспечивают работу различных подсистем ОС: терминального доступа, печати, сетевого доступа и т.п. Демоны не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть времени демоны ожидают пока тот или иной процесс запросит у ядра определённую услугу, которую ядро перенаправит соответствующему демону, например, запрос на запуск процесса или открытие файла и т.п.
- Прикладные процессы: К прикладным процессам относятся все остальные процессы, выполняющиеся в системе. К ним, как правило, относят процессы, порождённые в рамках пользовательского сеанса работы. Важнейшим из таких процессов является командный интерпретатор, который назначается пользователю при регистрации и запускается при входе пользователя в систему (login shell) и обеспечивает его работу в QNX. Завершение работы login shell приводит к отключению от системы. Пользовательские процессы могут выполняться как в интерактивном, так и в фоновом режиме, но в любом случае при завершении сеанса работы пользователя и выходе из системы все его процессы терминируются.

21. Какие функции позволяют запустить процесс на удалённом узле?

spawn()

Функция предоставляет максимальные средства управления запуском дочернего процесса как локально, так и в сети.

```
pid_t spawn(
const char *path, // полное имя исполняемого модуля, используемого родителем для запуска дочернего процесса
int fd_count, // число элементов в массиве fd_map
const int fd_map[], // массив ограниченного набора дескрипторов файлов, открытых в родительском процессе
const struct inheritance *inherit, // структура системного типа для настройки свойств дочернего процесса
char *const argv[], // вектор аргументов
char *const envp[]); // вектор указателей строк с определением переменных среды
```

22. Являются ли вызовы MsgReply(),MsgRead(),MsgWrite() блокирующими?

Функции MsgReply(), MsgRead(), MsgWrite() и им подобные при взаимодействии удаленных клиента и сервера в сети становятся блокирующими вызовами, т.к. при их реализации используются услуги администратора npm-qnet, которому ядро перенаправляет вызов как процессу-серверу. Выход из блокирующего состояния происходит, когда администратор npm-qnet либо корректно, либо с ошибкой завершает доставку сообщения

23. Какой вид имеет объявление функции, используемой для запуска нитей?

```
int pthread_create(
pthread_t *thread, // сохранится дескриптор потока
const pthread_attr_t *attr, // передаваемые атрибуты
```

void *(*start_routine)(void*), // функция запускаемая как нить
void *arg // передаваемые аргументы);

Функция создаёт и запускает как нить функцию, указанную в аргументе start_routine, с атрибутами, заданными в атрибутной записи, указанной в аргументе attr. Если предполагается использовать атрибуты нити по умолчанию, то attr можно положить равным NULL. При запуске нити ей в качестве аргумента функции start_routine передаётся указатель arg. Если функция без аргументов, то аргумент arg делают равным NULL. Аргумент thread указывает переменную, в которой сохраняется дескриптор (ID) созданной нити. Если необходимости в дескрипторе нити нет, можно задать NULL.

24. Может ли нить main() быть обособленной?

Нить main() не может быть обособленной.

Она ждет завершения выполнение всех других нитей. Завершение нити main() является исключительным, так как приводит к автоматическому завершению выполнения всех созданных нитей и удалению процесса. Таким образом, процесс является, по сути, контейнером созданных и параллельно выполняющихся нитей, в котором, в момент старта, системой автоматически создаётся и запускается на выполнение нить - main().

25. Может ли нить main() быть присоединяемой?

Нить main() не может быть присоединяемой.

Она ждет завершения выполнение всех других нитей. Завершение нити main() является исключительным, так как приводит к автоматическому завершению выполнения всех созданных нитей и удалению процесса. Таким образом, процесс является, по сути, контейнером созданных и параллельно выполняющихся нитей, в котором, в момент старта, системой автоматически создаётся и запускается на выполнение нить - main().

26. Какие параметры стека создаваемой нити можно задать одновременно?

1) Адрес стека (stackaddr): Этот параметр позволяет явно указать адрес области памяти, которая будет использоваться в качестве стека для создаваемой нити. Если stackaddr установлен в NULL, то стек будет формироваться по умолчанию
2) Размер стека (stacksize): Этот параметр позволяет задать размер стека в байтах. Новый стек будет иметь указанный размер. Реальный размер стека будет округлен до ближайшего кратного размера страницы памяти.

27. Какие значения приоритета можно назначить нити в QNX?

- 0 - системный процесс idle;
- от 1 до 63 - непривилегированная нить;
- от 1 до 255 - привилегированная нить (у которых EUID=0 - эффективный идентификатор владельца)

Каждому запущенному в системе процессу в дескрипторе устанавливаются два параметра - так называемые эффективный идентификатор владельца (EUID) и эффективный идентификатор группы (EGID). Для процессов с EUID=0 (т.е. для пользователя root) доступ к файлам предоставляется без процедуры проверки прав.

Каждая созданная нить может находиться в системе в одном из следующих состояний:

- Состояние активности – нить в данный момент выполняется системой.
- Состояние готовности – нить может выполняться и ждёт, когда система предоставит ей процессорное время.
- Блокированное состояние – нить не может выполняться, так как её дальнейшее выполнение зависит от определённых ожидаемых условий.

Нить в состоянии готовности постоянно конкурирует с другими готовыми и с активной нитью за переход в активное состояние. Конкуренция нитей осуществляется в соответствии с приоритетом, полученным при их создании. Приоритет нити является положительным целым числом в диапазоне от 1 до 63. Правило конкуренции заключается в том, что среди готовых нитей в состоянии активности может находиться нить, имеющая наибольший приоритет. Как только нить с более высоким, чем у текущей активной нити, приоритетом становится готовой, активная нить вытесняется (принудительно переводится в состояние готовности), а более приоритетная нить становится активной. Если среди готовых к выполнению нитей одинаково высокий приоритет имеют несколько нитей, то активной станет нить, которая раньше других оказалась в состоянии готовности.

28. Активная нить выполнила функцию освобождения процессора - sched_yield(), какая из готовых к выполнению нитей начнёт выполняться?

Если окажется готовой другая нить с таким же приоритетом, то она станет активной. Если такой нити не окажется, то данная нить вновь станет активной и продолжит работу.

29. Что понимается под *инверсией приоритетов* нитей при выполнении ПРВ?

Инверсия приоритетов нитей (Priority Inversion) — это ситуация, когда нить с более низким приоритетом временно получает преимущество выполнения перед нитью с более высоким приоритетом, в то время как готовые к выполнению нити с более высоким приоритетом оказываются заблокированными.

30. Какие функции связывают продолжение выполнения одной нити с моментом завершения существования другой нити?

Вид синхронизации нитей, при котором момент начала продолжения одной нити согласуется с моментом выполнения другой нити обеспечивается механизмом, который называется присоединением. Присоединение позволяет одной нити блокироваться в состоянии ожидания завершения выполнения другой нити.

Для этого нить, к которой осуществляется присоединение, должна допускать возможность присоединения (т.е. должна быть запущена с атрибутом присоединяющей нити).

Для присоединения к нити с целью ожидания её завершения присоединяющаяся нить должна выполнить функцию:

int pthread_join(
pthread_t thread, //ID-нити присоединения
void **value_ptr); //возвращаемое нитью значение

Если значение, возвращаемое нитью при завершении, не представляет интереса, то аргумент value_ptr следует положить равным NULL.

ДОПОЛНИТЕЛЬНАЯ ОБЩАЯ ИНФОРМАЦИЯ

Методы и функции синхронизации нитей

Присоединение

Присоединение позволяет одной нити блокироваться в состоянии ожидания завершения выполнения другой нити. Для этого нить, к которой осуществляется присоединение, должна допускать возможность присоединения (т.е. должна быть запущена с атрибутом присоединяющей нити)

int pthread_join(
pthread_t thread, // ID-нити присоединения
void **value_ptr); // возвращаемое нитью значение

Барьеры

Барьер - метод синхронизации, позволяющий нити перейти в состояние ожидания у барьера, как у закрытого "шлагбаума", сбора заданного числа нитей. Если нить достигает барьера, а количество нитей, ранее достигших барьера, меньше заданного для барьера значения, то выполнение нити приостанавливается (блокируется). После того, как заданное число нитей соберётся у барьера, все эти нити деблокируются и становятся готовыми для выполнения.

Может быть локальный и глобальный.

int pthread_barrier_init(pthread_barrier_t *barrier, const pthread_barrierattr_t *attr, int count); // определение переменной системного типа pthread_barrier_t и инициализация её атрибутов

int pthread_barrierattr_init(pthread_barrierattr_t *attr); // инициализация атрибутной переменной

int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr, int pshared); // формирование в атрибутной записи соответствующее значение

Мутексы

Мутекс - метод синхронизации, обеспечивающий нитям взаимное исключение по отношению к некоторому общему разделяемому нитями ресурсу, не допускающему его одновременное использование более одной нитью. Иными словами, в каждый момент времени ресурсом может владеть не более чем одна нить.

Логика управления мутексом выражается в его захвате и освобождении нитями. Если некоторая нить выполняет вызов захвата мутекса, уже ранее захваченного другой нитью, то она блокируется ядром до момента освобождения мутекса нитью его захватившей.

Может быть локальный и глобальный.
Поддерживает настройку инверсии приоритетов.

int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int pthread_mutexattr_init(const pthread_mutexattr_t *attr);

- Захват
  int pthread_mutex_lock (pthread_mutex_t *mutex);
  int pthread_mutex_trylock(pthread_mutex_t *mutex);
- Освобождение
  int pthread_mutex_unlock (pthread_mutex_t *mutex);
- Уничтожение
  int pthread_mutex_destroy (pthread_mutex_t *mutex);

Блокировка чтения/записи

Блокировка чтения/записи - метод синхронизации, согласующий поведение нитей по отношению к содержимому общей области памяти, требуемой одновременно нескольким нитям для чтения или записи. При этом нити-читатели могут одновременно читать содержимое памяти, исключая при этом доступ к ней нитей-писателей, или одна нить-писатель может изменять содержимое памяти, исключая при этом доступ к ней нитей-читателей и других нитей-писателей.

Главный принцип: несколько нитей могли одновременно считывать данные, и только одна нить могла их изменять.

Может быть локальной и глобальной.

int pthread_rwlock_init (pthread_rwlock_t *rwlock, const pthread_rwlockattr_t *attr);

int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

- Захват для читателя
  int pthread_rwlock_rdlock (pthread_rwlock_t *rwlock);
  int pthread_rwlock_tryrdlock (pthread_rwlock_t *rwlock);
- Захват для писателя:
  int pthread_rwlock_wrlock (pthread_rwlock_t *rwlock);
  int pthread_rwlock_trywrlock (pthread_rwlock_t *rwlock);
- Освобождение
  int pthread_rwlock_unlock (pthread_rwlock_t *rwlock);
- Уничтожение
  int pthread_rwlock_destroy (pthread_rwlock_t *rwlock);

Условные переменные

В ряде случаев захват ресурса не предполагает безусловную готовность ресурса к его обработке захватившей нитью. Нить предварительно должна проверить условие готовности ресурса к обработке. Если условие выполняется, то обработка осуществляется и при её завершении ресурс освобождается. Если условие не выполняется, то нить вынуждена освобождать ресурс, чтобы предоставить возможность доступа к нему другим нитям, которые, в общем случае, могут влиять на формирование результата проверяемого условия. Например, нить должна дождаться, когда X станет равным Y прежде, чем продолжит своё выполнение.

Метод основан на управлении мутексом и программным объектом системного типа pthread_cond_t, называемым условной переменной. Метод позволяет одним нитям ждать уведомления, посылаемые другими нитями, использующими в качестве посредника одну и ту же условную переменную.

Может быть локальной и глобальной.

- Создание условной переменной
  int pthread_cond_init (pthread_cond_t *condvar, pthread_condattr_t * attr);
- Ожидание уведомления по условной переменной
  int pthread_cond_wait (pthread_cond_t * condvar, pthread_mutex_t *mutex);
- Отправка уведомления по условной переменной
  int pthread_cond_broadcast (pthread_cond_t *condvar); (всем)

int pthread_cond_signal (pthread_cond_t *condvar); (с мин. приор. и с макс. ожиданием)

Ждущие блокировки

Этот метод является частным случаем метода условной переменной и не относится к стандарту POSIX и обеспечивается функциями QNX.

Метод использует виртуальный объект sleepon - "ждущая блокировка", и основан на использовании одного неявно создаваемого ядром системы мутекса и неявно используемой ядром системы условной переменной, в роли которой неявно может быть использована системой любая доступная нитям переменная.

- Захват ждущей блокировки
  int pthread_sleepon_lock (void);
- Освобождение ждущей блокировки
  int pthread_sleepon_unlock (void);
- Ожидание уведомления
  int pthread_sleepon_wait (const volatile void *addr);
- Отправка уведомления
  int pthread_sleepon_broadcast(const volatile void *addr); (все)
  int pthread_sleepon_signal (const volatile void *addr); (с мин. приор. – не определён)

Семафоры

Семафоры — это метод синхронизации, основанный на создании и управлении программным объектом системного типа sem_t, регулирующим количество нитей, осуществляющих одновременный доступ к некоторому ресурсу.

Управление семафором выражается в его захвате и освобождении. Семафор ведёт счётчик захватов. Начальное значение счётчика захватов семафора устанавливается при его создании. Если нить пытается захватить семафор, счётчик которого не больше 0, то она блокируется до момента, когда значение счётчика не станет больше 0. Семафоры бывают неименованные и именованные.

Неименованные семафоры:

- Эти семафоры создаются внутри процесса и не имеют имени в файловой системе.
- Обычно используются для синхронизации между нитями внутри одного процесса.
- Создаются с помощью функции sem_init() и уничтожаются с помощью функции sem_destroy().
- Не подходят для синхронизации между разными процессами или на удалённых узлах сети.

Именованные семафоры:

- Эти семафоры имеют уникальное имя в файловой системе и могут использоваться для синхронизации между процессами.
- Создаётся как файл особого типа, регистрируемый в файловой системе в каталоге /dev/sem
- Создаются и уничтожаются с помощью функций sem_open() и sem_unlink().
- Могут быть использованы для синхронизации между разными процессами, в том числе на удалённых узлах сети.

31. Рекурсивный мутекс позволяет нити…

Избежать «мёртвой блокировки».

В некоторых случаях требуется определять мутекс как рекурсивный (считающий). Это необходимо, когда выполнение нити, захватившей мутекс, предполагает, например, вызов различных функций, которые в свою очередь в процессе выполнения осуществляют захват этого же мутекса. Если не допускать повторного захвата мутекса одной и той же нитью, то возникнет "мертвая" блокировка (deadlock). Рекурсивный мутекс позволяет избежать этого. Повторный захват одной и той же нитью рекурсивного мьютекса не приводит к блокированию нити. При этом ядро контролирует количество захватов и освобождений рекурсивного мьютекса. Чтобы мутекс был рекурсивным его при определении необходимо явно проинициализировать значением PTHREAD_RMUTEX_INITIALIZER

32. Если нити требуется исключительный доступ к некоторому ресурсу, но использовать ресурс нить может только при выполнении определённых условий, то каким из приведённых ниже механизмов синхронизации можно воспользоваться наиболее эффективно?

Метод условной переменной (ждущей блокировки) – это метод синхронизации, основанный на использовании одного неявно создаваемого ядром системы мутекса и неявно используемой ядром системы условной переменной, в роли которой неявно может быть использована системой любая доступная нитям переменная

33. Какой объект управления синхронизацией нитей создаётся функцией: int sem_init(sem_t *sem, int pshared, unsigned value) ?

Неименованый семафор

Семафоры — это метод синхронизации, основанный на создании и управлении программным объектом системного типа sem_t, регулирующим количество нитей, осуществляющих одновременный доступ к некоторому ресурсу. Управление семафором выражается в его захвате и освобождении. Семафор ведёт счетчик захватов. Начальное значение счётчика захватов семафора устанавливается при его создании. Если нить пытается захватить семафор, счётчик которого не больше 0, то она блокируется до момента, когда значение счетчика не станет больше 0. Семафоры бывают неименованные и именованные.

Неименованные семафоры:

- Эти семафоры создаются внутри процесса и не имеют имени в файловой системе.
- Обычно используются для синхронизации между нитями внутри одного процесса.
- Создаются с помощью функции sem_init() и уничтожаются с помощью функции sem_destroy().
- Не подходят для синхронизации между разными процессами или на удалённых узлах сети.

Именованные семафоры:

- Эти семафоры имеют уникальное имя в файловой системе и могут использоваться для синхронизации между процессами.
- Создаётся как файл особого типа, регистрируемый в файловой системе в каталоге /dev/sem
- Создаются и уничтожаются с помощью функций sem_open() и sem_unlink().
- Могут быть использованы для синхронизации между разными процессами, в том числе на удалённых узлах сети.

Если аргумент pshared отличен от нуля, то семафор может разделяться нитями различных процессов, при условии, что семафор создан в разделяемой процессами памяти. С помощью аргумента value задаётся начальное значение счётчика семафора. После инициализации семафора указатель sem используется в функциях управления семафором. Для аннулирования неименованного семафора используется функция int sem_destroy(sem_t *sem);

34. Какая функция управления семафором приводит к увеличению значения счётчика семафора на 1?

sem_post(sem_t *sem)

Функция sem_post() увеличивает счётчик семафора sem на 1. Если имеются нити, которые в настоящее время блокированы, ожидая семафор, то одна из этих нитей возвратится успешно из вызова sem_wait. Нить, которая будет разблокирована первой, определяется в соответствии с приоритетом и временем ожидания (с наибольшим приоритетом, которая ждала дольше всех). Функция sem_post() может быть вызвана обработчиком сигналов.

35. Механизм отображения адресов системных областей памяти в адресное пространство процесса используется для доступа процесса к …

Адресам физических устройств или возникает потребность в оперативной памяти за пределами локальной памяти процесса (например, разделяемой различными процессами)

Если процессам требуется доступ к адресам физических устройств или возникает потребность в оперативной памяти за пределами локальной памяти процесса (например, разделяемой различными процессами), то операционная система предоставляет процессам такую возможность посредством механизма отображения адресов системных областей памяти в адресное пространство процесса. Отображение адресов системных областей памяти реализуется процессами с помощью специальных запросов к операционной системе, в результате успешного выполнения которых между адресами системной памяти и выделенными адресами адресного пространства процесса устанавливается взаимно-однозначное соответствие. В результате обращение процесса к этим адресам памяти преобразуется операционной системой в обращение к соответствующим адресам системной памяти.

36. Если процессу приходит сигнал, а диспозиция сигнала явно не установлена, то как это отразится на процессе?

Если процесс вообще не устанавливает для сигнала диспозицию или явно устанавливает диспозицию по умолчанию, то будет действовать диспозиция по умолчанию. Действие по умолчанию определено в ядре для каждого сигнала. В большинстве случаев по умолчанию при получении процессом сигнала происходит завершение его выполнения, а для сигналов SIGCHLD, SIGIO, SIGURG и SIGWINCH в качестве действия по умолчанию предписано игнорировать сигнал.

1) Если процессу не предписано выполнять каких-либо специальных действия по обработке сигнала, то по умолчанию поступление сигнала прекращает выполнение процесса;
2) Процесс может проигнорировать сигнал. В этом случае выдача сигнала не влияет на работу процесса (обратите внимание на то, что сигналы SIGCONT, SIGKILL и SIGSTOP не могут быть проигнорированы при обычных условиях);
3) Процесс может иметь обработчик сигнала, которому передается управление при поступлении сигнала. В этом случае говорят, что процесс может «ловить» сигнал. Фактически такой процесс выполняет обработку программного прерывания. Данные с сигналом не передаются.

37. Если сигнал генерируется функцией kill(pid, sig), где pid>0, то кому адресуется сигнал?

Когда сигнал генерируется функцией kill(pid, sig), где pid > 0, он адресуется процессу с указанным идентификатором (PID).

Если pid>0, то адресуется единственный процесс.

Если pid=0, то сигнал посылается всем процессам, входящим в группу (идентификатор группы процессов которых равен идентификатору отправителя), которой принадлежит пославший сигнал процесс.

Если pid<0, то сигнал посылается каждому процессу, являющемуся членом группы процессов с GID=-pid.(идентификатор группы процессов которых равен абсолютному значению pid)

Если sig равен 0, то сигнал не посылается, а проверяется возможность послать сигнал по указанному pid (проверка наличия адресата).

38. Какие функции позволяют послать сигнал процессу, находящемуся на удалённом узле локальной сети?

SignalKill()

Сигнал можно послать на удаленный узел, а также адресовать его непосредственно указанной нити.

Использование для посылки процессу сигнала функция kill() имеет ограниченные возможности. Во-первых, адресатом сигнала, посылаемого с помощью функции kill(), является процесс (процессы). Воздействие такого сигнала на нити процесса (если их более одной), не всегда очевидна. Во-вторых, если работа осуществляется в сети, функция предоставляет возможность адресовать сигнал процессам только местного узла.

39. Какие функции позволяют установить обработчик сигналов в режиме, который предусматривает возможность образования очереди сигналов?

sigaction().

В установке диспозиций сигналов могут участвовать все нити процесса. Однако в итоге сигналу в процессе можно назначить только одну диспозицию. Это будет та диспозиция, которую сформировала нить, последней выполнившая установку диспозиции этому сигналу. Все сформированные нитями диспозиции сигналов принадлежат процессу, а не конкретной нити. Вместо функции signal() механизм надёжных сигналов использует функцию sigaction(), позволяющую установить диспозицию сигнала, узнать текущую диспозицию или выполнить и то и другое одновременно.

40. Укажите имя пользователя, которое автоматически регистрируется при установке ОС QNX.

Root

Сервис глобальных имён в локальной QNET-сети стал возможным, начиная с QNX версии 6.3. Этот сервис обеспечивается так называемым GNS-менеджером (утилита gns), которая должна быть загружена в узлах локальной сети, в которых процессы будут создавать или открывать глобальные именованные каналы.

Используя этот сервис, процесс-сервер может создавать глобальные именованные каналы, а процессы-клиенты, используя глобальное имя канала, могут присоединяться к ним как локально, так и через QNET-сеть. Процессы могут воспользоваться сервисом глобальных имён, только если они имеют привилегии администратора системы (root).

41. **Клиент посылает 120 Кб данных серверу на удалённом узле. Какой объем данных будет получен сервером при выполнении функции MsgReceive()?**

    Не более 8 Кб

    Когда завершается запрос MsgReceive(), сервер может и не получить от клиента сообщение в полном объёме даже при наличии у сервера достаточного объёма буфера приёма сообщения. Это опять связано с тем, что перенаправленное ядром сообщение клиента предварительно принимает в собственный буфер администратор сети npm-qnet на узле клиента, который затем передаёт принятое сообщение администратору сети npm-qnet на удалённом узле сервера. Так как размер сообщения заранее администратору npm-qnet не известен, то он за один раз получает объем данных, не превышающий фиксированного максимального размера его буфера (в настоящее время 8KB).

42. **Родительский процесс, находящийся на узле с именем compA, запустил дочерний процесс на удалённом узле с именем compB и передал ему необходимые параметры. Какая последовательность действий дочернего процесса является верной для установления соединения с родительским процессом?**

43. **Какова особенность запуска дочернего процесса с помощью функции fork()?**

    Родительский процесс порождает свою копию, после чего оба процесса продолжат выполняться с оператора, следующего за вызовом функции fork().

    Функция fork() создаёт новый (дочерний) процесс, являющийся точной копией процесса, его породившего (родительского). При этом дочерний процесс имеет уникальный PID.

    После выполнения функции fork() родительским процессом оба процесса продолжат выполняться с оператора, следующего за вызовом функции fork()

44. **Какова особенность запуска дочернего процесса с помощью функции vfork()?**

    Родительский процесс порождает свою копию и ждет завершения дочернего процесса.

    Функция vfork() создаёт новый процесс, как и функция fork(), но в разделяемом адресном пространстве, и блокирует родительский процесс до тех пор, пока дочерний процесс не завершится или не вызовет функцию семейства exec*().

45. **В каком состоянии окажется процесс-сервер, выполнивший функцию MsgReceive(), если процесс-клиент был в Send-блокированном состоянии?**

    В состоянии RECEIVE-блокировки

    MsgReceive() используется сервером для приема сообщений от клиента, и после успешного приема, сервер остается в состоянии ожидания новых сообщений. Поэтому состояние сервера не меняется с RECEIVE-блокировки на другое состояние после выполнения MsgReceive(), если он ожидает новых сообщений.

46. **Что обеспечивает механизм векторов ввода/вывода при обмене сообщением между клиентом и сервером?**

    Механизм векторов ввода/вывода (IOV) обеспечивает гибкость при обмене сообщениями между клиентом и сервером, когда сообщение состоит из несвязанных частей.

    Если сообщение состоит из несвязных частей. При передаче несвязного сообщения для каждой части такого сообщения формируется свой вектор ввода/вывода. Затем из них формируется массив векторов ввода/вывода, в котором вектора располагают в нужном порядке следования частей сообщения при передаче. Для посылки клиентом сообщения, части которого находятся в несвязной области памяти, представленной массивом векторов IOVэ.

    Вектор ввода/вывода (IOV) — это структура, которая содержит адрес и длину части сообщения. Этот механизм позволяет эффективно передавать или получать данные, несмотря на то что они могут располагаться в разных областях памяти.

47. **Могут ли в приложении быть запущены нити с разными дисциплинами диспетчеризации?**

    Могут.

(из прочитанного не следует, что все нити должны иметь одинаковую дисциплину диспетчеризации).

При создании процесса его приоритет и дисциплина диспетчеризации наследуются нитью main() от родительского процесса и могут при необходимости программно меняться при запуске в процессе и выполнении нитей.

48. В каком состоянии окажется процесс-клиент, выполнивший функцию MsgSend(), если процесс-сервер был в Receive-блокированном состоянии?

В состоянии SEND-блокировки.

Функция MsgSend() используется клиентом для отправки сообщения серверу. Если процесс-сервер находится в состоянии RECEIVE-блокировки (ожидание прихода сообщения), то пославшая сообщение нить клиента будет поставлена в очередь к каналу, и она будет ожидать, пока процесс-сервер получит и обработает это сообщение.

49. Если для управления доступом нитей к некоторому разделяемому программному ресурсу создан мутекс, то что предпримет ядро по отношению к нити, которая попытается осуществить доступ к этому ресурсу, не захватывая мутекс?

Ничего

Мутекс - метод синхронизации, обеспечивающий нитям взаимное исключение по отношению к некоторому общему разделяемому нитями ресурсу, не допускающему его одновременное использование более одной нитью.

50. Что произойдёт с нитью, если она по отношению к блокировке чтения/записи выполнит функцию pthread_rwlock_wrlock() или pthread_rwlock_rdlock(), а блокировка чтения/записи уже захвачена на чтение?

pthread_rwlock_rdlock() Функция либо блокирует выполнение нити-читателя, если блокировка чтения/записи ранее уже была захвачена другой нитью-писателем, либо осуществляется захват блокировки чтения/записи нитью.

51. Что произойдёт с нитью, если она по отношению к блокировке чтения/записи выполнит функцию pthread_rwlock_wrlock() или pthread_rwlock_rdlock(), а блокировка чтения/записи уже захвачена на запись?

pthread_rwlock_wrlock() Функция либо блокирует выполнение нити-писателя, если блокировка чтения/записи ранее уже была захвачена другой нитью-писателем или нитью-читателем, либо осуществляется захват блокировки чтения/записи нитью-писателем

52. Какой объект синхронизации нитей создаётся функцией:
sem_t* sem_open(char* sem_name,int oflags,mode_t mode,unsigned value); ?

Именованный семафор.

Функция sem_open() позволяет создать в файловой системе новый именованный семафор и открыть к нему доступ, если семафор с указанным именем в системе отсутствует, либо только открыть доступ к уже существующему с указанным именем семафору.

Семафоры — это метод синхронизации, основанный на создании и управлении программным объектом системного типа sem_t, регулирующим количество нитей, осуществляющих одновременный доступ к некоторому ресурсу. Управление семафором выражается в его захвате и освобождении. Семафор ведёт счетчик захватов. Начальное значение счётчика захватов семафора устанавливается при его создании. Если нить пытается захватить семафор, счётчик которого не больше 0, то она блокируется до момента, когда значение счетчика не станет больше 0. Семафоры бывают неименованные и именованные.

Неименованные семафоры:

- Эти семафоры создаются внутри процесса и не имеют имени в файловой системе.
- Обычно используются для синхронизации между нитями внутри одного процесса.
- Создаются с помощью функции sem_init() и уничтожаются с помощью функции sem_destroy().
- Не подходят для синхронизации между разными процессами или на удалённых узлах сети.

Именованные семафоры:

- Эти семафоры имеют уникальное имя в файловой системе и могут использоваться для синхронизации между процессами.

- Создаётся как файл особого типа, регистрируемый в файловой системе в каталоге /dev/sem
- Создаются и уничтожаются с помощью функций sem_open() и sem_unlink().
- Могут быть использованы для синхронизации между разными процессами, в том числе на удалённых узлах сети.

Если аргумент pshared отличен от нуля, то семафор может разделяться нитями различных процессов, при условии, что семафор создан в разделяемой процессами памяти. С помощью аргумента value задаётся начальное значение счётчика семафора. После инициализации семафора указатель sem используется в функциях управления семафором. Для аннулирования неименованного семафора используется функция int sem_destroy(sem_t *sem);

53. Как изменяется значение счётчика семафора в результате успешного выполнения функции sem_post()?

Функция sem_post() увеличивает счётчик семафора sem на 1.

Если имеются нити, которые в настоящее время блокированы, ожидая семафор, то одна из этих нитей возвратится успешно из вызова sem_wait. Нить, которая будет разблокирована первой, определяется в соответствии с приоритетом и временем ожидания (с наибольшим приоритетом, которая ждала дольше всех). Функция sem_post() может быть вызвана обработчиком сигналов.

54. Как изменяется значение счётчика семафора в результате успешного выполнения функции sem_wait()? (51?)

Функция sem_wait() уменьшает значение счётчика семафора на 1.

Если при этом значение семафора не большее чем ноль, то вызвавшая функцию нить блокируется до тех пор, пока не возникнет возможность уменьшить счётчик или запрос не будет завершен в связи с приходом сигнала. Предполагается, что в какой-то момент времени некоторая нить, вызовет функцию sem_post(), чтобы увеличить счетчик семафора.

55. Какой результат ожидает процесс при выполнении функции shm_open().

Функция shm_open() создаёт и/или присоединяет к процессу именованную память с заданным именем name и возвращает дескриптор именованной памяти.

При успешном выполнении функция возвращает неотрицательное целое положительное число, которое является дескриптором именованной памяти. Если при этом произошло создание новой именованной памяти, то в соответствующем каталоге появиться файл с именем памяти. В случае ошибки - значение (-1) и заносит код ошибки в системную переменную errno.

56. Какой результат ожидает процесс при выполнении функции mmap().

Функция возвращает начальный адрес отображения именованной памяти в локальной памяти процесса или MAP_FAILED в случае ошибки. Код ошибки помещается в errno.

Открытие процессом именованной памяти не открывает процессу непосредственного доступа к ней, а только под управлением ядра операционной системы. Для этого процессу необходимо отобразить требуемую ему область именованной памяти нужного размера в адресное пространство процесса. Через это собственное адресное пространство процесс и получает возможность доступа к открытой именованной памяти. Отображение осуществляется с помощью функции mmap()

57. Если сигнал генерируется функцией kill(pid, sig), где pid=0, то кто является адресатом этого сигнала?

Когда сигнал генерируется функцией kill(pid, sig), где pid = 0, он адресуется всем процессам, входящим в группу, которой принадлежит процесс, пославший сигнал.

Если pid>0, то адресуется единственный процесс.

Если pid=0, то сигнал посылается всем процессам, входящим в группу (идентификатор группы процессов которых равен идентификатору отправителя), которой принадлежит пославший сигнал процесс.

Если pid<0, то сигнал посылается каждому процессу, являющемуся членом группы процессов с GID=-pid.(идентификатор группы процессов которых равен абсолютному значению pid)

Если sig равен 0, то сигнал не посылается, а проверяется возможность послать сигнал по указанному pid (проверка наличия адресата).

58. Если для посылки сигнала процесс использует функцию SignalKill(), то кто может быть адресатом сигнала?

Функция SignalKill позволяет послать сигнал группе процессов, процессу или нити. Сочетание значений pid и tid определяют, кому адресуется сигнал.

int SignalKill(uint32_t nd, pid_t pid, int tid, int signo, int code, int value);

Функция SignalKill позволяет послать сигнал группе процессов, процессу или нити. Сочетание значений pid и tid определяют, кому адресуется сигнал:

pid=0, tid = - , // адресат: Группа процессов, которой принадлежит процесс, пославший сигнал
pid<0, tid = - , // адресат: Группа процессов (GID = -pid)
pid>0, tid = 0 , // адресат: Процесс (ID процесса равен pid)
pid>0, tid > 0 , // адресат: Нить в процессе (ID нити равен tid, ID процесса равен pid)

Вызов не является блокирующим. Функция SignalKill() в случае ошибки возвращает –1, и устанавливает значение errno. Любое другое значение говорит об успешном завершении. Успешное завершение функции означает, что сигнал доставлен адресату. Если аргументу signo присваивается значение 0, то сигнал не посылается, но таким способом можно проверить актуальность адресованных в вызове процесса и нити.

59. Условия адекватного применения процессом функции SignalWaitinfo() для работы с сигналами.

1) Инициации одного или более сигналов из указанного набора оказываются задержанными маской блокирования
2) Когда ей в этом состоянии доставляется инициация сигнала, не блокированного маской

Помимо асинхронной реакции на приход сигнала в ряде случаев нити может потребоваться явно планировать обработку сигналов, предварительно задерживая их, устанавливая маску блокирования. Для этого может использоваться функция:

int SignalWaitinfo(const sigset_t* set, siginfo_t* info);

При выполнении функции нить блокируется в состоянии ожидания прихода сигнала (состояние блокирования STATE_SIGWAITINFO), если в указанном наборе сигналов, на который указывает set, нет ни одного задержанного сигнала. Нить выходит из этого состояния блокирования и функция SignalWaitinfo() завершается в двух случаях. Во-первых, когда инициации одного или более сигналов из указанного набора оказываются задержанными маской блокирования. Во-вторых, когда ей в этом состоянии доставляется инициация сигнала, не блокированного маской. В первом случае функция SignalWaitinfo() извлекает задержанный сигнал из набора сигналов типа sigset_t, на который указывает set, возвращает номер извлечённого сигнала и сохраняет информацию, полученную с извлечённым сигналом, в структуре siginfo_t, на которую указывает аргумент info. Во втором случае реализуется 166 диспозиция сигнала, после выполнения которой SignalWaitinfo() завершается с ошибкой, возвращает -1 и устанавливает в errno код ошибки EINTR.

Аргумент info может принимать значение NULL, если кроме номера сигнала другая информация с инициацией сигнала не важна или не передается.

60. Что необходимо выполнить удалённому процессу-клиенту с результатом выполнения функции nd=netmgr_strtond("/net/comp",NULL), используемого для организации соединения с процессом-сервером на удалённом узле с именем comp?

Использовать функцию ConnectAttach() для установления соединения с каналом процесса на удалённом узле. Первым параметром этой функции будет дескриптор удалённого узла nd.

Важно отметить, что дескриптор удалённого узла nd является локальным для текущего узла и будет действительным только для текущего соединения. Этот дескриптор не является уникальным или постоянным для всей сети, и его актуальность может быть утеряна при отсоединении узла от сети или при других изменениях в сети. Поэтому, важно использовать его сразу после получения для установления соединения с удалённым узлом.

61. Какие и чьи права доступа будет иметь файл, созданный при выполнении функции open("file.dat", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR)?

Владелец файла (пользователь, запустивший программу): чтение (R) и запись (W).

O_WRONLY: Файл будет открыт только для записи.

O_CREAT: Если файл не существует, он будет создан.

O_TRUNC: Если файл существует, его содержимое будет усечено (обрезано) до пустого.

S_IRUSR | S_IWUSR: Это аргумент, определяющий права доступа для владельца файла. S_IRUSR устанавливает права на чтение, а S_IWUSR устанавливает права на запись для владельца файла.

62. Как завершится выполнение функции open("file.dat", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR); если в текущем каталоге существует файл с именем file.dat?

Если файл "file.dat" уже существует, он будет открыт для записи, и его содержимое будет усечено до пустого. Файл сохранит тот же набор прав доступа, что и был у него до этого (переданное значение S_IRUSR | S_IWUSR в данном случае не повлияет на уже существующий файл).

O_WRONLY: Файл будет открыт только для записи.
O_CREAT: Если файл не существует, он будет создан.
O_TRUNC: Если файл существует, его содержимое будет усечено (обрезано) до пустого.
Если файл "file.dat" не существует, то он будет создан с правами доступа S_IRUSR | S_IWUSR и открыт для записи.

63. Что означает для нити, запущенной в процессе, свойство обособленности?

Если нить создается как синхронизирующая (PTHREAD_CREATE_JOINABLE), то у других нитей появляется возможность блокироваться и ждать ее завершения. Если нить создается как обособленная (PTHREAD_CREATE_DETACHED), то у других нитей нет возможности блокироваться и ждать ее завершения.

64. Если завершился квант процессорного времени текущей нити с приоритетом pr и дисциплиной диспетчеризации RR, но выполнение нити ещё не завершилось, то ожидающей нити с каким приоритетом будет предоставлен очередной квант процессорного времени?

Следующей нити будет предоставлен очередной квант процессорного времени с тем же приоритетом pr.

Таким образом, если текущая нить завершила свой квант времени, но ещё не завершила выполнение, ей будет предоставлен следующий квант времени с тем же приоритетом, и она продолжит конкурировать с другими нитями в системе за выполнение.

Базовые дисциплины диспетчеризации:

- FIFO (First In First Out) – нить, ставшая активной, выполняется до тех пор, пока не завершит свою работу, не будет вытеснена более приоритетной нитью или не перейдет в блокированное состояние.
- RR (Round Robin) – карусельная диспетчеризация, при которой продолжительность нахождения нити в активном состоянии ограничивается так называемым квантом времени выполнения (time slice), после истечения которого нить вытесняется и вновь поступает в очередь готовых к выполнению нитей, а первая готовая нить становится активной и выполняется в соответствии с её дисциплиной диспетчеризации.

Квант времени дисциплины RR является задаваемым системным параметром

65. Что означает механизм наследования приоритетов для борьбы с инверсией приоритетов?

Механизм наследования приоритетов в контексте борьбы с инверсией приоритетов является техникой, используемой в системах реального времени для предотвращения ситуаций, когда низкоприоритетные процессы блокируют более высокоприоритетные процессы, вызывая нежелательные задержки в выполнении последних.

Механизм наследования приоритетов решает эту проблему путем наследования сервером уровня приоритета клиентской нити во время обработки её запроса. Это означает, что, если во время обработки запроса от нити появляется высокоприоритетная нить, готовая к выполнению, серверная нить будет иметь низкий приоритет, и, следовательно, высокоприоритетная нить сможет вытеснить серверную нить и стать активной.

Инверсия приоритетов нитей (Priority Inversion) — это ситуация, когда нить с более низким приоритетом временно получает преимущество выполнения перед нитью с более высоким приоритетом, в то время как готовые к выполнению нити с более высоким приоритетом оказываются заблокированными.

66. Какой атрибут устанавливается процессом в атрибутной записи запускаемой нити, чтобы она была обособленной нитью?

В атрибутной записи запускаемой нити атрибут, который устанавливается для того, чтобы нить была обособленной, называется detachstate. Для установки этого атрибута используется функция pthread_attr_setdetachstate().

67. Что произойдёт с нитью, если она по отношению к некоторой блокировке чтения/записи выполнит функцию pthread_rwlock_trywrlock() или pthread_rwlock_tryrdlock(), а блокировка чтения/записи уже захвачена?

Если функция pthread_rwlock_trywrlock() вызвана для захвата записи и блокировка уже захвачена другой нитью в режиме чтения или записи, то нить, выполняющая попытку, получит значение EBUSY. Это означает, что блокировка уже используется, и нить не сможет её захватить. В этом случае нить может принять решение, как обработать отказ в захвате, например, повторить попытку позже или выбрать другую логику выполнения.

При этом необходимо контролировать значения, возвращаемые функциями pthread_rwlock_trywrlock() и pthread_rwlock_tryrdlock(). Если блокировка чтения/записи свободна, то она будет захвачена нитью. В противном случае - нить получит отказ захвата и продолжит своё выполнение. В этом случае для анализа ситуации нити необходимо контролировать возвращаемое функциями значение:

- EOK – успешный захват блокировки чтения/записи;
- EBUSY – отказ, блокировка чтения/записи ранее уже захвачена.

68. Какой программный объект создаётся функцией:
int shm_open(const char *name, int oflag, mode_t mode)?

Функцией int shm_open(const char *name, int oflag, mode_t mode) создается или открывается именованная память (shared memory).

Именованная память (Shared Memory) в операционных системах представляет собой механизм, позволяющий нескольким процессам разделять общую область памяти, доступную по имени в файловой системе. Создаваемая именованная память регистрируется как файл устройства, и процессы могут присоединяться к ней, обмениваться данными и иметь доступ к общей памяти даже после завершения работы. Этот механизм обеспечивает эффективный способ обмена информацией между параллельно работающими процессами, но требует внимательной синхронизации для избежания конфликтов при одновременном доступе к данным из разных процессов.

69. Выполнение какой функции приводит к уменьшению на 1 значения счётчика семафора?

Выполнение функции sem_wait() приводит к уменьшению на 1 значения счётчика семафора.

Если при этом значение семафора не большее чем ноль, то вызвавшая функцию нить блокируется до тех пор, пока не возникнет возможность уменьшить счётчик или запрос не будет завершен в связи с приходом сигнала. Предполагается, что в какой-то момент времени некоторая нить, вызовет функцию sem_post(), чтобы увеличить счетчик семафора.

70. Для чего процесс применяет функцию mmap()?

Функция **mmap**() используется **для отображения именованной памяти в адресное пространство процесса.**

Это позволяет процессу получить доступ к области именованной памяти, управляемой ядром операционной системы.

Является методом ввода-вывода через отображение файла на память и естественным образом реализует выделение страниц по запросу, поскольку изначально содержимое файла не читается с диска и не использует физическую память вообще. Реальное считывание с диска производится в «ленивом» режиме, то есть при осуществлении доступа к определённому месту.

void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t off);

addr: Начальный адрес области памяти процесса, с которой будет ассоциирована отображаемая область. Обычно указывается как NULL, и ядро само выбирает адрес.

len: Длина в байтах отображаемой области.

prot: Порядок использования именованной памяти, задаваемый флагами, такими как PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE и т.д.

flags: Флаги, определяющие возможность разделения отображаемой области с другими процессами, например, MAP_SHARED или MAP_PRIVATE.

fd: Дескриптор открытого файла, представляющего именованную память.

off: Смещение от начала именованной памяти до начала отображаемой области.

Пример кода, который использует **mmap**():

```
fd = shm_open("/common", O_RDWR, 0777);

addr = mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
```

71. **Если сигнал генерируется функцией kill(pid, sig), где pid<0, то кому адресуется сигнал?**

Когда сигнал генерируется функцией kill(pid, sig), где pid < 0, он адресуется каждому процессу, являющемуся членом группы процессов с GID, равным абсолютному значению pid.

Если pid>0, то адресуется единственный процесс.

Если pid=0, то сигнал посылается всем процессам, входящим в группу (идентификатор группы процессов которых равен идентификатору отправителя), которой принадлежит пославший сигнал процесс.

Если pid<0, то сигнал посылается каждому процессу, являющемуся членом группы процессов с GID=-pid.(идентификатор группы процессов которых равен абсолютному значению pid)

Если sig равен 0, то сигнал не посылается, а проверяется возможность послать сигнал по указанному pid (проверка наличия адресата).

72. **Какая из функций kill() или SignalKill() позволяют послать сигнал, адресуемый конкретной нити?**

Функция SignalKill() позволяет послать сигнал, адресованный конкретной нити, так как она имеет аргумент tid, который задает значение ID нити, которой посылается сигнал. В частности, если tid > 0, то сигнал адресуется конкретной нити в процессе, идентификатор которой равен tid, а идентификатор процесса равен pid.

73. **Если пришёл сигнал, адресованный нити, которая замаскировала этот сигнал, то что будет с этим сигналом?**

Если нить замаскировала определенный сигнал с использованием функции и пришел сигнал, адресованный этой нити, то сигнал будет задержан и будет ожидать разблокировки маски сигналов.

При использовании функции SignalWaitinfo(), нить блокируется в состоянии ожидания прихода сигнала, и сигналы, которые были замаскированы, будут задержаны до тех пор, пока маска блокировки не будет изменена для разблокировки этих сигналов. Когда сигнал будет разблокирован, нить выйдет из состояния блокирования и начнет обработку сигнала.

Если нить не разблокирует маску сигналов, то сигнал останется в задержанном состоянии, и функция SignalWaitinfo() будет ожидать, пока нить не изменит маску, чтобы разблокировать этот сигнал.

74. **Если для управления доступом нитей к некоторому разделяемому программному ресурсу создан мутекс, что предпримет ядро по отношению к нити, которая попытается осуществить доступ к этому ресурсу, не захватывая мутекс?**

Если для управления доступом нитей к разделяемому программному ресурсу создан мутекс, то ядро будет блокировать любую нить, которая пытается осуществить доступ к этому ресурсу без захвата мутекса. Если нить пытается захватить мутекс, который уже был захвачен другой нитью, она блокируется ядром до момента освобождения мутекса нитью, его захватившей.

75. **Как в ОСРВ QNX6 задаётся абсолютное значение реального времени для таймеров?**

Для задания значения как абсолютного, так и интервального времени используется одна и та же системная структура данных timespec (определена в time.h):

struct timespec{time_t tv_sec; //секунды
uint64_t tv_nsec; //наносекунды, 1сек=10^9 нс };

В ОСРВ QNX6 абсолютное значение реального времени задаётся в секундах с начала эпохи, установленной на 00 часов 00 минут 00 секунд 1 января 1970 года. Внутренний формат представления системного времени в QNX позволяет выражать время в секундах (абсолютное время) и имеет тип uint64_t. Этот формат рассчитан на использование до января 2554 года.

76. Как в ОСРВ QNX6 задаётся значение интервала реального времени?

В QNX Neutrino RTOS (QNX6) для задания значения интервала реального времени используется структура данных struct timespec - временной интервал в секундах и наносекундах.

Этот временной интервал может быть использован, например, при вызове функций планирования времени в ядре операционной системы, для управления задержками или периодическими операциями.

77. Какой тип уведомления для таймера формирует вызов SIGEV_PULSE_INIT()?

Вызов SIGEV_PULSE_INIT() формирует уведомление типа "послать импульс" (PULSE).

Чтобы создаваемый с помощью функции timer_create() таймер настроить на посылку уведомлений-импульсов, необходимо установить полю sigev_notify структуры event значение SIGEV_PULSE и дополнительно задать значения ряду соответствующих полей.

SIGEV_PULSE_INIT(struct sigevent *event, //структура, значение которой задаёт тип уведомления и его параметры
int coid, // ID соединения (связи) с каналом, по которому уведомляющий импульс будет посылаться. Если процесс планирует для себя уведомление импульсом, то ему необходимо создать канал и ID соединения с собственным каналом использовать в качестве coid в макрокоманде.
short priority, // приоритет, связываемый с импульсом, который будет наследоваться нитью, принявшей импульс. Нулевое значение не допускается. Если нет необходимости в изменении приоритета принимающей импульс нити, то для priority следует установить специальное значение SIGEV_PULSE_PRIO_INHERIT.
short code, // код импульса
union sigval value) // 32-битное значение импульса (типа int или void*)

78. Какой тип уведомления формирует для таймера вызов SIGEV_SIGNAL_INIT()?

Вызов SIGEV_SIGNAL_INIT() формирует тип уведомления таймера "послать сигнал", при котором таймер посылает сигнал процессу. В этом случае, посылается простой сигнал без какой-либо дополнительной информации, адресованный процессу.

SIGEV_SIGNAL_INIT(struct sigevent *event, //структура, значение которой задаёт тип уведомления и его параметры
int signo) // номер сигнала, который должен быть в диапазоне от 1 до NSIG-1

79. Какой тип уведомления формирует для таймера вызов SIGEV_SIGNAL_CODE_INIT()?

Вызов SIGEV_SIGNAL_CODE_INIT() формирует тип уведомления таймера "послать сигнал" с дополнительной информацией. В этом случае, при срабатывании таймера, посылается сигнал адресованный процессу, и к этому сигналу прикрепляются дополнительные данные, такие как значения sigev_code и sigev_value.

Если при посылке сигнала необходимо передать дополнительную информацию в виде значений sigev_code и sigev_value, то для формирования уведомления надо использовать макрокоманду:

SIGEV_SIGNAL_CODE_INIT(struct sigevent *event, //структура, значение которой задаёт тип уведомления и его параметры
int signo, // номер сигнала, который должен быть в диапазоне от 1 до NSIG-1
void *value, // 32-битное значение, предназначенное обработчику сигнала.
short code) // код, который должен быть в диапазоне от SI_MINAVAIL до SI_MAXAVAIL и предназначен для интерпретации обработчиком сигнала.

80. Какой тип уведомления таймера формирует вызов SIGEV_SIGNAL_THREAD_INIT()?

Вызов SIGEV_SIGNAL_THREAD_INIT() формирует тип уведомления таймера "послать сигнал", который предназначен для доставки сигнала конкретной нити, которая запланировала таймер с помощью функции timer_settime().

Использование SIGEV_SIGNAL_THREAD_INIT() позволяет запланировать посылку сигнала, адресуемого конкретной нити. В этом случае сигнал доставляется той нити, которая его запланировала с помощью функции timer_settime().

В этом случае нить должна установить обработчик сигналов (асинхронный приём сигналов) или ожидать прихода сигнала, используя функцию SignalWaitinfo() или sigwait(). Чтобы таймер настроить на посылку сигналов, необходимо установить полю sigev_notify структуры event одно из значений SIGEV_SIGNAL, SIGEV_SIGNAL_CODE или SIGEV_SIGNAL_THREAD. Использование первых двух значений планирует уведомление сигналом, адресуемым процессу. Использование третьего значения позволяет запланировать посылку сигнала, адресуемого конкретной нити.

81. Какой тип уведомления таймера формирует вызов SIGEV_THREAD_INIT()?

Вызов SIGEV_THREAD_INIT() формирует тип уведомления таймера "создать нить". В этом случае, при срабатывании таймера, создается новая нить, которая выполняет указанную функцию, переданную через аргумент void (*fn)(void * arg).

Этот тип уведомления предполагает, что в результате срабатывания таймера в процессе создаётся новая нить. Для задания уведомления полю sigev_notify структуры event необходимо установить значение SIGEV_THREAD.

SIGEV_THREAD_INIT(struct sigevent *event, //структура, значение которой задаёт тип уведомления и его параметры
void (*fn)(void * arg) // указатель на функцию, которую нужно запустить как нить
void *arg, // указатель на атрибутную запись нити, он должен указывать на структуру, которая будет использована функцией pthread_attr_init(), или быть NULL для значений атрибутов по умолчанию.
pthread_attr *attributes) // значение, которое передаётся функции, запускаемой как нить.

82. Какой режим срабатывания таймера планируется вызовом timer_settime(), если поля переменной struct itimerspec timer равны:
timer.it_value.tv_sec=3600;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=0;
timer.it_interval.tv_nsec=0;
?

Относительный однократный - сработает 1 раз через 3600 секунд

Виды таймеров:

- Абсолютный – точность до секунды
- Относительный – точность до долей секунды
- Однократный – посылает уведомление только 1 раз
- Периодический – посылает уведомление по истечении текущего времени и автоматически планирует очередной временной интервал

83. Как сработает таймер, запланированный вызовом timer_settime(), если поля переменной struct itimerspec timer равны:
timer.it_value.tv_sec=0;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=1;
timer.it_interval.tv_nsec=500000000;
?

Таймер не сработает. Установка параметра it_value в ноль выключает таймер

Виды таймеров:

- Абсолютный – точность до секунды
- Относительный – точность до долей секунды
- Однократный – посылает уведомление только 1 раз
- Периодический – посылает уведомление по истечении текущего времени и автоматически планирует очередной временной интервал

84. Поля переменной struct itimerspec timer имеют значения:
timer.it_value.tv_sec=1;
timer.it_value.tv_nsec=0;
timer.it_interval.tv_sec=1;
timer.it_interval.tv_nsec=0.
Какой тип и режим срабатывания таймера имеет смысл с такими полями системного времени?

Относительный периодический таймер.

Виды таймеров:

- Абсолютный – точность до секунды
- Относительный – точность до долей секунды
- Однократный – посылает уведомление только 1 раз
- Периодический – посылает уведомление по истечении текущего времени и автоматически планирует очередной временной интервал

85. Какую роль играет функция Timer_Timeout()?

Установить таймаут для заблокированной нити перед выполнением блокирующего запроса к ядру.

Функция TimerTimeout() использует механизм таймаута ядра в операционной системе QNX.

Эта функция предоставляет нити возможность временно отказаться от выполнения блокирующего запроса и продолжить свое выполнение после истечения установленного таймаута. По прошествии указанного времени, ядро выводит нить из блокированного состояния и отправляет ей уведомление типа SIGEV_UNBLOCK.

int TimerTimeout(clockid_t clock_id, // Задает тип часов реального времени (например, CLOCK_REALTIME).
int flags, // Специфицирует соответствующее блокирующее состояние, связанное с запросом к ядру.
const struct sigevent *event, // Указывает тип уведомления, который будет использоваться для сигнализации о завершении таймаута. В данном случае, должно быть установлено в SIGEV_UNBLOCK (уведомление типа "разблокировка").
const uint64_t *timout, // Указывает относительное время в наносекундах, после которого ядро должно послать нити уведомление об истекшем таймауте и вывести нить из блокированного состояния. Если установить равным NULL, то блокирование вообще не допустимо.
uint64_t *oldtimout); // Сохраняет предыдущее значение таймаута. Если нет необходимости в этой информации, то может быть установлено в NULL.

86. Какова реакция на тайм-аут ядра, установленного клиентом для REPLY-блокированного состояния при передаче сообщения, если при создании канала сервером был установлен флаг _NTO_CHF_UNBLOCK?

Если флаг _NTO_CHF_UNBLOCK установлен, то клиент остается заблокированным в REPLY-блокированном состоянии после истечения таймаута. В этом случае ядро посылает серверу уведомление в виде импульса об истечении таймаута ожидания ответа от клиента. После получения импульса сервер берет на себя ответственность за дальнейшую задержку ответа клиенту в REPLY-блокированном состоянии

Реакция на тайм-аут ядра при установленном флаге _NTO_CHF_UNBLOCK зависит от того, в каком состоянии (SEND-блокированное или REPLY-блокированное) находится клиент.

Если тайм-аут происходит в REPLY-блокированном состоянии, то поведение зависит от установки флага _NTO_CHF_UNBLOCK при создании канала сервером:

1) Если флаг _NTO_CHF_UNBLOCK не установлен, то клиент немедленно разблокируется при истечении таймаута для REPLY-блокированного состояния. В этом случае сервер не получает оповещения о том, что клиент разблокирован, и может продолжить выполнение функции MsgReply(), хотя клиент уже не ждёт ответа.
2) Если флаг _NTO_CHF_UNBLOCK установлен, то клиент остается заблокированным в REPLY-блокированном состоянии после истечения таймаута. В этом случае ядро посылает серверу уведомление в виде импульса об истечении таймаута ожидания ответа от клиента. После получения импульса сервер берет на себя ответственность за дальнейшую задержку ответа клиенту в REPLY-блокированном состоянии.

Если таймаут истекает в SEND-блокированном состоянии, то функция MsgSend() завершается, возвращая клиенту признак ETIMEDOUT. Так как сервер ещё не выполнил функцию MsgReceive(), то он практически не замечает, что клиентом осуществлялась попытка послать сообщение.