УДК 510.635

Л.В. Шабанова-Кушнаренко, Н.В. Борисова, О.Ю. Чередниченко, Ю.Н. Гонтарь

Национальный технический университет "ХПИ", Харьков

ПРОТОКОЛ RACPWP ИЕРАРХИЧЕСКОЙ ПЛАНОВОЙ СИСТЕМЫ ДЛЯ ЗАДАЧ С ВЫСОКОЙ КРИТИЧНОСТЬЮ

Иерархическое планирование используется для группировки приложений на основе их функциональности для улучшения агрегирования ресурсов и временной изоляции между приложениями. Коэффициент критичности задачи используется для сопоставления задач с ядрами в кластере подсистем. В статье представлены результаты полученного подхода иерархического планирования, когда планирование подсистем корректируется на основе уровня критичности, а не только строго установленного срока. Результатом является встроенная система, которая лучше приспособлена для адаптации к вычислительным вариациям, обеспечивающим гарантии времени для выполнения задач с высокой критичностью, что обеспечивает минимальный уровень обслуживания для снижения требований к критичности.

Ключевые слова: *иерархическое планирование, встроенная система, протокол синхронизации ресурсов, адаптивное планирование.*

Введение

Для обеспечения безопасного механизма принудительного внедрения вводится новая конструкция программирования для синхронизации ресурсов, известная как превентивные критические разделы (PCS). PCS использует форму транзакционной памяти программного обеспечения (STM) для перезапуска транзакции, которая была вытеснена задачей с более высоким приоритетом [1]. После того, как задача с более высоким приоритетом освободила ресурс, задача с более низким приоритетом откатывается и перезапускается.

Основное преимущество этого подхода заключается в том, что задача с более высоким приоритетом выполняется быстро. Фактически, наихудший случай блокировки равен оставшемуся бюджету задачи с более низким приоритетом, которая разделяет глобальный ресурс. Другим преимуществом этого подхода является устранение увеличенных периодов блокировки, возникающих в случае, когда задача с более низким приоритетом позволяет перерасходовать свой бюджет, как используется в протоколе OPEN-HSRP. В результате RACPwP обеспечивает улучшенное время отклика жестких задач в реальном времени, ослабляя ограничение, наложенное на время выполнения критических секций иерархическими плановыми системами [2].

1 Превентивные критические разделы

Чтобы проиллюстрировать протокол RACPwP [3], рассмотрим сценарий на рис. 1. Во время $t = 2$ $Task_2$ запрашивает и получает доступ к критическому разделу. Во время $t = 3$ жесткая задача $Task_1$ в режиме реального времени вытесняет $Task_2$ и выполняется. Во время $t = 4$ $Task_1$ запрашивает общий ресурс, заблокированный $Task_2$, и блокируется. Во время $t = 5$ бюджет для $Task_2$ истек. Задача $Task_2$ предварительно упущена, и $Task_1$ разрешено блокировать ресурс и продолжать. Наконец, в момент $t = 11$ $Task_1$ завершается, и $Task_2$ позволяет снова заблокировать ресурс и ввести его критический раздел.

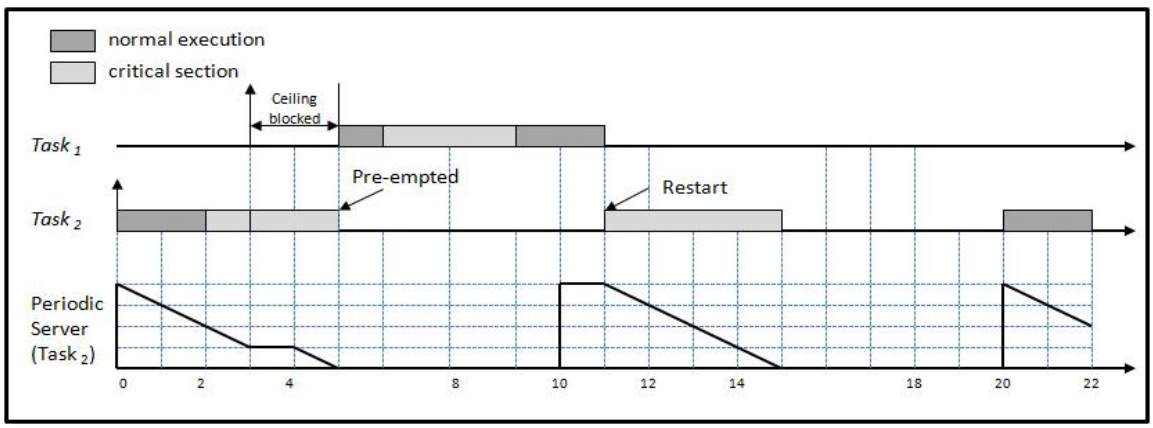

Рис. 1. Пример протокола контроля доступа к ресурсам с преимущественным доступом

На рис. 2 представлена примерная функция, которая имитирует функцию переноса банковского баланса, реализованную с использованием традиционных механизмов блокировки (т.е. семафоров [4]).

```
int transfer ( ) {
1.  sem_wait ( );
2.  value = source.balance
3.  value = value – amount
4.  source.balance = value;
5.  value = destination.balance
6.  value = value + amount
7.  destination.balance = value
8.  sem_give( )
}
```

Рис. 2. Традиционный механизм блокировки

Рис. 3 обеспечивает ту же функцию, реализованную с использованием критически важного раздела с преимущественным доступом.

```
int transfer ( ) {
1.  PCS_START ( );
2.  value = PCS_LOAD(source.balance)
3.  value = value – amount
4.  PCS_STORE(source.balance)
5.  value = PCS_LOAD(destination.balance)
6.  value = value + amount
7.  PCS_STORE(destination.balance)
8.  PCS_COMMIT
}
```

Рис. 3. Предопределяемый критический раздел

На рис. 2 семафор получен в строке 1, а остальные 6 команд не могут быть выгружены. Независимо от приоритета задача не может быть пресечена до тех пор, пока семафор не будет выпущен в строке 8. На рис. 3 фрагмент кода иллюстрирует механизм PCS, который реализуется как набор макросов. Макрос PCS_START в строке 1 выполняет начальную контрольную точку, требуемую, если функция выгружена и требует перезапуска. Другие макросы PCS_STORE и PCS_LOAD выполняют транзакции доступа к памяти, а макрос PCS_COMMIT в строке 8 фиксирует транзакции в памяти. Преимущество этих макросов в отличие от традиционных механизмов синхронизации, функция передачи может быть выгружена в любой момент ее выполнения (за исключением операции атомного PCS). Поэтому, если задача превысила свой бюджет, но все еще находится внутри критического раздела (т.е. PCS_COMMIT не был выполнен), RACPwP может безопасно выполнить задачу. Затем задача блокируется до следующего пополнения бюджета и разрешается снова перезапустить.

Важно отметить, что существуют некоторые ограничения, связанные с использованием PCS. Одно из них – это дополнительные вычислительные издержки, которые накладывает система на транзакционную память программного обеспечения. Другим ограничением является традиционный ввод-вывод, который не должен выполняться в транзакции.

2 Анализ эффективности

В этом разделе представлен анализ производительности RACPwP как части иерархической плановой системы. Производительность оценивается с использованием анализа времени отклика в худшем случае. Используя метод, предоставленный авторами в [5], наихудшее время отклика задачи T_i, обслуживаемой подсистемой S_i, происходит во время одного из следующих сценариев:

– Бюджет подсистемы исчерпан, как только запущены задачи с более низким приоритетом, и если задача находится в критическом разделе, она выгружается.

– Задача T_i и все другие приоритетные задачи в приложении поступают сразу после исчерпания бюджета подсистемы.

– Бюджет подсистемы пополняется, но исполнение задерживается как можно дольше из-за помех от других подсистем с более высоким приоритетом.

Основываясь на приведенных выше сценариях, наихудшее время отклика задачи может быть вычислено путем определения интервала времени, когда могут выполняться задачи на уровне приоритета i или выше. Этот интервал времени или окна выполнения w определяется тремя компонентами [6]:

1. Выполнение задачи T_i вместе со всеми задачами с более высоким приоритетом на первом уровне приоритета.

2. Периоды пополнения любых полных серверов.

3. Взаимодействие с серверами с более высоким приоритетом (задачи, выполняемые в подсистеме с более высоким приоритетом).

Поэтому наихудшее время отклика задачи T_{si} можно вычислить, используя уравнение:

$$\mathrm{wcrt} = w_{si}^{n} + J_s , \qquad (3)$$

где w_{si}^{n} может быть определена с помощью функции повторения и J_s является джиттером выполнения задачи T_{si}:

$$w_{si}^{n+1} = L(w_{si}^{n}) + G(w_{si}^{n}) + I(w_{si}^{n}) . \qquad (4)$$

Анализ ответа наихудшего случая был расширен авторами в [7], чтобы включить разделение ресурсов между подсистемами. Нагрузка $L(w_s^n)$ на i-м уровне приоритета расширяется на один термин до включения последствий локальных и глобальных факторов блокировки и определяется как:

$$L(w_{si}^n) = B_{si} + C_{si} \sum_{\forall j \in hp(i)} \left(\frac{w_s^n + j_i}{T_i} \right) C_{si}, \qquad (5)$$

где коэффициент блокировки B_{si} связан с локальным и глобальным доступом к ресурсам. Период пополнения $G(w_{si}^n)$, включенный в глобальные блокирующие факторы, определяется как:

$$G(w_{si}^n) = \left(\left[\frac{L(w_s^n) + j_i}{C_s} \right] - 1 \right) (T_s - C_s) + B_s, \qquad (6)$$

где B_s представляет собой самое длинное время S_i, когда сервер может быть заблокирован от выполнения сервером с более низким приоритетом. Интерференция $I(w_{si}^n)$ с любого сервера с более высоким приоритетом определяется как:

$$I(w_{si}^n) = \sum_{\forall X \in hp(S)} \left(\frac{\max\left(0, w_{si}^n - \left(\frac{L(w_{si}^n)}{C_S} - 1 \right) T_i \right)}{T_X} \right) \times \qquad (7)$$

$$\times (C_X - B_{X0}),$$

где B_{S0} представляет собой время переполнения сервера S_j, которое является самым длинным временем, которое может выполнить сервер. Кроме того, уравнение (10) представляет собой переполнение сервера без окупаемости. Для анализа перерасхода бюджета с окупаемостью $I(w_{si}^n)$ определяется следующим образом:

$$I(w_{si}^n) = \sum_{\forall X \in hp(S)} B_{x0} + \sum_{\forall x \le i} \left(\frac{\max\left(0, w_{si}^n - \left(\frac{L(w_{si}^n)}{C_S} - 1 \right) T_s \right)}{T_X} \right) C_X \qquad (8)$$

(Обратите внимание, что для RACPwP термин B_S перерасход сервера в уравнении (6) равен нулю, поскольку задача будет выгружена, если срок действия бюджета сервера истек, удерживая блокировку на глобальном ресурсе) [8].

3 Сравнительный анализ протоколов

Здесь представлен пример для целей оценки между RACPwP и другими протоколами, использующими механизмы перерасхода бюджета. В нашем примере мы сравниваем время отклика сервера и задачи наихудшего случая для RACPwP, OPEN-HSRP с окупаемостью бюджета и OPEN-HSRP без окупаемости бюджета. Время отклика сервера для OPEN-HSRP с окупаемостью бюджета рассчитывается на основе следующего:

$$w_s^{n+1} = C_s + B_s + \sum_{\forall X \in hp(S)} B_{x0} + \sum_{\forall X \in hp(S)} \left(\frac{w_{si}^n)}{T_x} \right) C_x. \qquad (9)$$

Время отклика сервера для OPEN-HSRP без окупаемости бюджета рассчитывается на основе следующего:

$$w_s^{n+1} = C_s + B_{s0} + \sum_{\forall X \in hp(S)} \left(\frac{w_s^n)}{T_x} \right) (C_x + B_{x0}). \qquad (10)$$

Рекуррентные функции для уравнений (9) и (10) начинаются с $w_s^0 = 0$ и заканчиваются, когда $w_s^{n+1} = w_s^n$, это наихудшее время отклика сервера [9]. Если $w_s^{n+1} > T_s$, тогда сервер не является планируемым и поэтому не рассматривается. Моделируемые системы состоят из трех отдельных подсистем, каждый из которых запланирован глобальным упреждающим периодическим сервером. Глобальный ресурс распределяется между каждой подсистемой C_i, а время выполнения критического раздела представлено как $CSET_i$. Термин J_i представляет собой джиттер сервера подсистемы.

Чтобы оценить время ответа сервера в худшем случае, мы использовали 100 симуляционных прогонов, в которых использовался генератор случайных чисел для изменения емкости сервера, бюджета сервера и времени выполнения критического раздела. Диапазоны параметров сервера, которые были использованы для генерации параметров сервера, приведены в табл. 1.

Таблица 1

Параметры сервера

S_i	C_i	T_i	J_i	$CSET_i$
S_1	[50,500]	[200,2000]	[150,1500]	[35,200]
S_2	[1250, 2500]	[5000, 10000]	[3750, 7500]	[35,200]
S_3	[3000, 5000]	[12000, 20000]	[9000, 15000]	[35,200]

В табл. 2 приведены наихудшее время ответа для RACPwP, OPEN-HSRP без окупаемости бюджета (HSRPnP) и OPEN-HSRP с окупаемостью бюджета (HSRPwP).

Как показывает анализ планирования, приведенный в табл. 2, время ответа сервера улучшается с помощью RACPwP, так как не выполняет перерасход бюджета. Время отклика сервера практически идентично для подсистемы S_1, так как сервер с наи-

высшим приоритетом не подлежит переполнению. Однако обратите внимание, что RACPwP значительно улучшает время отклика сервера для подсистем S_2 и S_3 не влияет на механизм перерасхода.

Таблица 2

Среднее время ответа сервера в худшем случае

S_i	RACPwP	HSRPnP	HSRPwP
S_1	390	390	390
S_2	2360	3400	2900
S_3	5550	12150	10012

Следующим шагом будет оценка наихудшего времени отклика задачи RACPwP по сравнению с OPEN-HSRP с и без окупаемости бюджета. Функция повторения для наихудшего времени отклика задачи начинается с $w_s^0 = 0$ и заканчивается, когда $w_s^{n+1} = w_s^n$, где время ответа наихудшего случая - $w_i^n + J_i$.

Задача не является планируемой, если $w_i^{n+1} > D_i - J_i$. В этом случае она не рассматривается для анализа. Подсистема была выбрана для выполнения задач, поскольку это подсистема приоритета среднего уровня. В этом примере общий ресурс $CSET_i$ распределяется между задачами, а также локальным ресурсом. Время выполнения критического раздела локального ресурса представлено $CSET_{si}$. Подобно параметрам сервера, генератор случайных чисел использовался для изменения времени выполнения худшего случая задачи, периода задачи и крайнего срока. Выполнение локального ресурса также различалось. Диапазоны параметров задачи, которые были использованы для изменения параметров задачи, определены в табл. 3.

Таблица 3

Параметры задачи

τ_i	C_i	T_i	D_i	$CSET_{si}$
T_1	[1180, 2375]	[12500, 25000]	[12500, 25000]	[150, 350]
T_2	[3150, 4500]	[35000, 50000]	[35000, 50000]	[150, 350]

Табл. 3 представляет наихудшее время отклика для всех задач в подсистеме S_2. Подсистема S_2 была выбрана, поскольку она является подсистемой приоритета среднего уровня и лучше всего иллюстрирует влияние нашего протокола на общую систему. Время ответа худшего случая задается в соответствии с функцией повторения (4).

Как показано в табл. 4, очевидно, что общий доступ к ресурсам может существенно повлиять на общую нагрузку задачи. Результатом является то, что задачи с более низким приоритетом могут быть выгружены и должны быть перезапущены. Результат иллюстрируется увеличением времени отклика.

Таблица 4

Время отклика худшего случая

τ_i	RACPwP	HSRPnP	HSRPwP
T_1	7665	8860	9388
T_2	28900	23800	26200

Задачи с более низким приоритетом оплачивают самую крутую цену, поскольку она может страдать от множества превенций, связанных с задачами с более высоким приоритетом [10].

Обратите внимание, что RACPwP обеспечивает улучшенное время отклика по HSRPnP и HSRPwP для задач с более высоким приоритетом, но задачи с более низким приоритетом могут ухудшить время отклика.

Это неотъемлемый компромисс с RACPwP. Задачи с более высоким приоритетом, которые, как правило, трудны в режиме реального времени, выигрывают от более низкого времени отклика, что обеспечивает улучшенный детерминизм.

Однако задачи с более низким приоритетом, которые обычно являются мягкими в режиме реального времени, более терпимы к увеличению времени отклика.

4 Синхронизация адаптивных ресурсов

В этом разделе представлен обзор политики синхронизации адаптивных ресурсов [11], которая представляет собой протокол доступа к ресурсам, который синхронизирует доступ к общим ресурсам в иерархически запланированной системе. Мы рассматриваем только взаимоисключающие ресурсы. А именно, разделяемая память, но также может включать в себя другие общие ресурсы (например, области с отображением памяти, регистры устройств, периферийные устройства). Доступ к этим ресурсам выполняется как часть критического раздела и защищен семафором.

Подобно протоколам SIRAP и OPEN-HSRP, ARAP использует двухуровневую иерархию для управления ресурсами [12]. Ресурсы, которые совместно используются в подсистеме, управляются с помощью SRP, а ресурсы, которые совместно используются в подсистемах, управляются с расширенной версией HSRP.

Общая последовательность действий для ARAP приведена в блок-схеме, изображенной на рис. 4.

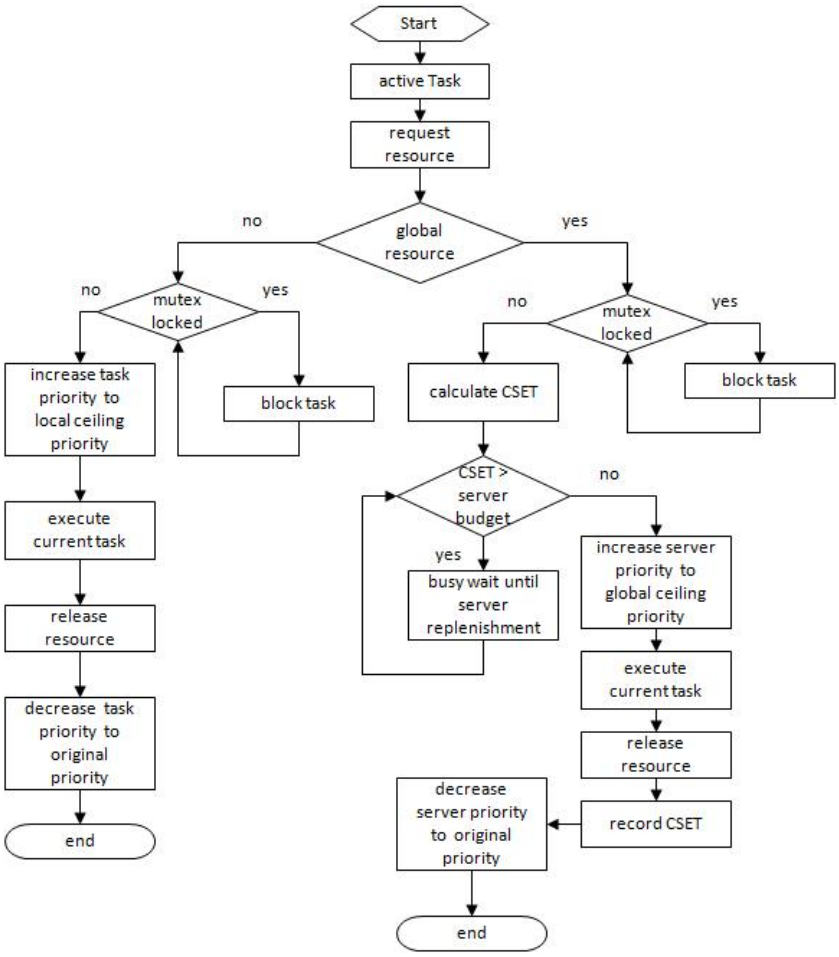

Рис. 4. Блок-схема архитектуры ARAP

Выводы

Основное различие между различными протоколами доступа к ресурсам в HSF заключается в том, как обрабатывается бюджетное истощение сервера подсистемы. Как упоминалось в предыдущем разделе, протокол SIRAP выполняет проверку бюджета, в то время как OPEN-HSRP разрешает пополнение бюджета. Подобно протоколу SIRAP, наш метод также выполняет форму проверки бюджета. Однако вместо использования статического априорного вычисления $CSET_i$, ARAP включает в себя обратную связь для оценки следующего времени удержания ресурса $h_{i,j}$ для экземпляра выполнения критического раздела.

References

1. Behnam, M., Nolte, T., Sjodin, M and Shin, I. (2007), SIRAP: A synchronization protocol for hierarchical resource sharing real-time open systems, *Proc. 7th ACM and IEEE Int. Conf. Embedded Software (EMSOFT 07)*.

2. Davis, R.I. and Burns, A. (2006), Resource Sharing in Hierarchical Fixed Priority Pre-emptive Systems, *Proc. of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*.

3. Fisher, N., Bertogna, M. and Baraugh, S. (2007), The Design of an EDF-Scheduled Resource Sharing Open Environment, *RTSS '07*.

4. Asberg, M., Nolte, T. and Behnam, M. (2013), Resource Sharing Using the Rollback Mechanism in Hierarchically Scheduled Real-Time Open Systems, *RTSA '13*.

5. Davis, R.I. and Burns, A. (2005), *Hierarchical Fixed Priority Pre-emptive Scheduling*, Dept. Comp. Sci.Univ. of York.

6. Behnam, M., Nolte, T. Sjodin, M. and Shin, I. (2010), Overrun Methods and Resource Holding Times for Semi-Independent Real-Time Systems*, IEEE trans. on Indus. Informatics*.

7. Sha, L., Rajkumar, R. and Lehoczky, J.P. (1990), Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE trans. Comput.,* Vol 39, pp. 1175-1185.

8. Phinjaroenphan, P., Beivinakoppa, S. and Zeephongsekul, P. (2005), A Method For Estimating the Execution Time of a Parallel Task on a Grid mode, *Advances in Grid Computing – EGC 2005*.

9. Carpenter, J., Frank, S., Holman, P., Srinivasan, A., Anderson, J. and Baruah, S. (2003), A categorization of real-time multiprocessor scheduling problems and algorithms, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, CRC Press LLC.

10. Zhang, F. and Burns, A. (2007), Analysis of hierarchical EDF pre-emptive scheduling, *Proc. of the 28th IEEE International Real-Time Systems Symposium (RTSS'07)*.

11. Richardson, P. and Sarkar, S. (1999), Adaptive scheduling: overload scheduling for mission critical systems, *RTSA*.
12. Feng, X., Shen, X., Liu, L., Wang, Z. and Sun, Y. (2005), Fuzzy logic based feedback scheduler for embedded control systems, *Advances in Intelligent Computing*.

Поступила в редколлегию 6.10.2017
Одобрена к печати 2.11.2017

Відомості про авторів:

Шабанова-Кушнаренко Любов Володимирівна
кандидат наук асистент кафедри
Національного технічного університету
"Харківський політехнічний інститут",
Харків, Україна
https://orcid.org/0000-0002-2080-7173
e-mail: l.v.shabanova.kushnarenko@gmail.com

Борисова Наталя Володимирівна
кандидат наук доцент кафедри
Національного технічного університету
"Харківський політехнічний інститут",
Харків, Україна
https://orcid.org/0000-0002-8834-2536
e-mail: borisova_nv@mail.ru

Чередніченко Ольга Юріївна
кандидат наук доцент кафедри
Національного технічного університету
"Харківський політехнічний інститут",
Харків, Україна
https://orcid.org/0000-0002-9391-5220
e-mail: olha.cherednichenko@gmail.com

Гонтар Юлія Миколаївна
аспірант кафедри
Національного технічного університету
"Харківський політехнічний інститут",
Харків, Україна
https://orcid.org/0000-0002-3748-5086
e-mail: gontaryn@gmail.com

Information about the authors:

Shabanova-Kushnarenko Lyubov
Candidate of Sciences Assistant Lecturer of Department
of National Technical University
"Kharkiv Polytechnic Institute",
Kharkiv, Ukraine
https://orcid.org/0000-0002-2080-7173
e-mail: l.v.shabanova.kushnarenko@gmail.com

Borisova Natalya
Candidate of Sciences Associate Professor of Department
of National Technical University
"Kharkiv Polytechnic Institute",
Kharkiv, Ukraine
https://orcid.org/0000-0002-8834-2536
e-mail: borisova_nv@mail.ru

Cherednichenko Olga
Doctor of Philosophy Associate Professor of Department
of National Technical University
"Kharkiv Polytechnic Institute",
Kharkiv, Ukraine
https://orcid.org/0000-0002-9391-5220
e-mail: olha.cherednichenko@gmail.com

Gontar Yulia
PhD student of Department
of National Technical University
"Kharkiv Polytechnic Institute"
Kharkiv, Ukraine
https://orcid.org/0000-0002-3748-5086
e-mail: gontaryn@gmail.com

ПРОТОКОЛ RACPWP ІЄРАРХІЧНИХ ПЛАНОВОЇ СИСТЕМИ ДЛЯ ЗАВДАНЬ З ВИСОКОЮ КРИТИЧНІСТЮ

Л.В. Шабанова-Кушнаренко, Н.В. Борисова, О.Ю. Чередніченко, Ю.М. Гонтар

Ієрархічне планування використовується для угруповання додатків на основі їх функціональності для поліпшення агрегування ресурсів і тимчасової ізоляції між додатками. Коефіцієнт критичності завдання використовується для зіставлення задач з ядрами в кластері підсистем. У цій статті представлені результати отриманого підходу ієрархічного планування, коли планування підсистем коригується на основі рівня критичності, а не тільки строго встановленого терміну. Результатом є вбудована система, яка краще пристосована для адаптації до обчислювальних варіацій, що забезпечує гарантії часу для виконання завдань з високою критичністю, що забезпечує мінімальний рівень обслуговування для зниження вимог до критичності.

Ключові слова: *ієрархічне планування, вбудована система, протокол синхронізації ресурсів, адаптивне планування.*

THE RACPWP PROTOCOL OF THE HIERARCHICAL PLAN SYSTEM FOR TASKS WITH HIGH CRITICALITY

L. Shabanova-Kushnarenko, N. Borisova, O. Cherednichenko, Y. Gontar

Hierarchical planning is used to group applications based on their functionality to improve resource aggregation and temporary isolation between applications. The task criticality factor is used to compare tasks with the cores in a subsystem cluster. Tasks with high criticality are statically compared with specific kernels for assurance of determinism, while lower criticality tasks allow you to transfer through the cores in a cluster to improve performance. Adaptive hierarchical planning on fuzzy logic for periodic real-time tasks is an approach in which a heuristic method based on fuzzy logic is used to enable the system to adapt. This article presents the results of the obtained approach of hierarchical planning, when the planning of subsystems is corrected on the basis of the level of criticality, and not just the strictly established term. The result is an integrated system that is better suited to adapt to computational variations, providing time guarantees for tasks with high criticality, which provides a minimum level of service to reduce criticality requirements. A practical application for sharing resources in hierarchical planning systems is actual equipment based on aerospace equipment in the simulation of cycles. Protocols for resource synchronization based on pre-emptive and predicted resources were used in the simulation and proved to be effective in improving response time and eliminating time-misses for hard real-time tasks compared to traditional resource synchronization protocols.

Keywords: *hierarchical planning, built-in system, resource synchronization protocol, adaptive planning.*