РЕФЕРАТ

Расчетно-пояснительная записка N страниц, N рисунков, N таблиц, N источников, 1 приложение.

Объектом исследования является операционная система «Нейтрино-Э».

Цель работы – исследование инструментов управления потоками в операционной системе «Нейтрино-Э». Изучить основные функции управления потоками и методы организации работы потоков, и выбрать подходящие для разрабатываемой программной подсистемы.

В результате работы был проведен анализ возможностей управления и способов организации потоков, выявлены преимущества и недостатки, исходя из которых была построена структура управления потоками в разрабатываемой программной подсистеме.

Содержание

ОБОЗНЧЕНИЯ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

ПП ДП – программная подсистема «Диспетчер потоков».

ПО – программное обеспечение.

БЦВМ – бортовая цифровая вычислительная машина.

ЗОСРВ – защищенная операционная система реального времени.

??? ТКЗ – таймер контроля задачи.

??? ПТКЗ – поток таймера контроля задачи.

??? ПЗ – поток задач (циклических).

??? ИТД – интервальный таймер диспетчера.

Интервальный таймер – это таймер настраиваемый и запускаемый в программе.

Поток – это исполняемый элемент внутри процесса, предусмотренный планировщиком операционной системы.

Семафор – это объект, который используется для контроля доступа нескольких потоков до общего ресурса.

Мьютекс – механизм, обеспечивающий взаимное исключение доступа к критической области.

Кондвар -

IPC (Inter Process Communications) – компьютерная технология, осуществляющая обмен данными между потоками одного и/или разных процессов.

Введение

На данный момент существует проблема покупки комплектующих деталей и поддержка программного обеспечения для вычислительных машин, так как поставки от многих зарубежных производителей прекращены. АО «РПКБ» и другие научно-технические российские компании нуждаются в отечественных аналогах, чтобы продолжить незавершенные проекты и начать разработку новых.

В России самая мощная и надежная процессорная архитектура считается «Эльбрус». Процессоры с этой архитектурой активно применяются в технической отрасли. Поэтому компания по разработке программного обеспечения в Санкт-Петербурге «СВД Встраиваемые Системы» выпустила ЗОСРВ «Нейтрино-Э» – операционную систему под архитектуру Эльбрус. Разрабатываемая программная подсистема «Диспетчер потоков» будет управлять циклическими задачами БЦВМ, процессор которой будет архитектуры Эльбрус с операционной системой Нейтрино-Э.(?? грамотно ли построено предложение?)

Исследовательская работа посвящена выявлению особенностей операционной системы Нейтрино-Э и рассмотрению возможностей управления потоками в ней. Также в работе определяется какие именно из рассмотренных возможностей могут быть применены в ПП ДП.

1. Выбор операционной системы для разрабатываемой программной подсистемы

На данный момент существует множество операционных систем, которые имеют свои преимущества, но необходимо учесть требования надежности и актуальности АО «РПКБ» при выборе ОС и разработке программного продукта.

«Нейтрино» – это защищенная операционная система реального времени (ЗОСРВ), предназначенная для предсказуемого и отказоустойчивого управления ресурсами многомашинных и многопроцессорных вычислительных комплексов в режиме реального времени [1].

ЗОСРВ «Нейтрино» поддерживает процессоры мировых производителей с архитектурами Intel x86, ARM, MIPS и PowerPC, но компания-производитель «СВД Встраиваемые Системы» обеспечила поддержку отечественных аналогов таких как процессоры «Эльбрус» и выпустила программный продукт ЗОСРВ «Нейтрино-Э» [1].

Свойства и особенности «Нейтрино-Э»:

- Архитектура на основе микроядра. При такой архитектуре процессы отделены от ядра, и если в процессе возникают ошибки, то это не окажет влияние на ядро.
- Поддержка мультипроцессорности. Данная ОС способна распределять задачи между процессорами, что позволяет достичь более эффективного использования ресурсов вычислительной машины.
- Улучшенный планировщик, в котором осуществляется распределение наследования приоритетов и защита от инверсии приоритетов, а также выбор между динамическим назначением процессора для выполнения потоков и привязкой потоков к конкретным процессорам или процессорным ядрам.

- Динамически обновляемые приложения, системные сервисы и драйверы, изоляция сбоев и автоматическое восстановление системы.
- Производительность и предсказуемость жесткого реального времени, что гарантирует актуальность результата.

АО «РПКБ» в настоящее время нуждается в импортозамещении деталей, используемых для БЦВМ, и возникает необходимость создания ПО на базе ОС, поддерживающих российские процессорные архитектуры. Поэтому для реализации программной подсистемы «Диспетчер потоков» целесообразнее выбрать ОС «Нейтрино-Э», которая удовлетворяет требованиям информационной безопасности, надежности и быстродействию, а также обладает рядом плюсов в диспетчеризации потоков.

2. Анализ возможностей управления потоками

2.1 Диспетчеризация потоков

Поток – это исполняемый элемент внутри процесса, предусмотренный планировщиком операционной системы. Каждый поток является индивидуально запланированным действием, позволяющим каждому процессу выполнять более одного действия в каждый момент времени [2].

Потоки имеют два основных состояния: «заблокирован» или «работоспособен». Заблокированный поток ожидает события, которое приведет поток в работоспособное состояние. Этим событием будет один из вариантов: запрос или ответ от IPC, открытие мьютекса или семафора. Работоспособное состояние позволяет потоку использовать центральный процессор. Есть несколько вариантов такого состояния: когда поток запущен, то есть использует процессор, и когда поток готов к работе, то есть имеет возможность использовать процессор, но ожидает завершения работы другого запущенного потока.

Для готовых потоков имеет значение приоритет, диапазон которого от 0 (наименьший) до 255 (наибольший). Ядро всегда выбирает готовый поток с наибольшим приоритетом, чтобы он был единственным, кто использует центральный процессор.

Можно выделить основные этапы работы с потоками:

1) создание потоков;
2) синхронизация;
3) соединение;
4) завершение потоков.

Рассмотрим подробнее и проанализируем все этапы, их функции и объекты, и определим роль в процессе управления потоками.

2.2 Создание потоков

Вначале нужно создать поток и установить параметры потока. Для создания потока необходимо вызвать специальную функцию: pthread_create (pthread_t *tid, pthread_attr_t *attr, void *(func) (void *), void *arg); где tid – ID потока, attr – настройка атрибутов потока, func – функция потока, arg – параметры для функции потока. Если остальные параметры функции могут быть NULL, то func всегда должна быть определена.

Чтобы изменить атрибуты потоков, для начала необходимо инициализировать объект pthread_attr_t с помощью функции pthread_attr_init(). Далее необходимо установить параметры с помощью функций: pthread_attr_setdeachstate(), pthread_attr_setschedparam(), pthread_attr_setschedpolicy(), pthread_attr_setstackaddr() и другие. При завершении работы необходимо обязательна очистка: pthread_attr_destroy().

Если указать NULL, в параметре attr то атрибутам будут присвоены значения по умолчанию. Атрибуты потока позволяют программам изменять множество свойств потока, таких как приоритет, размер стека, параметры планирования, начальный отдельный статус и другое [2].

Таким образом, если не изменять значения атрибутов, а оставить их по умолчанию, то у потоков будет одинаковый приоритет, соответственно они все будут равные в очереди, что приведет к хаотичному выбору готового потока для разрешения доступа к центральному процессору. В такой ситуации будет необходим другой способ сохранения порядка в очереди.

Итак, функция создания является неотъемлемой частью использования потоков в программе, особенно важно указать функцию, которую поток будет выполнять, иначе нет смысла его создавать. Изменение атрибутов потока позволяет настроить параметры для работы потока под конкретные условия задачи, но если нет необходимости, то при создании устанавливаются значения по умолчанию, что очень удобно.

2.3 Синхронизация потоков

В работе с потоками можно столкнуться с проблемой доступа к общей области памяти. Например, оба потока могут вносить изменения одновременно, так может появиться потеря информации или неточные данные. Для недопущения подобных ошибок есть особые инструменты синхронизации потоков. Среди них часто используемые это: семафоры, мьютексы и кондвары. Остальные используются крайне редко и в частных случаях.

Объект, который используется для контроля доступа нескольких потоков до общего ресурса, называется семафором. Семафор имеет два состояния: «открыт» (равно значению 1 счетчика семафора) и «закрыт» (равно значению 0). Его логика использования весьма простая: если семафор закрыт – доступ к ресурсу запрещен, а если семафор открыт – доступ к ресурсу разрешен.

Объект sem_t необходимо инициализировать с помощью функции sem_init(sem_t *sem, int pshared, unsigned int val), где sem – указатель на семафор, pshared – флаг расширения при использовании fork(), val – начальное значение семафора. Чтобы перевести ресурс в режим ожидания доступа необходимо зарыть семафор, то есть уменьшить его значение на единицу, для этого используется функция sem_wait(sem_t *sem). Чтобы разрешить доступ необходимо увеличить значение семафора, то есть открыть его с помощью sem_post(sem_t *sem). Перед завершением работы необходимо уничтожить семафор с помощью sem_destroy(sem_t *sem).

Стоит отметить, что семафор можно применять не только к ресурсу, но и к потоку. Таким образом, пока семафор закрыт – поток находится в заблокированном состоянии, когда семафор открыт – поток готов к работе.

Уязвимость семафора в том, что его значение может менять любой поток. Более надежным механизмом является мьютекс, который обеспечивает взаимоисключающую блокировку.

Мьютексы представляются объектом pthread_mutex_t. Как и большинство объектов в API P-потоков, это подразумевает их непрозрачную структуру, обеспечивающую разнообразие интерфейсов мьютексов. Хотя вы можете создавать мьютексы динамически, в большинстве случаев их использование статично. Определить и запустить мьютекс можно командой: pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER. Этот фрагмент кода определяет и инициализирует мьютекс под названием mutex. Это все, что мы должны сделать, чтобы начать его использовать.

Запирание, называемое также завладением, P-поточного мьютекса обеспечивается с помощью функции: pthread_mutex_lock(pthread_mutex_t *mutex). Успешный вызов этой функции заблокирует вызывающий поток, пока мьютекс, указанный как mutex, не станет доступным. После этого вызывающий поток активируется и эта функция вернет 0. Если мьютекс доступен в момент вызова, функция вернет значение немедленно. В случае ошибки функция возвращает один из следующих ненулевых кодов ошибки:

- EDEADLK — вызывающий поток уже владеет запрашиваемым мьютексом; этот код ошибки не обязательно возвратится по умолчанию; попытка завладеть уже имеющимся мьютексом может привести к взаимной блокировке;
- EINVAL — значение mutex недопустимо.

Вызывающие потоки, как правило, не проверяют возвращаемую величину, поскольку хорошо написанный код не должен генерировать каких-либо ошибок во время выполнения.

Противоположностью запиранию является отпирание, или высвобождение, мьютексов. Успешный вызов pthread_mutex_unlock

(pthread_mutex_t *mutex) высвобождает мьютекс, указанный как mutex, и возвращает 0. Вызов не блокируется и мьютекс освобождается немедленно. В случае ошибки функция возвращает ненулевой код ошибки, включающий:

- EINVAL — значение mutex недопустимо;
- EPERM — вызывающий процесс не владеет мьютексом, указанным как mutex; этот код ошибки не гарантируется; попытка высвободить мьютекс, которым вы не владеете, является ошибкой.

Кондвары

2.4 Соединение потоков

Учитывая, что потоки достаточно просто создаются и уничтожаются, должен быть и способ их синхронизировать вместо завершения других потоков — эквивалент wait() для поточности. Действительно, он существует. Это присоединение потоков.

Присоединение позволяет одному из потоков заблокироваться в ожидании завершения другого: pthread_join (pthread_t thread, void **retval); После успешного выполнения вызывающий поток блокируется до тех пор, пока поток, указанный как thread, не завершится (если thread уже завершен, pthread_join() возвращается немедленно). Как только thread завершается, вызывающий поток активизируется и, если retval не равен NULL, получает возвращаемое значение завершенного процесса, переданное pthread_exit() или возвращенное от его стартовой процедуры. После этого можно сказать, что потоки присоединились друг к другу.

Присоединение всегда позволяет потокам синхронизировать свое выполнение по отношению к периоду существования других потоков. Все потоки в P-потоках являются равноправными; каждый поток может присоединяться к любому другому. Один поток может присоединяться ко многим (фактически, как мы скоро увидим, чаще всего один главный поток ожидает других потоков, которые сам и создал), но только один поток может

пытаться присоединиться к определенному другому, несколько потоков не должны стараться присоединиться к какому-либо одному.

В случае ошибки pthread_join() возвращает один из следующих ненулевых кодов ошибок:

- EDEADLK — произошла взаимная блокировка — thread уже ожидает присоединения к вызывающему потоку или сам является вызывающим потоком;
- EINVAL — невозможно присоединить поток, определенный через thread (см. следующий раздел);
- ESRCH — значение thread недопустимо.

По умолчанию потоки создаются способными к присоединению. Однако они могут и отсоединяться, но в этом случае они станут в дальнейшем неприсоединяемыми. Поскольку до присоединения потоки потребляют какие-либо системные ресурсы, как делают это и процессы, пока их предки вызывают wait(), потоки, которые вы не планируете присоединять, должны быть отсоединены функцией: pthread_detach (pthread_t thread). В случае успеха pthread_detach() отсоединяет поток, указанный как thread, и возвращает 0. Результаты не определены, если вы вызываете pthread_detach() относительно потока, который уже отсоединен. В случае ошибки функция возвращает значение ESRCH, означающее, что значение thread недопустимо.

Для каждого потока в процессе необходимо вызвать pthread_join() или pthread_ detach(), чтобы системные ресурсы могли высвободиться после завершения потока (конечно, после того как завершается весь процесс, все поточные ресурсы высвобождаются, но присоединение или отсоединение всех процессов в явной форме остается хорошей практикой).

2.5 Завершение потоков

Естественной противоположностью созданию потоков является их завершение. Завершение потоков очень похоже на завершение процессов, за

исключением того, что, когда поток завершается, остальные потоки в процессе продолжают выполняться. В некоторых поточных шаблонах, таких как поток на соединение, потоки часто создаются и уничтожаются. Потоки могут прерываться при определенных обстоятельствах, которые имеют аналоги в завершении процессов:

- если поток возвращается из стартовой процедуры, он прерывается; это аналог «выхода за пределы» в main();
- если поток вызывает функцию pthread_exit() (будет рассмотрена далее), он завершается; это аналог вызова exit();
- если поток отменяется другим потоком через функцию pthread_cancel(), он завершается; это аналог отправки сигнала SIGKILL через kill().

В этих трех примерах завершается только поток, на который направлено действие. Все потоки в процессе завершаются, останавливая таким образом сам процесс, при следующих обстоятельствах:

- процесс возвращается из своей функции main();
- процесс завершается через exit();
- процесс выполняет новый двоичный образ через execve().

Сигналы могут убить процесс или отдельный поток в зависимости от того, как они направлены. P-потоки делают обработку сигналов несколько сложнее, поэтому лучше минимизировать использование сигналов в многопоточных программах.

Самый простой путь для потока, чтобы завершить самого себя, — это «выход за пределы» своей начальной процедуры. Однако часто нужно будет завершить поток где-то в глубине стека вызова функции, далеко от стартовой процедуры. Для таких случаев в P-потоках имеется вызов pthread_exit(void *retval), поточный эквивалент exit(). По выполнении вызывающий поток завершается; retval обеспечивается для каждого потока, ожидающего завершения. Ошибка не может произойти.

P-потоки вызывают завершение других потоков через их отмену. Это обеспечивает функция pthread_cancel (pthread_t thread). Успешный вызов посылает запрос на отмену потоку, представленному через идентификатор потока thread. Может ли поток быть отменен и когда, зависит от его состояния отмены и типа отмены соответственно. В случае успеха pthread_cancel() возвращает 0.

Стоит обратить внимание, что успех в данном случае означает лишь успешную обработку запроса на отмену. В действительности же завершение происходит асинхронно. В случае ошибки pthread_cancel() возвращает ESRCH, означающее, что значение thread недопустимо. Условия, при которых поток может быть отменен, не так просты. Состояние отмены потока может быть доступно или недоступно. По умолчанию оно является доступным для новых потоков. С другой стороны, тип отмены указывает, когда происходит отмена.

Потоки могут изменять свое состояние через pthread_setcancelstate (int state, int *oldstate). В случае успеха состояние отмены вызывающего потока устанавливается на state, а предыдущее состояние сохраняется в oldstate. Значением state может быть PTHREAD_CANCEL_ENABLE или PTHREAD_CANCEL_DISABLE для разрешения или запрещения отмены соответственно. В случае ошибки pthread_setcancelstate() возвращает EINVAL, что означает недопустимое значение state. Тип отмены потока может быть асинхронным или отложенным; по умолчанию обычно установлен последний.

С асинхронным типом отмены поток может быть убит в любой точке после получения команды на отмену. С отложенным типом поток может быть убит только в специальных точках отмены, которые являются функциями P-потоков или библиотеки C и представляют собой безопасные моменты, в которых вызывающий поток может быть прерван. Асинхронная отмена может быть полезна лишь в определенных ситуациях, так как она может оставить процесс в неопределенном состоянии. Чтобы программа вела себя корректно,

асинхронная отмена должна использоваться только потоками, для которых не предусмотрено совместное использование каких-либо ресурсов и возможен вызов только сигнально-безопасных функций. Потоки могут изменить свой тип через pthread_setcanceltype (int type, int *oldtype). В случае успеха статус отмены вызывающего потока устанавливается в type, а старый тип сохраняется в oldtype. Значением type может быть PTHREAD _CANCEL_ASYNCHRONOUS или PTHREAD_CANCEL_DEFERRED для установки асинхронной или отложенной отмены соответственно. В случае ошибки pthread_setcanceltype() возвращает EINVAL, что означает недопустимое значение type.

3. Применение возможностей в разрабатываемой программной подсистеме

Описание ПП ДП, возможно изображение диаграммы функционала, после чего можно сделать выводы какие функции и объекты можно использовать.

Этап	Функция или объект	Описание	Применение
Создание потоков	pthread_attr_t	Изменение стандартных значений атрибутов потока	Не будет использоваться в ПП ДП
	pthread_create();	Создание потока и присвоение ему ID, атрибутов и выполняемой функции	Будет использоваться в ПП ДП
Синхронизация			
Соединение			
Завершение потоков			

А вот устанавливать атрибуты потоков необязательно, потому что функция создания может установить значения по умолчанию и в ПП ДП можно организовать порядок выполнения функций потоков.

Заключение

Список используемых источников

1. СВД ЗОСРВ Нейтрино. Сайт: tadviser.ru
2. Лав Р. «Linux. Системное программирование», 2-е изд. — СПб.: Питер, 2014 г. — 448 с.
3.

Приложение А

Черновик технического задания ВКРБ