Е.М. Кравцунов, д.т.н., проф. С.В. Семенихин
(ЗАО «МЦСТ», ОАО «ИНЭУМ им. И.С. Брука»)

E. Kravtsunov, S. Semenikhin

УПРАВЛЕНИЕ ЭНЕРГОПОТРЕБЛЕНИЕМ СИСТЕМЫ НА КРИСТАЛЛЕ «ЭЛЬБРУС-2С+» В СОСТОЯНИИ ПРОСТОЯ ПРОЦЕССОРНОГО ЯДРА

POWER MANAGEMENT CPUIDLE SUBSYSTEM IN OS «ELBRUS» KERNEL

Описана система программного управления энергопотреблением, разработанная для двухъядерного микропроцессора «Эльбрус-2С+» в составе ВК «Монокуб». Управление выполняется средствами ОС «Эльбрус», основанной на ядре Linux-2.6.33. Приведены результаты измерений и некоторые особенности реализации.

Ключевые слова: ядро Linux, управление энергопотреблением, «Эльбрус-2С+».

Paper reports on the experimental results of power management algorithm research for «Elbrus-2s+» dual core processor. Power management is carried out from operation system «Elbrus», based on linux-2.6.33 kernel, ported to e2k processor architecture. Paper covers the details of architecture dependent implementation of cpuidle driver and dynamic ticks subsystem implementation. Results of power consumption measurements for «Monocub» server platform built on «Elbrus-2s+» processor are presented in the arcticle.

Keywords: linux kernel, power management, cpuidle.

Введение

Существенное значение для серверных платформ имеет проблема энергопотребления в состоянии простоя [1], вызванного временным отсутствием подлежащих выполнению задач. В системах на базе операционной системы (ОС) Linux при этом вызывается служебная функция, которая в наиболее простом случае представляет собой цикл, ожидающий

прихода прерывания или установки специального флага, приводящего к перепланированию задач, т.е. конвейер команд работает, и подается питание. Таким образом, при простое часть энергии от источника питания тратится впустую. Для уменьшения энергопотребления в этом случае было предложено использовать состояния сна процессора, если они поддерживаются аппаратурой [2]. В принципе, могут вводиться несколько состояний C1–Cn, которые различаются глубиной сна – количеством отключаемых физических элементов и временем возврата в активное состояние C0.

При аппаратной поддержке состояний сна задача сводится к реализации в ОС алгоритма, определяющего номер наиболее приемлемого из них (с учетом средней загруженности процессора) и фактически переводящего в это состояние процессорное ядро при простое. Существенное значение в логике алгоритма имеет продолжительность периода простоя, не прерываемого от внешних источников. В частности, к нежелательному эффекту приводят прерывания от таймера, инициирующего периодическое переключение задач. По этой причине для исключения лишних прерываний в случае отсутствия задач (типичная ситуация для простоя) вводится режим динамических прерываний от таймера.

Поставленная проблема решалась применительно к вычислительному комплексу (ВК) «Монокуб», построенному на базе микропроцессора (системы на кристалле) «Эльбрус-2С+», где интегрируются универсальная часть, включающая два процессорных ядра и северный мост, и кластер из четырех цифровых сигнальных процессоров (DSP). ВК «Монокуб» работает под управлением ОС «Эльбрус» на базе ядра Linux-2.6.33 и может быть использован для построения мощных серверных платформ.

В микропроцессор «Эльбрус-2С+» введены следующие механизмы аппаратной поддержки управления энергопотреблением, выполняемого средствами ОС:

- система команд дает возможность реализовать состояние сна C1, основанное на отключении конвейера;
- в составе северного моста имеются контроллеры, позволяющие обеспечить генера-

цию динамических прерываний по таймеру.

Кроме того, возможность отключения синхроимпульса в процессорном ядре позволила провести измерения, необходимые для оценки эффективности описанных в статье методов.

В состав программных средств, на основе которых решалась поставленная задача, входили как стандартные архитектурно-независимые модули Linux [3, 4], так и реализованные автором архитектурно-зависимые модули.

Важным фактором при проведении работы было наличие встроенного программно-недоступного алгоритма управления энергопотреблением, физически реализованного в микропроцессоре «Эльбрус-2С+». В связи с этим, помимо решения основной проблемы, в работе проведена сравнительная оценка эффективности встроенного алгоритма.

1. Основной цикл управления энергопотреблением

Функциональное описание

При попадании процессорного ядра в режим простоя ОС входит в основной цикл управления энергопотреблением – архитектурно-зависимую функцию cpu_idle – до появления задачи пользователя или прихода внешнего прерывания. На каждой итерации цикла делается попытка перевести процессор в одно из состояний сна. С этой целью алгоритм модуля governor, учитывающий длительность простоя и среднюю нагрузку ОС, определяет, можно ли перейти в одно из состояний сна, и при положительном результате вычисляет его номер. Решение формируется с учетом влияния выбранного состояния на производительность системы (падение производительности не допускается, ибо оно само по себе приводит к дополнительному расходу энергии). Перевод вычислительного ядра в выбранное состояние производится с помощью архитектурно-зависимой функции enter, реализованной в модуле e2k_idle. Если же оценка предсказывает падение производительности в любом из состояний, то процессор остается в цикле простоя.

Перечисленные программные компоненты объединяются в архитектурно-независимом модуле cpuidle. Используя интерфейсы модуля governor, он осуществляет выбор наиболее приемлемого состояния сна и переводит в него процессорное ядро через интерфейс модуля e2k_idle. Функциональность модуля cpuidle вызывается на каждой итерации основного цикла cpu_idle.

Функция cpu_idle. Без поддержки состояний сна cpu_idle выполняет цикл, приведенный в листинге 1. На каждой его итерации вызывается функция cpu_relax(), представляющая собой барьер – ожидание завершения всех операций обращения в память. Выход из цикла возможен через установку условия необходимости перепланировки need_resched или через прерывание, обработка которого приводит к перепланировке (вызову функции __schedule), например, прерывание от таймера.

Листинг 1: Функция cpu_idle

```
arch/e2k/kernel/process.c:
……
/*
 * Powermanagement idle function, if any..
 */
void (*pm_idle)(void);
EXPORT_SYMBOL(pm_idle);

……
/*
* We use this if we don't have any better
* idle routine..
*/
void default_idle(void)
{
   int cpu = raw_smp_processor_id();

   if (psr_and_upsr_irqs_disabled()) {
            local_irq_enable();
   }

   /* loop is done by the caller */
   cpu_relax();
}
EXPORT_SYMBOL(default_idle);

void __cpuinit select_default_idle_routine()
{
   pm_idle = default_idle;
}

void cpu_idle(void)
{
   int cpu = raw_smp_processor_id();
   int cpuup = 0;

while (1) {
```

```
        while (!need_resched()) {

                rmb(); /* check the case carefully */
                if (cpu_is_offline(cpu)) {
                        play_dead();
                        cpuup = 1;
                        break;
                }

                 /* Process RCU */
                 ……

                  local_irq_disable();
                 pm_idle();

        }

      /* enable interrupts on cpuup path after play_dead */
        if (cpuup) {
                raw_local_irq_enable();
                cpuup = 0;
        }

        local_irq_disable();
        __preempt_enable_no_resched();
        __schedule();
        preempt_disable();
        local_irq_enable();
    }
    machine_halt();
}
```

Для того чтобы поддержать переход в состояние сна, в архитектурно-зависимой части ядра объявляется глобальный указатель на функцию pm_idle, а в тело цикла cpu_idle добавляется вызов pm_idle (*см.* листинг 1). Он инициализируется значением указателя на функцию default_idle, которая в цикле ожидает завершения всех операций обращения в память. Указатель pm_idle в процессе работы может быть изменен на функцию cpu_idle_call (листинг 2), реализующую основную часть алгоритма управления переходами в состояния сна (выбор состояния и переход в него).

Листинг 2: функция cpu_idle_call

```
drivers/cpuidle/cpuidle.c:
……
/**
 * cpuidle_idle_call - the main idle loop
 *
 * NOTE: no locks or semaphores should be used here
 */
static void cpuidle_idle_call(void)
{
        struct cpuidle_device *dev = __get_cpu_var(cpuidle_devices);
        struct cpuidle_state *target_state;
        int next_state;

        /* check if the device is ready */
```

```
        if (!dev || !dev->enabled) {
                if (pm_idle_old)
                        pm_idle_old();
                else
                        default_idle();
                return;
        }

        /* ask the governor for the next state */
        next_state = cpuidle_curr_governor->select(dev);
        if (need_resched()) {
                local_irq_enable();
                return;
        }

        target_state = &dev->states[next_state];

        /* enter the state and update stats */
        dev->last_state = target_state;
        dev->last_residency = target_state->enter(dev, target_state);
        if (dev->last_state)
                target_state = dev->last_state;

        target_state->time += (unsigned long long)dev->last_residency;
        target_state->usage++;

        /* give the governor an opportunity to reflect on the outcome */
        if (cpuidle_curr_governor->reflect)
                cpuidle_curr_governor->reflect(dev);
        }
}
```

Модуль cpuidle. Значение глобального указателя pm_idle может менять только архитектурно-независимый модуль cpuidle. При его инициализации вызывается функция, которая присваивает вызову pm_idle указатель на функцию cpu_idle_call. Подсистема cpuidle выполняет реальную работу, если в ядре вводится псевдоустройство dev, имеющее тип cpuidle_device. Оно регистрируется и активизируется при загрузке архитектурно-зависимого модуля (в данном контексте – модуля e2k_idle). Из листинга 2 видно, что при отсутствии dev, имеющего архитектурно-зависимую реализацию, действие функции cpuidle_idle_call также сводится к вызову default_idle, т.е. поддержке единственного состояния C0. В противном случае она с помощью вызова select выбирает состояние сна, в которое будет переведен процессор, после чего с помощью вызова enter осуществляет перевод.

Модуль governor. Выбор состояния выполняется архитектурно-независимой реализацией модуля governor, управляющего политиками переходов. Он принимает решение на основе статистики продолжительности работы функции cpu_idle и характерного времени выхода из соответствующего состояния сна. Как правило, время выхода из состояния глу-

бокого сна с минимальным потреблением электроэнергии больше, чем время выхода из состояния неглубокого сна с большим энергопотреблением. При выборе решения может использоваться информация о средней загрузке процессора, а также выполняться адаптация к неравномерным по времени изменениям загрузки процессора. Возможность адаптации реализуется с помощью вызова reflect (*см.* листинг 2) после выхода из состояния сна. Таким образом, функция cpu_idle_call для взаимодействия с модулем governor использует два его интерфейса – select и reflect. Для перевода процессора в состояние сна эта функция использует вызов интерфейса enter устройства dev, реализация которого является архитектурно-зависимой.

В Linux-2.6.33 реализованы два типа модуля governor – ladder и menu. Алгоритм ladder («лестница») основан на сравнении времени, проведенного в последнем основном цикле cpu_idle, с временем выхода из состояния сна. Очевидно, что при больших периодах отсутствия активности на процессоре в варианте ladder принимается решение в пользу более глубоких состояний сна. В варианте menu используется похожий алгоритм, который лучше адаптируется к изменяющимся нагрузкам. При принятии решения используется информация о средней загруженности системы за определенный промежуток времени, а также через вызов reflect учитываются изменения статистики продолжительности cpu_idle.

Драйвер e2k_idle. Единственным состоянием сна для универсального ядра микропроцессора «Эльбрус-2С+» является состояние C1, аппаратная поддержка которого состоит в отключении конвейера команд (рассматривается ниже). Переход в это состояние вследствие вызова enter выполняется в архитектурно-зависимом драйвере e2k_idle, который выполнен в виде загружаемого модуля. Если драйвер не загружен, то устройство dev в функции cpu_idle_call (*см.* листинг 2) является неактивным, что приводит к выполнению default_idle. При загрузке драйвера устройство dev инициализируется с помощью архитектурно-независимых интерфейсов подсистемы cpuidle – cpuidle_register_driver, cpuidle_register_device. В результате загрузки модуля драйвера для устройства определя-

ются два состояния сна: C0 – простой цикл ожидания с вызовом cpu_relax на каждой итерации цикла и C1 – отключение конвейера процессорного ядра в ожидании прерывания. В качестве вызова enter для входа состояния C0 и C1 определяется функция e2k_enter_idle, представленная на листинге 3.

Листинг 3: вход в состояние C1 в драйвере e2k_idle

```
arch/e2k/kernel/cpuidle.c:
….
/* Interface for entering the sleep state */
static int e2k_enter_idle(struct cpuidle_device *dev,
                          struct cpuidle_state *state)
{
  ktime_t before, after;
  thread_info_t *thread_info = current_thread_info();

  thread_info->wtrap_jump_addr = PG_JMP;

  before = ktime_get();
  if (state == &dev->states[0]) {
        while (!need_resched()) {
              default_idle();
        }
  } else if (state == &dev->states[1]) {
        SET_WTRAP_JUMP_ADDR("jump_over_wtrap");
        local_irq_enable();
        wtrap(); /* wait trap=1 */
        JUMP_OVER_WTRAP_LABEL("jump_over_wtrap");
  }

  if (thread_info->wtrap_jump_addr == PG_JMP) {
        after = ktime_get();
        thread_info->wtrap_jump_addr = 0UL;
  } else {
        after = (ktime_t) (thread_info->time_in_idle_ns);
  }
  return ktime_to_ns(ktime_sub(after, before)) >> 10;
}
```

Отсюда видно, что вход в состояние C0 является очень простым действием – вызовом функции default_idle в цикле ожидания условия выхода из цикла (необходимость перепланировки) или прихода прерывания, обработка которого приведет к перепланировке. Состояние C0 полностью повторяет действия функции cpu_idle без поддержки состояний сна. Переход в состояние C1 основан на использовании аппаратных средств поддержки состояний сна.

Аппаратная поддержка состояний сна

В процессорном ядре «Эльбрус-2С+» в качестве одной из разновидностей операции

WAIT введена операция остановки конвейера команд wait trap. При выполнении wait trap=1 (запись программно-доступного бита) конвейер команд останавливается в ожидании прерывания от контроллера прерываний LAPIC, а при прерывании возобновляет свою работу. Согласно системе команд «Эльбрус-2С+» операция wait trap=1 остается невыполненной в период ожидания прерывания и по приходу прерывания. Это означает, что командный поток, в котором встретилась такая операция, не может самостоятельно преодолеть точку остановки конвейера.

Для использования этого механизма в cpu_idle в данной работе было реализовано искусственное «перепрыгивание» командного потока через операцию wait trap (листинг 3), которое выполняется с помощью макросов SET_WTRAP_JUMP_ADDR и JUMP_OVER_WTRAP_LABEL. Макросы реализуют подготовку к «перепрыгиванию» командного потока через операцию wait trap=1. путем изменения адреса возврата командного потока cpu_idle при входе в обработчик прерывания. В этом случае после завершения обработки прерывания, запустившего конвейер, возврат из прерывания происходит на команду, следующую за командой, содержащей операцию wait trap=1.

Динамические прерывания от таймера

Источниками прерываний, вызывающих переключение процессов в микропроцессоре «Эльбрус-2С+», являются таймеры контроллера LAPIC, расположенные в северном мосте. Для каждого вычислительного ядра реализован отдельный таймер, поддерживающий два режима работы: периодический (periodic) и режим генерации одиночного прерывания (oneshot). Если вычислительный комплекс относительно долгое время находится в простое, то таймер, работающий в периодическом режиме, продолжает генерировать прерывания, вызывающие переключения с цикла простоя обратно на него же. Полученная статистика позволяет сделать вывод о том, что в этом режиме фактически исчезает повод для перевода ядра в состояние сна. Эта проблема решается с использованием механизма дина-

мических прерываний от таймера [5], который базируется на аппаратных таймерах, поддерживающих режим генерации одиночного прерывания. В режиме oneshot интервал срабатывания таймера может быть программно изменен в любой момент. Таким путем можно откладывать генерацию прерывания до ситуации, когда прерывание от таймера действительно будет необходимо, например, при появлении задач пользователя или приходе внешнего прерывания. За счет этого продолжительность непрерываемых временн х интервалов в состоянии простоя существенно увеличивается.

Результаты, полученные при реализации этого принципа применительно к микропроцессору «Эльбрус-2С+» при отсутствии пользовательских задач, показывают, что частота прерываний от таймера в состоянии простоя за счет использования режима oneshot (15 прерываний/с) снизилась почти в семь раз по сравнению с режимом periodic (101 прерываний/с).

Реализация режима динамических прерываний от таймера для «Эльбрус-2с+» потребовала внесения изменений в архитектурно-зависимую функцию cpu_idle. Переключение таймера в режим oneshot выполняется перед входом в цикл простоя (листинг 4). В случае появления задачи пользователя обратное переключение в режим periodic выполняется после основного цикла функции. При необходимости обработки внешнего прерывания с последующим переводом таймера в режим periodic обратное переключение выполняется из функции irq_enter архитектурно-независимой части ядра.

Листинг 4: поддержка режима динамических прерываний в cpu_idle

```
arch/e2k/kernel/process.c:

void cpu_idle(void)
{
        ……....
        /* Tell NOHZ code that we are in the idle loop */
         tick_nohz_stop_sched_tick(1);

        while (!need_resched()) {

             ……...

        }
        /* Disable NOHZ mode */
        tick_nohz_restart_sched_tick();
```

```
                ……...
}
```

Все приведенные ниже результаты измерения мощности получены в режиме динамических прерываний.

2. Экспериментальные результаты

Решения по части управления энергопотреблением микропроцессора «Эльбрус-2с+» были опробованы в серии измерений, проведенных на ВК «Монокуб» с одним микропроцессором этого типа. Важным фактором при проведении работы было наличие физически реализованного в микропроцессоре встроенного алгоритма управления энергопотреблением. Поэтому важна была сравнительная оценка эффективности этих аппаратных механизмов с введенными автором в ОС механизмами поддержки состояния сна.

В процессе всех экспериментов с помощью мультиметра измерялся потребляемый тока на номинале источника питания 12 В, отвечающем за питание процессора. Это позволило легко привести полученные оценки к значениям мощности.

Наборы тестов

Тестовые измерения выполнялись для двух случаев:

- режим низкой нагрузки – отсутствие какой-либо деятельности на процессоре в течение 30 с;
- нагрузочное тестирование многопоточными счетными задачами.

Измерения тока

Для того чтобы оценить эффективность встроенного аппаратного алгоритма, был выбран стандартный набор счетных задач Splash-2 [6], загружающих оба ядра с архитектурой «Эльбрус». Предварительно экспериментальным путем было установлено, что в этом варианте хорошо моделируется естественная нагрузка ВК. Измерялись пиковые значения

тока в трех случаях: при наличии нагрузки – 2,27 А; при отсутствии нагрузки в состоянии простоя C0 – 2,17 А; при отсутствии нагрузки в состоянии сна C1 (конвейер команд отключен) – 2,11 А. Сравнение показывает, что при переходе из состояния нагрузки в состояние простоя за счет встроенного алгоритма потребление тока на источнике 12 В снижается на величину 100 мА, т.е. экономия мощности составляет около 4%. Отключение конвейера команд, выполняемое из ядра ОС в состоянии C1, позволяет экономить до 60 мА, что составляет экономию мощности 1,7% по сравнению с состоянием простоя C0. Таким образом, текущая реализация процессора «Эльбрус-2с+» с помощью встроенных аппаратных алгоритмов и алгоритмов управления энергопотреблением ОС позволяет снизить мощность на 5,7%.

Моделирование состояния сна с отключенным синхроимпульсом

Одним из вариантов реализации состояния сна является отключение синхроимпульса. Тем не менее, в микропроцессоре «Эльбрус-2С+» использовать его нет возможности в силу того, что аппаратура не позволяет выполнить обратное включение по внешнему прерыванию в «спящее» ядро. Поэтому в интересах будущих разработок была поставлена задача получить количественные данные, позволяющие сделать вывод о необходимости изменений в аппаратуре, связанных с устранением этой особенности, иными словами, определить, какие выгоды даст реализация состояния сна, в котором отключается синхроимпульс.

В проведенном эксперименте был использован принятый для ядра Linux принцип логического и аппаратного включения/выключения процессоров из ОС (механизм hotplug) [7].

В табл. 1 приведены результаты измерений для вариантов работы с двумя и одним включенными ядрами, которые показывают, что отключение синхроимпульса на вычислительном ядре позволяет экономить до 500 мА, т.е. получить экономию мощности в 28%.

Таблица 1

Значения тока в трех вариантах нагрузки с учетом отключения синхроимпульса

	2 ядра			**1 ядро**		
	Нагрузка	**Простой С0**	**Состояние сна С1**	**Нагрузка**	**Простой С0**	**Состояние сна С1**
Ток, А	2,27	2,17	2,11	1,77	1,7	1,67

Измерения времени

Временн е характеристики, которые подтверждают целесообразность проведения описанной работы в рамках стандартных механизмов ОС Linux, способствующих сокращению энергопотребления, приведены в табл. 2. Для иллюстрации совместимости принятых решений с алгоритмами реализованных в ядре ОС модулей governor menu и governor ladder оценивались относительные интервалы времени нахождения вычислительных ядер процессора «Эльбрус-2С+» в состояниях С0 и С1 в двух вариантах: простой ВК в течение 30 с (отсутствие нагрузки); работа ВК, загруженного счетной задачей raytrace из набора splash-2. Здесь: T_{CO} – время, проведенное в состоянии простоя, в котором энергия не экономится (цикл ожидания новой активности); T_{C1} – время, проведенное в состоянии сна, устанавливаемом средствами ОС, развитыми в данной работе.

Таблица 2

Временн е параметры модулей governor menu и governor ladder

	При отсутствии нагрузки ВК				При наличии нагрузки ВК			
	Ladder		**Menu**		**Ladder**		**Menu**	
	T_{C0}, %	**T_{C1}, %**	**T_{C0}, %**	**T_{C1}, %**	**T_{C0}, %**	**T_{C1}, %**	**T_{C0}, %**	**T_{C1}, %**
Cpu0	16,63	83,37	3,07	96,93	3,28	96,72	73,95	26,05
Cpu1	2,57	97,43	2,58	97,42	0,38	99,62	76,5	23,5
Average	**9,6**	**90,4**	**2,83**	**97,17**	**1,83**	**98,17**	**75,23**	**24,77**

Из таблицы следует, что при простое более существенную экономию обеспечивает модуль governor menu. В режиме нагрузки этот модуль менее эффективно экономит мощность. Однако, вследствие адаптивности алгоритма menu к изменениям нагрузки, его использование является более предпочтительным, т.к. минимизируется влияние модуля

governor на общую производительность. По умолчанию в ядре ОС «Эльбрус» используется governor menu.

Заключение

В статье описаны программные средства, введенные в ОС Linux для реализации состояния сна универсальных процессорных ядер микропроцессора «Эльбрус-2С+» с целью сокращения энергопотребления. Проведенные измерения показали, что при их использовании энергопотребление в периоды простоя может быть снижено на 5,7% по сравнению с моментами пиковой нагрузки. Экономия на каждое ядро может быть увеличена до 28%, если в будущих реализациях микропроцессоров этой серии помимо остановки конвейера команд будет использоваться отключение синхроимпульса.

Дальнейшая работа в этой области предполагает реализацию алгоритмов управления частотой в аппаратуре и ОС, расширение набора состояний сна и усовершенствование механизма динамических прерываний.

Литература

1. David Meisner, Brian T. Gold, Thomas F. Wenisch, PowerNap: Eliminating Server Idle Power, ASPLOS'09, March 7-11 2009, Washingtonm DC, USA.

2. Advanced Configuration and Power Interface Specification, Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, Toshiba Corporation, Revision 4.0a, April 5, 2010.

3. Thomas Gleixner, Douglas Niehaus, Hrtimers and Beyond: Transforming the Linux Time Subsystems, Proceedings of the Linux Symposium 2006, Volume One, p. 333.

4. Venkatesh Pallipadi, Shaohua Li, Adam Belay, cpuidle – Do nothing, efficiently… , Proceedings of the Linux Symposium 2007, Volume Two, p. 119.

5. Suresh Siddha, Venkatesh Pallipadi, Arjan Van De Ven, Getting Maximum Mileage out

of Tickless, Proceedings of the Linux Symposium 2007, Volume Two, p. 201.

6. Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, «The SPLASH-2 Programs: Characterization and Methodological Considerations», Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995.

7. Rusty Russell, Srivatsa Vaddagiri CPU hotplug Support in Linux Kernel – http://git.kernel.org linux-2.6.33/Documentation/cpu_hotplug.txt.