Поддръжка на CSP в екзоядрото fiberOS

Милен Луканчевски, Николай Костадинов, Хованес Авакян

CSP Support by fiberOS exokernel: *The main contemporary contradiction in the area of computer systems is between sequential thinking and modern architectures with global structural parallelism. Accordingly, the main goal of education in the area of parallel computer systems is overcoming of that contradiction in some extent. To succeed in this authors apply as educational methodology the principle of minimum formal and practical instruments used in different evolving environments.*

CSP is used as formal model and specification language along with three-stage consistence package of workshops: single node multitasking environment (Windows/x86 platform, fiberOS executive, programming language C); conventional multiple node multitasking environment (X51/MCS51 node platform, X51m executive, programming language C); advanced multiple node environment with direct architectural support of parallelism in terms of processes and communications (SMT/DLP core xCORE, parallel programming language XC).

In first of those stages fiberOS non-preemptive cooperative exokernel is used. It is as simple as possible according to KISS principle and is based on Windows fibers. This paper presents main features of fiberOS in context of CSP support provided.

Key words: *CSP, exokernel, fibers, parallelism.*

ВЪВЕДЕНИЕ

Водещ методологичен принцип в науката е насочването на изследванията към основното, съществената противоречие на предметната област. Анализът на развитието на компютърните системи и архитектури през последните десетилетия позволява да се отдели най-съществената за съвременния етап противоречие: това между последователното мислене и съвременните архитектури с явен паралелизъм. Противоречие, определено в [1] за третата особена точка или третата сингулярност.

Пътят за преодоляването на тази особена точка преминава през съответния учебен материал. В рамките на специализирания курс по „Паралелни компютърни системи", преподаван в катедра „Компютърни системи и технологии" на Русенския университет „А. Кънчев", това разбиране се проследява в два възлови момента: изучаването на един от фундаменталните паралелни изчислителни модели и практическото му поетапно усвояване.

Теорията на *взаимодействащите последователни процеси CSP* [8] е предпочетена заради нейната многопластовост – CSP е едновременно формален математически модел, език за спецификация на паралелни системи и език за паралелно програмиране. Той може да се използва в целия диапазон – от формалната математическа абстракция за паралелизъм до физическата паралелна система. Оттук следва и възможността за следната етапност в практическото усвояване на основните механизми и понятия, свързани с паралелизма:

- едновъзлова апаратна среда с многозадачна надстройка (платформа *Windows/x86*, изпълнителна среда *fiberOS*, програмен език *C*);

- конвенционална многовъзлова апаратна среда с многозадачна надстройка (платформа на възлите *X51/MCS51*, изпълнителна среда *X51m*, програмен език *C*);

- многовъзлова апаратна среда с директна архитектурна поддръжка на паралелизма (*SMT/DLP* ядро *xCORE*, език за паралелно програмиране *XC*).

Системите, обект на всеки един от тези етапи, се специфицират на *CSP*. При това постоянно се извършва преход от абстрактното към конкретното и обратно, като паралелизмът се капсулира в основните абстрактни елементи на *CSP* (процеси, канали, оператори) и едновременно с това се вниква във вътрешните им механизми.

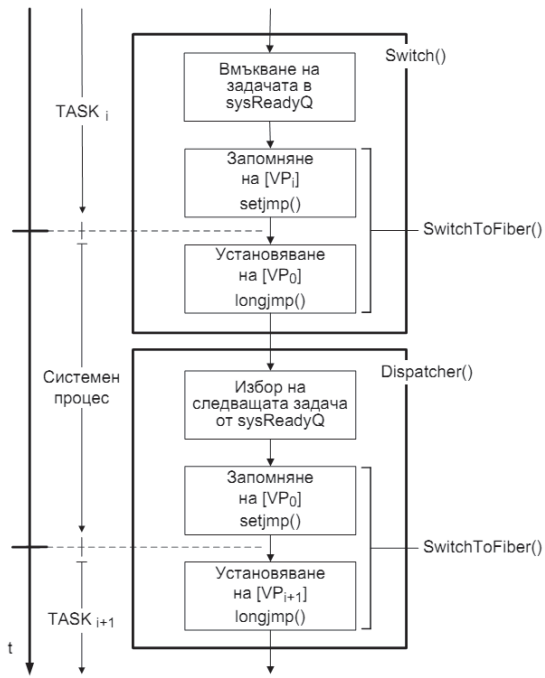
В доклада се разглежда принципът на работа на екзоядрото *fiberOS*, използвано на първия етап, когато се усвояват ключовите понятия на паралелния изчислителен модел *CSP*. Отделя се внимание на осигуряваната от ядрото поддръжка на *CSP*.

1. ПРИНЦИП НА РАБОТА

Основният движещ мотив при проектирането и използването на екзоядрото *fiberOS* е възможно най-опростената адекватна реализация на основните абстрактни елементи на паралелния изчислителен модел *CSP* (процеси, канали, оператори). Реализация, която позволява плавния взаимен преход между абстрактното и конкретното в хода на обучението.

Екзоядрото *fiberOS* е с диспечеризация на процесите на приложно ниво, базирана на влакната на ОС *Windows*. Подходът е описан за пръв път в [7].

Диспечеризацията е *кооперативна*, без изтласкване (*non-preemptive*). Известен начин за реализацията на подобна стратегия е използването на комплементарната двойка библиотечни функции *setjmp()/longjmp()*. На фиг. 1 е показано превключването на контекста на задачите (процесите) в ядрото *MICROS* [2].



Фиг. 1 – Превключване контекста на задачите (процесите)

Схемата на пострàничното резервиране на стека при ОС *Windows* затруднява приложението на двойката библиотечни функции *setjmp()/longjmp()*. Вместо тях се препоръчва използването на *влакна* (*fibers*). *Влакното* се определя като активна единица, изпълнявана в контекста на дадена нишка. Една нишка може да съдържа множество *влакна*, чиято диспечеризация се извършва в рамките на нишката на *кооперативен* принцип [10, 12].

Превключването на *влакната* се извършва симетрично или асиметрично. *Симетричният режим* на работа на влакната е аналогичен на *копроцедурите* (*coroutines*), но за разлика от тях *влакната* са системни, а не езикови конструкции [4, 6, 9]. При *fiberOS* се прилага *асиметричния режим на работа* – предаването на управлението не се извършва от самите *работни влакна*, а от *специализирано влакно - системен диспечер*.

За илюстрация на превключването между влакната отново може да се използва фиг. 1. Но за превключването на контекста VP_i на влакното $TASK_i$ се използва *API* функцията *SwitchToFiber()* на ОС *Windows*. Ролята на системния процес се изпълнява от *основното влакно*, в което предварително, чрез *API* функцията *ConvertThreadToFiber()*, е конвертирана *нишката* на изпълнение на влакната.

Умишлено не се използват библиотечни реализации, подобни на осигуряваните от библиотеката *Boost* [11]. Така се избягват допълнителни нива на абстракция и нежеланото в случая скриване на детайлите на реализацията. От друга страна се запазва максимална еднотипност в езика на реализацията - и на трите етапа от практическата работа той е *C*.

2. СИСТЕМНИ ПРИМИТИВИ

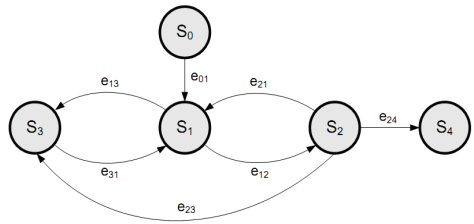
Понятията *„процес”* и *„канал”* са ключови за *CSP* [1, 8]. Процесът се разглежда като базова активна единица за декомпозиция на системата. На конкретното му изпълнение като *процес*, *задача*, *нишка* или *влакно* не се акцентира.

Съгласно *принципа на нивото* [2], *процесите* се интерпретират от ядрото като системни таблици (*дескриптори*). Дескрипторът *TASK* на процесите във *fiberOS*

```
typedef struct
{
  char tag[TAGSIZE];              // име на задачата
  TASKSTATE state;                // текущо състояние на задачата
  PTASKCONTEXT context;           // контекст на задачата
} TASK;                           // ДЕСКРИПТОР НА ЗАДАЧА /////////////////
```

съдържа трите най-съществени полета – *име*, текущо *състояние* и *контекст*.

От изключително значение е осмислянето на процеса като *машина на състоянията*. На фиг. 2 е представена диаграмата на състоянията и преходите на процесите при *fiberOS*.



Фиг. 2 – Диаграма на състоянията и преходите на задачите (процесите)

Множеството възможни състояния включва:
- S_0 или *TS_IDLE*, пасивно начално състояние;
- S_1 или *TS_READY*, готовност - процесът е в опашката на готовите задачи;
- S_2 или *TS_RUN*, активно - процесът се изпълнява;
- S_3 или *TS_BLOCK*, блокирано – процесът очаква събитие;
- S_4 или *TS_STOP*, пасивно крайно състояние.

Множеството възможни преходи между състоянията съдържа:
- e_{01}, води до преход от *TS_IDLE* в *TS_READY*;
- e_{12}, води до преход от *TS_READY* в *TS_RUN*;
- e_{21}, води до преход от *TS_RUN* в *TS_READY*;
- e_{31}, води до преход от *TS_READY* в *TS_BLOCK*;
- e_{13}, води до преход от *TS_BLOCK* в *TS_READY*;
- e_{23}, води до преход от *TS_RUN* в *TS_BLOCK*;

- e_{24}, води до преход от *TS_RUN* в *TS_STOP*.

След създаването на процеса чрез *CreateTask()* преходите между възможните състояния се извършват чрез останалите системни функции на ядрото – *Switch()*, *Suspend()*, *Resume()*, *Dispatcher()*, *Terminate()*.

3. ПОДДРЪЖКА НА *CSP*

Каналите са средство за взаимодействие между процесите. За разлика от процесите, те не винаги се поддържат непосредствено от ОС.

Семантиката на *CSP* предполага еднопосочни, двуточкови (*1:1*) канали, с двустранна синхронизация от вида рандеву. Дескрипторът *CHAN* на канала във *fiberOS* е съобразен с тези изисквания и има вида:

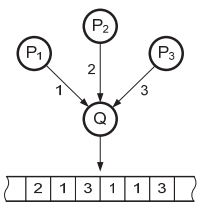
```
typedef struct
{
  char tag[CHAN_TAG_LEN];          // име на канала
  TASK* in;                        // задача на входа на канала
  TASK* out;                       // задача на изхода на канла
  TASK* block;                     // задача, блокирана на канала
  CHAN_MSG msg;                    // съобщение в канала
} CHAN;                            // ДЕСКРИПТОР НА КАНАЛ //////////////
```

Всеки един канал се създава чрез обръщение към *CreateChan()*, докато останалите системни функции директно отговарят на съответен *CSP* оператор:

```
CHAN* CreateChan(char* name, TASK* in, TASK* out);
bool SEND(CHAN* chanOut, CHAN_MSG* src);
bool RECV(CHAN* chanIn, CHAN_MSG* dst);
int ALT(CHAN* chanIn1, CHAN* chanIn2, CHAN_MSG* dst);
int ALT(vector < CHAN* > & vChanIn, CHAN_MSG* dst);
#define STOP    Terminate();
```

Така, функциите *SEND()* и *RECV()* отговарят на семантиката на операторите <*!*> и <*?*> на *CSP* за взаимодействие между процесите, а функцията ALT() – на оператора за недетерминиран избор (алтернативната команда) на *CSP*.

Функцията *ALT()* има два варианта – съкратен двуканален и обобщен *n*-канален. Поддържат се само *защити* (*guards*), представляващи оператори <*?*> за получаване на съобщения, тъй като единствено при тях може да се получи недетерминизъм [3, 5].



Фиг. 3 – Паралелна машина с недетерминизъм

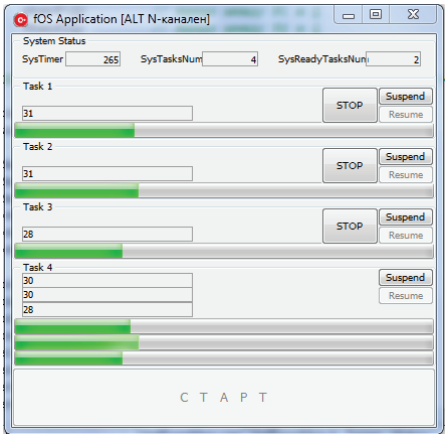
На фиг. 3 от [3] е показан графът на взаимодействията на паралелната система, описвана от *CSP* уравненията

$$\{P_1||P_2||P_3||Q\},$$
$$P_1 = P_2 = P_3 = P =* \{Q!msg \rightarrow \Delta \rightarrow SKIP\}_L,$$
$$Q =* \{P_1?x \rightarrow write(x) \rightarrow SKIP \square P_2?x \rightarrow write(x) \rightarrow SKIP \square P_3?x \rightarrow write(x) \rightarrow SKIP\}_{3\times L}.$$

Източниците P_1, P_2 и P_3 изпълняват по L на брой цикъла. В рамките на всеки цикъл изпращат към Q съобщението *msg*. Консуматорът Q е недетерминиран по определение – съдържа три алтернативи, чиито защити са команди за

взаимодействие. Изпълнява се (*3* x *L*) пъти, което отговаря на общия брой съобщения, изпращани от P_1, P_2 и P_3. Недетерминизмът е илюстриран на фиг. 3 чрез регистрацията на процеса, от който е получено съобщение – *1*, *2* или *3*.

На фиг. 4 е представено примерно изпълнение чрез *fiberOS* на така дефинираната паралелна машина с недетерминизъм.



Фиг. 4 – Примерно изпълнение във fiberOS на паралелна машина с недетерминизъм

Процесите *Task1*, *Task2* и *Task3* отговарят съответно на P_1, P_2 и P_3, а *Task4* – на *Q*. Източниците *Task1*, *Task2* и *Task3* работят независимо, като изпълняват един и същ алгоритъм. Той се свежда до генерирането, визуализирането и изпращането на целочислено съобщение към приемника *Task4*. От своя страна, приемникът *Task4* визуализира стойността на съдържащото се в съобщението цяло число.

ЗАКЛЮЧЕНИЕ

Основният движещ мотив при проектирането и използването на екзоядрото *fiberOS* е възможно най-опростената адекватна реализация на основните абстрактни елементи на паралелния изчислителен модел *CSP* (процеси, канали, оператори). Реализация, позволяваща в хода на обучението да се извършва плавен взаимен преход от абстрактното към конкретното и обратно.

Екзоядрото *fiberOS* поддържа диспечеризация на процесите на приложно ниво, базирана на *влакната* на ОС *Windows*. Диспечеризацията е *кооперативна*, без изтласкване (*non-preemptive*). *Процесите* са завършени машини на състоянията. Преходите между възможните състояния се осъществяват чрез няколко системни функции на ядрото – *Switch()*, *Suspend()*, *Resume()*, *Dispatcher()*, *Terminate()*.

Каналите, като средство за взаимодействие между процесите, отговарят на семантиката на *CSP* и са еднопосочни, двуточкови (*1:1*), с двустранна синхронизация от вида *рандеву*. На операторите *<!>* и *<?>* на *CSP* за взаимодействие между процесите отговарят съответно системните функции *SEND()* и *RECV()*. А чрез системната функция *ALT()* се поддържа семантиката на оператора за недетерминиран избор (алтернативната команда) на *CSP*.

Представеният подход е отражение на принципа за обучение, предполагащ *съчетаването на минимално количество формални модели и практически инструменти в рамките на различни среди с плавно нарастваща сложност.*

ЛИТЕРАТУРА

[1] Луканчевски, М. Моделиране на квантови изчисления в паралелна изпълнителна среда. Русе, Издателски център на РУ „А. Кънчев", 2013, стр. 164, ISBN 978-619-7071-25-2.

[2] Луканчевски, М. Системно програмиране за едночипови микрокомпютри. – С.: Техника, 1993.

[3] Луканчевски, М., Н. Костадинов, Х. Авакян. Относно недетерминизма при обработката на събитията в една SMT/DLP машина. В: Научни трудове на Русенския университет, Русе, Русенски университет, 2012, стр. 111-116, ISBN 1311-3321.

[4] Conway, M. Design of Separable Transition-Diagram Compiler. // Communications of the ACM, Vol. 6, Num. 7, pp. 396-408.

[5] Dijkstra, E. Guarded Commands, nondeterminancy and formal derivation of programs. // Communications of the ACM, Vol. 18, Num. 8, pp. 453-457.

[6] Dolan, S., S. Muralidharan, D. Gregg. Compiler Support for Lightweight Context Switching. // ACM Trans. Arch. Code Opt., Vol. 9, No. 4, Art. 36, 2013.

[7] Engler, D., M. Kaashoek, J. O'Toole, Jr. Exokernel: An Operating System Architecture for Application-Level Resource Management. // ACM SIGOPS Operating Systems Review, Vol. 29, Issue 5, pp. 251-266, Dec. 3, 1995.

[8] Hoare, C.A.R. Communicating Sequential Processes. // Communications of the ACM, Vol. 21, Num. 8, pp. 666-677.

[9] Moura, A., R. Ierusalimschy. Revisiting Coroutines. // ACM Trans. Program. Lang. Syst., Vol. 31, No. 2, Art. 6, 2009.

[10] Richter, J., C. Nasarre. Windows via C/C++, 5th Ed. - Microsoft Press, 2007.

[11] Web: Boost Library, извлечено септември 2014 от <http://www.boost.org/>

[12] Web: Fibers. - MSDN Library, извлечено септември 2014 от <http://msdn.microsoft.com/en-us/library/windows/desktop/ms682661(v=vs.85).aspx>

За контакти:

доц. д-р инж.Милен Луканчевски, Катедра "Компютърни системи и технологии", Русенски университет "Ангел Кънчев", тел.: 0887 303 850, e-mail: mil@ieee.org

гл. ас. инж. Николай Костадинов, Катедра "Компютърни системи и технологии", Русенски университет "Ангел Кънчев", тел.: 082 888 674, e-mail: nkostadinov@ecs.uni-ruse.bg

гл. ас. инж. Хованес Авакян, Катедра "Компютърни системи и технологии", Русенски университет "Ангел Кънчев", e-mail: havakian@ecs.uni-ruse.bg

Докладът е рецензиран.

РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ”
UNIVERSITY OF RUSE „ANGEL KANCHEV“

Д И П Л О М А

**Програмният комитет на
Научната конференция РУ&СУ’14
награждава с КРИСТАЛЕН ПРИЗ
“THE BEST PAPER”
доц. д-р МИЛЕН ЛУКАНЧЕВСКИ,
гл. ас. НИКОЛАЙ КОСТАДИНОВ
и гл. ас. ХОВАНЕС АВАКЯН
автори на доклада
“Поддръжка на CSP в екзоядрото fiberOS”**

D I P L O M A

**The Programme Committee of
the Scientific Conference RU&SU'14
Awards the Crystal Prize
"THE BEST PAPER"
to Assoc. prof. MILEN LOUKANTCHEVSKY,
NIKOLAY KOSTADINOV
and HOVANES AVAKIAN
authors of the paper
“CSP Support by fiberOS exokernel”**

РЕКТОР
RECTOR

проф. дтн Христо Белоев
Prof. DSc Hristo Beloev

25.10.2014