Компьютерные инструменты в образовании, 2018
№ 2: 25–30
УДК: 004.42
http://ipo.spb.ru/journal

инженерия программного обеспечения

ПАРАЛЛЕЛЬНЫЕ ПРОГРАММЫ В ПРОЕКТЕ РУСИ

Терехов А. Н. [1], Головань А. А.[1], Терехов М. А.[1]

[1]Санкт-Петербургский государственный университет, Санкт-Петербург, Россия

Аннотация

В статье описано расширение проекта РуСи, позволяющее использовать параллельные нити стандарта POSIX Threads. Это дало возможность существенно повысить эффективность использования многочисленных датчиков в популярных ныне системах Интернет вещей и в роботах, разрабатываемых на математико-механическом факультете СПбГУ.

Ключевые слова: *Параллельные нити, стандарт POSIX Threads, проект РуСи.*

Цитирование: Терехов А. Н., Головань А. А., Терехов М. А. Параллельные программы в проекте РуСи // Компьютерные инструменты в образовании. 2018. № 2. С. 25–30.

1. ВВЕДЕНИЕ

Даже школьников нужно учить параллельному программированию. Не будем произносить громких слов, что развитие вычислительной техники традиционными способами подошло к физическим пределам, поэтому дальнейшее ускорение вычислений возможно только за счет их распараллеливания, а это требует совершенно новых приемов и инструментов программирования. Приведем более простые и близкие нам примеры из робототехники [1].

Предположим, что робот должен получить информацию из нескольких десятков или даже сотен датчиков (сенсоров), что особенно актуально для популярного ныне понятия Интернет вещей, обработать эту информацию в реальном масштабе времени и принять оптимальное решение. Датчики выдают информацию в непредсказуемые моменты времени, поэтому, если опрашивать их последовательно (есть такой жаргонный, но многое хорошо объясняющий термин «жужжать»), будут большие потери времени. Существенно более эффективная реализация состоит в помещении каждого датчика в отдельный процесс; если информация появляется, процесс ее посылает в виде сообщения специального вида, в противном случае процесс ничего не делает, процессор не занимает ресурсов (программисты говорят «спит»), поэтому операционная система может отдать все ресурсы другим процессам.

В принципе, переключение процессов тоже занимает определенные ресурсы системы (во многих операционных системах весьма заметные), поэтому были придуманы упрощенные варианты процессов — нити. Не будем вдаваться в технические детали, но сразу скажем, что есть стандарт POSIX Threads [2], где все необходимые действия над нитями предусмотрены.

2. ПРИМЕНЕНИЕ СТАНДАРТА POSIX THREADS В ПРОЕКТЕ РУСИ

Один из авторов этой статьи (Александр Головань [3]) реализовал для проекта РуСи [4] в довольно общем виде библиотеку поддержки процедур стандарта POSIX Threads. Оказалось, что правила работы с ней довольно трудно объяснить начинающим программистам, в основном, из-за большого количества указателей на произвольные объекты (void *) и работы с динамически отводимой памятью, поэтому в данной статье мы представим его же существенно упрощенную модель работы с нитями.

Мы исходим из того, что основным способом применения РуСи является цепочка:

«графическая модель –> текст на РуСи –> компилятор -> интерпретатор»,

причем интерпретатор может работать как на компьютере, так и непосредственно на роботе.

В графической модели нити представляются просто точкой разветвления на диаграмме, поэтому в этом случае никаких динамически формируемых нитей быть не может, соответственно все нити могут быть пронумерованы в процессе генерации текста на РуСи.

В тексте на РуСи каждая нить, сгенерированная из графического представления, обрамляется скобками специального вида t_create_direct и t_exit_direct, префикс t_ означает thread, русские эквиваленты — н_создать_непоср и н_конец_непоср, префикс н_ означает нить.

Для динамического формирования нити в тексте на РуСи нужно вызвать функцию t_create (русский эквивалент н_создать) с параметром — функцией, являющейся телом нити. Эта функция имеет один параметр и тип результата void *, чаще всего в РуСи ни то, ни другое не используется, но мы оставили такой тип функции для совместимости со стандартом POSIX Threads. Чуть позже мы покажем использование функции t_create на примере. Нити нумеруются последовательно с 1 (главная программа считается нитью номер 0) в порядке вызовов функции t_create.

Стандрат POSIX Threads включает в себя около ста функций, большинство из которых используется очень редко. В то же время наш многолетний опыт программирования встроенных систем реального времени привел нас к выбору одного из многих возможных вариантов синхронизации параллельных процессов (семафоры, почтовые ящики, мониторы Хоара, рандеву языка Ада) — это обмен сообщениями, которые позволяют не только синхронизировать параллельные процессы, но и дают удобный механизм обмена информацией между ними. А как раз обмена сообщениями в стандарте POSIX Threads нет. Авторам пришлось заняться расширением этого стандарта.

В нашем случае обмен сообщениями нужен в основном для получения информации от многочисленных датчиков. Датчики не работают со сложными структурами данных — только с целыми числами, если встретится более сложный случай, всегда можно придумать какую-то специальную кодировку или табличное представление. Таким образом, мы предлагаем все сообщения между нитями представлять в виде структуры

```
struct msg_info{int numTh; int data;}
```

При посылке сообщения поле numTh содержит номер нити, в которую посылается сообщение, при приеме сообщения — номер нити, откуда сообщение пришло, data — это содержание сообщения.

Для приостановки работы нити используется функция t_sleep с одним целым параметром — количество миллисекунд, на которые нужно приостановить работу нити, русский эквивалент н_спать.

Посылка сообщения осуществляется функцией t_msg_send с параметром типа struct msg_info, русский эквивалент н_послать. РуСи имеет расширение стандартного языка Си для записи структур {a, b}, особенно удобное для формирования параметра оператора н_послать.

Прием сообщения осуществляется функцией t_msg_receive без параметров, выдающей значение типа struct msg_info, русский эквивалент н_получить .

Каждая нить имеет очередь входных сообщений. Функция н_послать асинхронная, то есть нить, выполнившая эту функцию, может спокойно продолжать свою работу. Функция же н_получить синхронная, то есть она проверяет наличие у нити, из которой она вызвана, входных сообщений в очереди. Если сообщения есть, первое из них удаляется из очереди и выдается в качестве значения функции н_получить, если же входных сообщений нет, нить засыпает и не занимает никаких ресурсов. Когда нити следует проснуться и возобновить работу — это забота операционной системы, а не РуСи, но понятно, что это произойдет, когда нить получит ожидаемое сообщение.

Чтобы не возиться с динамически выделяемой памятью, мы ограничили длину очереди получаемых сообщений константой __COUNT_MSGS_FOR_TH, в данный момент эта константа равна 4, но, разумеется, ее легко изменить.

В начале 1960-х годов знаменитый ученый Эдсгер Дейкстра предложил новую концепцию в программировании — семафоры, над которыми возможны три операции: *установить* (level), *опустить* (down) и *поднять* (up). Операция *опустить* уменьшает значение семафора на 1, если это значение становится отрицательным, процесс, выполнивший эту операцию, приостанавливается (suspend). Операция *поднять* увеличивает значение семафора на 1, если какой-то процесс был приостановлен из-за этого семафора, он продолжит работу (resume). Важно, что операции *опустить* и *поднять* являются неделимыми, то есть действия добавить/отнять единицу и проверка на знак должны выполняться подряд без каких-либо прерываний со стороны других процессов.

С помощью семафоров можно гарантировать, что в так называемых критических участках может работать не более одного из параллельных процессов. Оказалось, что основную идею семафоров — неделимость операций *опустить* и *поднять* — невозможно реализовать (с той же эффективностью) другими традиционными операциями языков программирования, то есть семафоры — действительно принципиально новые языковые конструкции.

Стандарт POSIX Threads включает в себя эти три действия над семафорами.

Определить новый семафор (действие *установить*) можно с помощью функции t_sem_create, которая имеет целый параметр level (начальное значение семафора) и выдает целое значение — номер нового семафора. Русский эквивалент этой функции н_создать_сем. Семафоры нумеруются в порядке вызовов функции t_sem_create.

Действие *опустить* осуществляется с помощью функции t_sem_wait с целым параметром — номером семафора, русский эквивалент н_вниз_сем.

Действие *поднять* осуществляется с помощью функции t_sem_post с номером семафора в качестве параметра, русский эквивалент н_вверх_сем.

Функция t_create получает параметром функцию, которая, собственно, и является телом нити, но в РуСи все функции исполняются одним и тем же интерпретатором.

Эту техническую трудность мы преодолели следующим образом. Вся доступная память делится на куски одинакового размера (по одному для каждой нити), есть глобальный массив threads, *i*-ый элемент которого указывает на начало памяти *i*-ой нити, а в первых словах этой памяти хранятся регистры *l* и *x* *i*-ой нити. Функция t_create получает

параметром функцию interpreter, которая сначала выясняет номер нити, устанавливает l и x, а потом продолжает работу с нужного места (значение pc передается параметром).

3. ПРИМЕР «БАССЕЙН»

Есть бассейн, в котором хотелось бы иметь теплую воду и который надо автоматизировать. Рядом с бассейном стоит расширительный бачок, нагревательные элементы стоят именно в нем. Из бассейна в расширительный бачок идет труба с насосом для откачки воды и ее подогрева. Из расширительного бачка идет труба с насосом для подкачки воды в бассейн, температура воды измеряется именно в бассейне, а не в расширительном бачке.

Управляемые устройства:

- Насос для набора воды;
- Насос для откачки воды;
    - У насосов есть всего 3 команды:
        1. Включить;
        2. Выключить;
        3. Установить мощность насоса при работе.
- Нагревательный элемент с 2 командами (включить и выключить).

Датчики:

- Датчик уровня воды (разовый, срабатывает, когда вода достигает определенного уровня);
- 2 датчика расхода воды (на входе и на выходе);
- Датчик температуры воды.

Процедура наливки воды:

- Включить насос для набора воды и ждать, пока не придет сообщение от датчика уровня воды (это датчик событийный, то есть разового характера).
- Запускаются две параллельных нити: в одной измеряется уровень воды (циркуляция) в бассейне, а в другой — температура воды в бассейне.
- В нити опроса температуры в цикле выполняются следующие действия: опросить датчик, если выдаваемое им значение больше некоторого параметра, послать сообщение в главную программу об этом событии, иначе вызвать процедуру t_sleep(int time) с определенным параметром (скажем, 1000 миллисекунд).
- В нити измерения циркуляции воды в цикле выполняются следующие действия: опросить датчики s_1 и s_2 расхода воды и вычислить разницу между этими значениями U. Пусть стандартная мощность мотора P, тогда на первый насос надо подать мощность. $P - k * U$, где k — коэффициент управляющего воздействия (подбирается экспериментально), а на второй насос подать $P + k * U$. Опрос датчиков надо делать каждые 20–50 мсек.

```
struct msg_info{int numTh; int data}msg;

// numTh при посылке сообщения говорит куда, при приеме – откуда
int state = 0; /* 0 – нагревание, 1 – остывание */
int tmin = 30, tmax = 60, /* градусов */
hit = 1; /* 1 – это номер датчика нагрева */
```

Каждый датчик надо оформить нитью, например,

```
t_create_direct
do
        val = getdigsensor(hit);
        t_msg_send({N, val});
        t_sleep(1000);  /* 1 секунда */
while(1);
t_exit_direct;

t_create_direct /* нагрев */
msg = t_msg_receive();
if (state == 0 \&\& msg.data > tmax)
        {
                t_msg_send({hit, off});  /* команда выключить нагреватель */
                state = 1;
        }
else if (state == 1 \&\& msg.data < tmin)
        {
                tmsgsend({hit, on});  /* команда включить нагреватель */
                state = 0;
        }
t_exit_direct;
```

4. ЗАКЛЮЧЕНИЕ

В статье предложена «обертка» стандарта POSIX Threads, которая существенно упрощает взаимодействие пользователей с параллельными нитями, реализованными во всех популярных операционных системах, с расширениями, позволяющими нитям обмениваться сообщениями. Эта «обертка» реализована в проекте РуСи, прошла практическую проверку на многих нетривиальных примерах и показала свою высокую эффективность с точки зрения времени переключения процессов, обмена сообщениями и работы с семафорами.

Список литературы

1. *Терехов А. Н., Лучин Р. М., Филиппов С. А.* Образовательный кибернетический конструктор для использования в школах и вузах. Сборник избранных трудов «VI Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование». С. 11. URL: http://www.math.spbu.ru/user/ant/All_articles/096_Terekhov_Filipov_Luchin_robots.pdf (дата обращения: 22.04.2018).
2. Blaise Barney Lawrence Livermore National Laboratory. POSIX Threads Programming. URL: https://computing.llnl.gov/tutorials/pthreads/ (дата обращения: 22.04.2018).
3. *Головань А. А.* Реализация многопоточности в проекте РуСи. Дипломная работа — 2018.
4. *Терехов А. Н., Терехов М. А.* Платформа РуСи для обучения и создания высоконадежных программных систем // Известия высших учебных заведения Северо-Кавказский Регион. Серия: Технические науки. 2017. № 3. С. 70–75.

Поступила в редакцию 19.03.2018, окончательный вариант — 22.04.2018.

Computer tools in education, 2018
№ 2: 25–30
http://ipo.spb.ru/journal

PARALLEL PROGRAMS IN RuC PROJECT

Terekhov A. N.[1], Golovan A. A.[1], Terekhov M. A.[1]

[1]Saint-Petersburg State University, Saint Petersburg, Russia

Abstract

The article describes the extension of RuC project, which allows the use of parallel threads of the POSIX Threads standard. This made it possible to significantly improve the efficiency of the use of numerous sensors in the now popular systems of the Internet of things and in robots developed at the faculty of mathematics and mechanics of St. Petersburg state University.

Keywords: *Parallel threads, POSIX Threads standard, RuC project.*

Citation: A. N. Terekhov, A. A. Golovan, and M. A. Terekhov, "Parallel Programs in RuC Project," *Computer tools in education*, no. 2, pp. 25–30, 2018 (in Russian).

Received 19.03.2018, the final version — 22.04.2018.

Andrey N. Terekhov, professor, Head of Software Engineering Chair, SPbSU; 198504, Saint-Petersburg, Peterhof, Universitetsky pr. 28, Software Engineering Chair, a.terekhov@spbu.ru
Alexandr A. Golovan, student of Department of Informatics, SPbSU, alex_golovan@bk.ru
Mikhail A. Terekhov, student of Software Engineering Chair, SPbSU, st054464@student.spbu.ru

Терехов Андрей Николаевич, доктор физико-математических наук, профессор, заведующий кафедрой системного программирования СПбГУ; 198504, Санкт-Петербург, Петергоф, Университетский пр. 28, кафедра системного программирования, a.terekhov@spbu.ru

Головань Александр Александрович, студент кафедры информатики СПбГУ, alex_golovan@bk.ru

Терехов Михаил Андреевич, студент кафедры системного программирования СПбГУ, st054464@student.spbu.ru