Годишник на секция "Информатика" **Annual of "Informatics" Section**
Съюз на учените в България **Union of Scientists in Bulgaria**
Том 6, 2013, 95–102 **Volume 6, 2013, 95–102**

ФОРМАЛНО ОПИСАНИЕ НА КОМПОНЕНТИ В ОПЕРАЦИОННИ СИСТЕМИ

Асен Петков Илиев

Университет „Проф.д-р Асен Златаров", Бургас
email: asen@asen.iliev.name

Резюме: Съвременното развитие на апаратните елементи са предпоставка за увеличаване на концентрацията на изчислителна мощност. Системният софтуер се развива с доста по бавни темпове. За оптимално използване на наличните ресурси се търси механизъм за моделиране. Формализацията на елементи, представена в материала предоставя база за моделиране. Представени са примери, доказващи работоспособността на концепцията.
Ключови думи: Операционни системи, Моделиране, Формално описание, Системно програмиране

1. Въведение

Развитието на компютърната техника поставя все по високи изисквания към програмното осигуряване с цел оптимално приложение. Приложното програмно осигуряване представлява сложен комплекс приложения, осигуряващи поддръжка на различни области. Изискванията към програмното осигуряване са различни с оглед на сферата на тяхното приложение. Много често тясното място на компютърните системи се оказват реализацията на компоненти от системното програмно осигуряване – операционна система, драйвери и др. – и взаимодействието между приложените и системните компоненти. Оптимизацията на системните компоненти е фактор, който до голяма степен се явява решаващ за реализация на специализирани приложения. За да се реализира оптимизация на системни компоненти се съблюдават принципи, които в някои отношения са различни от тези в приложното програмиране. В настоящия материал се представя методика за формално описание на компоненти от операционните системи. Такъв подход би дал възможност за реализация на оптимизация и моделиране на определени ситуации в дадена компютърна система.

2. Теоретична постановка

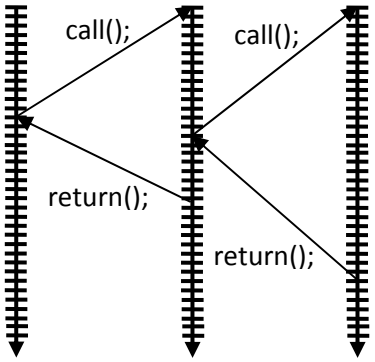
Абстрахирайки се от конкретната компютърна архитектура и спецификата на апаратната реализация на съответната компютърна система, съставните елементи на дадена операционна система могат да се представят по следния начин [1, 2]:

- ресурси – физически и логически;
- активности;
- интерфейси;
- протоколи.

Ресурсите в дадена операционна система са физически и логически. По презумпция на това място се пропускат и виртуални, имайки предвид, че те представляват част от логическите ресурси. Физическите ресурси са всички ресурси, които притежават материално изражение в компютърната система – физическа памет, процесор, диск, периферия и др. Логическите ресурси са въпрос на логическа дефиниция и са свързан по-скоро с представяне на компоненти в системата – страница във виртуална памет, сегмент с определена големина, програмен текст и др. От тази гледна точка може да се каже, че програмата от гледна точка на операционната система не представлява нищо повече от определен ресурс с „право на обработка". Правото на обработка се дава експлицитно от определения потребител или администратор на системата.

Под активности се има предвид всяко идентифицируемо въздействие в дадена система. Обработката на всяко приложение е свързана с обработка на съвкупност от инструкции по определен начин и/или в определен ред. Обработката на определен програмен код в смисъла на тази дефиниция води до определена активност. Налични са различни форми на активности [2]:

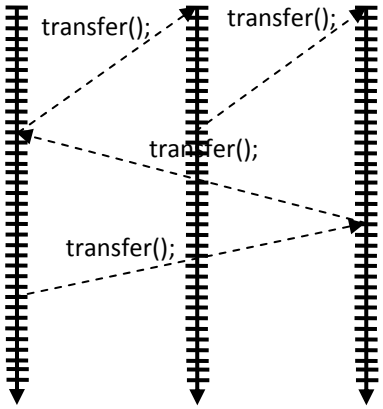
- процедури – те могат да притежават, както собствени ресурси, така и да се възползват от глобални такива. Имат собствен идентификатор. Предаването на идентификатора е на база call-return механизъм. Притежават една единствена входна точка. Могат и да се самоактивират;



Фигура 1. Диаграма на механизъм call-return

- съпътстващи процедури – представляват обобщаване на концепцията за процедурите. Притежават локални структури данни, но могат да се възползват и от глобални. За разлика от процедурите, съпътстващите процедури използват механизъм transfer() за активиране и деактивиране. Този механизъм предо-

ставя целево и експлицитно предаване на управлението при определени условия. Всяка активирана съпътстваща процедура се обработва до момента, до който тя предаде управлението посредством transfer(). След повторно активиране на дадена съпътстваща процедура, нейната обработка продължава от мястото, на което е реализиран предишния transfer();



Фигура 2. Механизъм transfer()

- процеси – те представляват до голяма степен отделени и независими елементи на операционната система. Всеки процес притежава минимум един активиращ идентификатор. Характерното за процеса е наличие на неговото глобално състояние и респективно неговото микросъстояние. Глобалното състояние на даден процес описва, какво в момента той прави – активен, изчакващ и др. Микросъстоянието на даден процес дефинира степента на неговата обработка към определен времемомент. Микросъстоянието може да се определя и алтернативно, на база брой инструкции, обработени до момента, съвкупност от ресурси в определен момент и др. Всеки процес се обработва в собствено адресно пространство.

Интерфейсите дефинират статичните аспекти на релацията между комуникиращи компоненти. Те описват синтаксиса на релацията – какви данни, типове, структури, как ще се използват те. Интерфейсите могат да бъдат ориентирани към управлението и ориентирани към данните.

Протоколите описват динамичните аспекти на релацията между комуникиращи компоненти. Те описват начина на протичане на комуникацията. Протоколите са решаващи за описание на семантиката в комуникацията между компоненти.

3. Формално описание на компоненти

3.1. Формална дефиниция на ресурси

За по голяма яснота, в началото се приема, че всеки ресурс представлява съвкупност елементи [2], структурирани по един и същ начин (памет – съвкупност от клетки, процесор – съвкупност от регистри, диск – съвкупност сектори и т.н.). Даден ресурс при това приемане се състои от множество M абстрактни, структурирани по един и същ начин елементи m, притежаващи адрес adr и съответстваща на адреса стойност w, т.е.:

$$m = (adr, w) \qquad adr \in ADR,\ w \in W$$

Съвкупността от всички $m \in M$ определя актуалното състояние на ресурса в определен времемомент. Както индивидуалният елемент m, така и конструираният на тази база ресурс M са времезависими. Символът $Z_p(t_i)$ ще се използва за представяне на състояние на ресурс M, в дискретен времемомент t_i:

$$Z(t) = \begin{pmatrix} w_1(t) \\ w_2(t) \\ \dots \\ w_n(t) \end{pmatrix}$$

при условие $ADR = \{1, \dots, n\}$.

Освен това, към всеки елемент може да се асоциират функции $f \in$ FUNC, трансформиращи състоянието на ресурси:

$$f : M \to M$$

$$Z(t_{i+1}) := Z(t_i)$$

Опростено може да се приеме, че $FUNC$ обхваща всички функции, които биха могли да бъдат прилагани към ресурса.

Формалното определение на ресурс може да се обобщи по следния начин: ресурса представлява подредена четворка, състояща се от идентификатор id, адресно пространство ADR, множество на заеманите стойности W за всеки един от елементите на ресурса и множество функции $FUNC$, които могат да се прилагат към всеки от елементите:

$$R = (id, ADR, W, FUNC)$$

3.2. Формална дефиниция на процес

Въз основа на вербалното описание на понятието процес P, може да се даде следното формално описание:

$$P = \left(id,\quad \vec{R}_a,\quad \vec{R}_r,\quad F,\quad W_a\left(\vec{R}_a\right)\right)$$

Отделните елементи на процеса P означават:

- *id* - еднозначен идентификатор;
- $\vec{R}_a$ - множеството заделени ресурси (вектор на притежаваните от процеса ресурси);
- $\vec{R}_r$ - множеството заявени от процеса ресурси (вектор на заявените от процеса ресурси);
- F - последователност от инструкции f ($f \in F$, при което $f : R \to R'$);
- $W_a\left(\vec{R}_a\right)$ - стойностите на всички заделени ресурси по време на създаването на процеса.

Процесите трябва еднозначно да се различават и затова е *id*. Още при създаването му, на процеса се заделят множество ресурси. Операционната система трябва да има информация за тях – като видове, конкретно кои и тяхното количество. Всяка машинна инструкция в състава на определена програма (с изключение на някои, напр. NOP) променя стойностите на определени ресурси. По този начин, обработката на процеса може да се обхване като последователност и/или съвкупност от трансформации на отделните ресурси. Самата програма обаче не представлява съвкупността от трансформации – тя просто ги описва. И на последно място, но не и последни по значение, са стойностите на заделяните ресурси, заявени от процеса. Пример за това: За осигуряване на по-висока степен на сигурност е необходимо предварително нулиране на стойностите на заявената памет.

4. Приложение на формалното описание

4.1. Модел на състоянията на процес

Поради факта, че в дадена компютърна система се обработват значително по-голям брой процеси от броя на процесорите, е необходимо да се въведат глобални състояния на процесите. В определен момент могат да се обработват точно толкова процеси, колкото е броят на процесорите. Диаграмите на състоянията в различните операционни системи се отличават, но тези различия не са драстични от общия модел, гарантиращ минимум три състояния [3, 4]. На базата на въведената формализация, този модел може да се опише по следния начин:

[1] активно: Процесът притежава всички заявени от него ресурси, включително процесор. Формално това условие се описва така:

$$\left|\vec{R}_a\right| = \left|\vec{R}_r\right|$$

[2] готовност: Процесът притежава всички заявени от него ресурси, но без процесор. Формалното описание изглежда така:

$$\left|\vec{R}_a\right| = \left|\vec{R}_r\right| - 1$$

[3] блокировка: Процесът чака заделянето на заявен(и) ресурс(и). Това състояние формално изглежда така:

$$\left|\vec{R}_a\right| < \left|\vec{R}_r\right| - 1$$

Формалните дефиниции определено опростяват описанието на диаграмата на състоянието.

4.2. Формализация на слоен модел на операционна система

На това място, използвайки формалните дефиниции, ще бъде показан интуитивен модел на операционна система, състояща се от m на брой слоя. Множеството S на всички слоеве S^i се дефинира:

$$S = \{S^i \mid 0 \le i \le m-1\}$$

Моделът се ограничава от най-ниския слой S^0, представляващ апаратната платформа и S^{m-1}, който е последният, най-висок в йерархията, който се определя от създаващия модела (при OSI, $m = 7$) [5, 6, 7].

Два съседни слоя S^j и S^{j+1} са свързани с две различни релации:

1. По-високият слой S^{j+1} използва услугите на по-ниския слой S^j:

$$S^{j+1} \nabla_d S^i$$

2. По-ниският S^j слой управлява по-високия S^{j+1}:

$$S^{j+1} \nabla_s S^i$$

(използва се в смисъл „извиква", „използва"). Последователността релации при обслужване $S^{m-1}\nabla_d S^{m-2}, S^{m-2}\nabla_d S^{m-3}, ..., S^1\nabla_d S^0$ конституират йерархията на обслужването. Всеки слой S^j притежава ресурси. Множеството ресурси на всеки слой се представя така:

$$R^i = \{R^i_j \mid 0 \le j \le n-1\}$$

Всеки слой може да обхваща различен брой ресурси (от там параметъра n). Всеки ресурс се консолидира само с определен слой:

$$R^i \cap R^j = 0$$

за $0 \le i, j \le m-1$ и $i \ne j$.

От множеството на всички налични ресурси в даден слой S^j могат да се определят подмножества, които в определен аспект могат да се изследват. От тях може да се формира съвкупност за управление:

$$C^i_j \subseteq R^i$$

Тази контролна единица представя определена активност, но тя принадлежи на същия слой, от който са ресурсите. Поради факта, че всяка активност е с ограничено във времето действие (ограничен живот), тя може да се дефинира като временна съвкупност от ресурси (temporary aggregation). Пример: На база програмен код, памет и идентификатор се създава процес. Възможният (теоретично, разбира се) брой

активности ще се характеризира посредством всички подмножества на R^i, т.е. мощността на R^i:

$$\{C_j^i\} = \prod(R^i)$$

Очевидно е, че не всяко свободно обхващане на ресурси в даден слой може да формира смислена активност. Действителното формиране на активност на база съвкупност от ресурси от даден слой S^i се поема от функция g, която е локализирана в непосредствено по-ниския слой:

$$g^{i-1} : \{R^i\} \Rightarrow C^i \quad \text{при } i \geq 1.$$

След прилагане на така дефинираната функция, пасивната съвкупност ресурси се превръща в активност. Пример: системното извикване fork() (всъщност, fork() е корефункция, елемент от основната библиотека, която се занимава с функционалността на системното извикване sys_fork(), но разглеждането на системни извиквания в детайли е извън рамките на настоящия материал [8]).

Не винаги агрегацията на ресурси от даден слой i е предпоставка за възникване на активност. По-точно казано, на базата на агрегация на ресурси вместо активност, резултатът може да е създаване на нов ресурс. Пример: Въз основа на известно количество блокове физическа памет, може да се създаде логическа памет. Това „създаване" на ресурс на по високо ниво би могло да се дефинира с друга функция f, по следния начин:

$$f^i : \{R^i\} \Rightarrow R^{i+1}$$

Функцията f^i създава ресурс на слой $i + 1$ на база на ресурс от слой i. Локализацията на такава функция е в двата слоя.

5. Насоки за бъдещи изследвания

Предложената в настоящия материал формализация би могла да намери приложение при моделиране процеси в системи реално време, с приложение в роботиката. Поради факта, че апаратната част се развива с по-бързи темпове от програмната, макар че днешните компютри не се отличават принципно от тези от 1960-а година, има видима тенденция за приложение на универсални операционни системи в конкретни управления – примерно Linux или Windows. Не остава скрит фактът, че тенденцията на „безплатен софтуер" налага използването на универсални операционни системи, но те са „неподготвени" за управление от тип реално време. Този подход налага моделиране на цялостни приложения в роботиката. Едно интересно приложение би могло да бъде моделиране на съвкупност интерфейси, предназначени за комплексно използване от незрящи потребители [9, 10].

Литература

[1] Tanenbaum A, Wetherall D., *Computer Networks, 5th Ed.*, Pearson Education Inc. 2011.

[2] Kalfa W., *Betriebssysteme*, TU-Chemnitz, 2002

[3] Bryant B., O' Hallarion D., *Computer Systems: A Programmer Perspective 2nd Ed.*, Prentice Hall 2011, ISBN-13: 978-0-13-610804-7

[4] Borovska, P., O. Nakov , D. Ivanova, K. Ivanov, G. Georgiev, Communication Performance Evaluation and Analysis of a Mesh System Area Network for High Performance Computers, *12th WSEAS International Conference on Mathematical Methods, Computational Techniques and Intelligence Systems (MAMECTIS'10)*, Kantaoui, Sousse, Tunisia, May 3-6, 2010, ISBN: 978-960-474-188-5, pp. 217-222

[5] Kurose J., Ross K., *Computer Networking: A Top-Down Approach, 5th Ed.*, Pearson Education Imc. 2010, ISBN 978-0-13-607967-5

[6] Симеонов С., Катъров П., *Съвременни компютърни комуникации*, АПН, 2002.

[7] Easley D., Kleinberg J., *Networks, Crowds, and Markets*, Cambridge University Press, 2010.

[8] Hennessy J., Patterson D., C*omputer Architecture, 5th Ed.: A Quantitative Approach*, Elsevier Inc., 2012.

[9] Germanov, V., Simeonov, S., Simeonova, N., Graphical Interface for Visually Impaired People Based on Solenoids, *John Atanasoff Celebration Days, International Conference "Robotics, Automation And Mechatronics"* RAM 2011, Sofia, 3–7 October 2011, I-17 – I-20.

[10] Simeonov S., Karastoyanov D, Simeonova N., Text and speech conversation technologies for helping visually impaired people, *John Atanasoff Celebration Days, International Conference "Robotics, Automation And Mechatronics"* RAM 2011, Sofia, 3–7 October 2011, I-13 – I-16.