International Journal of Media and Communication
2017; 1(1): 11-15
http://www.sciencepublishinggroup.com/j/ijmc
doi: 10.11648/j.ijmc.20170101.13

Virtual Process: Inside Approach to Understanding

Iuri Vitalijovych Koval

Department of Theoretical Cybernetics, Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

Email address:
smith@uis.kiev.ua

To cite this article:
Iuri Vitalijovych Koval. Virtual Process: Inside Approach to Understanding. *International Journal of Media and Communication.* Vol. 1, No. 1, 2017, pp. 11-15. doi: 10.11648/j.ijmc.20170101.13

Received: February 28, 2017; **Accepted**: April 6, 2017; **Published**: June 13, 2017

Abstract: This paper is devoted to virtual process notion investigation. Brief history of process notion present. Virtual process, multi-language program, process equivalence, code generalization, and code simplification notion are discussed. Simple example of behavior-equivalence and conditional behavior-equivalence proposed. Virtual process termination problem is discussed also.

Keywords: Virtual Process, Multi-Language Program, Process Equivalence, Code Generalization, Code Simplification

1. Introduction

Term ‘virtual process’ consist of two words. The main part essentially is ‘process’. The part ‘virtual’ is commonly used to extend the existing notion. The understanding of internal structure of virtual process is a task for this research. Usual processes may be named as atomic in further text of paper.

For beginning, let understand what the atomic process from internal point of view is. A good description in appropriate manner is in [1]. There one can see that the atomic process associated with some amount of memory, each atomic part of which is accessible and, in the same time, identifiable by the unique number. Usually, this number is named ‘address’. For abstracting from real memory, this can be named as ‘address space’. In such address space four main parts can be pointed: code, global/static data, heap and stack. This presentation shows an internal structure of atomic process. Some information about the atomic process is stored in the kernel memory and can be associated with record in the kernels’ process table. For determining if this information is crucial for the process notion, let look in the history of the process notion.

2. The History of the Process Notion

At the pre_operating_system times there are the job and the task notions. Both notions correspond to the program and the program execution at the same time. The nascence of operating systems demands and determines the separation of two phases of program existence. So, the term ‘process’ comes in. But, from the operating system point of view, the process is the record, mentioned above. Exactly – the number of this record, which is known now as a pid – process identifier. So, from this point of view, it is impossible to determine the internal structure of the process.

Authors of UNIX operating system provided two base primitives to operate with process. The first one is a fork and the second one is an exec. There is a part of the exec description form [2]:

> ‘*exec overlays the calling process with the named file, then transfers to the beginning of the core image of the file. ... There can be no return from the file; the calling core image is lost.*’

It must be stressed, that the pid of process didn’t changed.

In the other hand, the description of fork in [2] saes the next:

> ‘*fork -- spawn new process*
> *fork is the only way new processes are created. The new process’s core image is a copy of that of the caller of fork the only distinction is the return location and the fact that r0 in the old process contains the process ID of the new process. This process ID is used by wait.*’

The new pid appearance means the new process nascence. So, in the first case (exec), the code for the process changed. Without recognising by the operating system that the process is changed. In the second case (fork), the same code attached to the new process, as the new data also. So, the next

conclusion reached: the data is the crucial part of the process.

To give the additional arguments for this conclusion, let consider the description of the Emil Leon Posts' machine [3] and the Alan Mathison Turings' machine [4]. Both authors stated that the solution for a problem is the finite state of data (in the now_a_day terminology). Also, one of the main principles of current computer architecture states the same: the solving of the problem corresponds to obtaining the final data set from the beginning data set. The process of solving of the problem is a sequence of machine state changing.

In his paper [3] Emil Post uses the term 'process' in the sense that correlates with the current notion of process for the operating systems. So, with grate pleaser, the term process will be used to note the process of program execution.

3. The Process Equivalence

The next question to clarify is about process equivalence. The strictest equivalence of processes is code-equivalence. This one exists for two processes with the same code. This is almost useless equivalence, except the case of copyright. For problem solving this equivalence is unusable as fork primitive demonstrates. All other equivalences are considers on the memory states, or data sets in words of this paper. Two process are supposed to be behaviour-equivalent if they produce the same sequence D0, D1, … Dn for any D0, where D0 is beginning memory state and Dn – is the final memory state. This equivalence is so strict, that rare to exist in wild. The code refactoring is one possible and, I dare to say, very useful case for such equivalence. The behaviour term correlates with the Turings' machine moves and behaviour [4].

The less strict equivalence is conditional behaviour-equivalence which defined as the previous one, except the condition, that only for D0 from some set DS the sequences are the same. This equivalence is useful to produce the generalization or simplification of some code. Next example demonstrates such a case.

Table 1. *Simple function for factorial.*

Code	States		Action
int f (int a) {	f() a		f(2)
int r = 1;	2	r	int r;
while (a)		1	r=1;
r*=a--;		2	r*=a;
return r;	1		a--;
}		2	r*=a;
	0		a--;
	2		return r;

Function f() calculates factorial for its argument. During this calculations several memory states complete a sequence: {(a,2)}, {(a,2),(r,1)}, {(a,2),(r,2)}, {(a,1),(r,2)}, {(a,1),(r,2)}, {(a,0),(r,2)}. The result value is 2 and named by 'r'. In the table 1 only changing of values are shown. Identical states changing present in sequence. It seems clear, that for all non negative numbers f() provides correct answer. To protect f() from incorrect answer for the negative number the protective if statement must be added:

Table 2. *Protected function for factorial.*

Code	States	Action
int f (int a) {	f() a	f(0)
if(a <= 0)	0	
return 1;	1	return 1;
int r = 1;		
while (a)		
r*=a--;		
return r;		
}		

Protective if statement has such side effect, that the calculations for zero provides different states sequences for functions from tables 1 and 2. Nevertheless, both functions are behaviour-equivalent for the positive integer numbers.

The function from table 2 is a protected variant from incorrect incoming data of one from the table 1. Such case will be named as a code generalisation or a generalisation of code. So, the code generalisation is such a code transformation, which provides extending of DS set so, that the new code can be used without problem on it, and is behaviour-equivalent to the old code on DS. On the contrast, the code simplification is such code transformation that is behaviour-equivalent to the old code on DS but demand protection in calling code for data that extends DS.

The equivalence-by-result is such one, when only D0 and Dn must be the same for both processes. In such case the Dn note lost sense as n neither shows the number of data sets (memory states) in the sequence nor the number of step to perform the problem solution. This equivalence is the weakest among all mentioned before. But it is exactly one that is very useful for reengineering of programs. This equivalence is de-facto the tests set for the problem solution. And, as a result, tests set for the program. The more tests exist, the more complete problem specification is formed.

Conditional equivalence-by-result unites the notions of equivalence-by-result and conditional equivalence in obvious way.

At this point the notion of program must be clarified.

4. The Notion of the Program

According to the documentation for different programming languages it can be concluded that no common definition of program exist. The FORTRAN program [5] is a sequence of formulas, or instructions in modern terminology. The C program [6] is a set of files that each is a sequence of variable, function, and type declaration mixed with function prototypes. The Java program [7] is a set of classes. And so on. The common feature of all programs is that they are a text written according to some rules. The next question can be asked: is it possible to write a multi-language program? The first answer is no. But the multi-praradigmic programming languages exist [8]. But the library for one language can be used from another one. More over, syntax elements migrate from one programming language to other.

The main difficulty in combining different programming languages in one program is interpreting for one of them and compiling for other as a method of program execution. The other difficulty is dynamic memory using approach.

The answer for stated question now must be yes. Our time programming systems and platforms moves into direction of multi-language programs. This move is very slow and hardly visible, but it exists. It is understandable, that the best programming language impossible to develop. But every existing programming language is the best in particular case. Let us use the best choice for every problem and the summary result will be better.

Other question to be asked: is it possible to write a program that do program? The first answer is yes. But what is "to do program": write or execute? The difference between program and process become obvious. The program is written. The process is created. Prohibiting of writing process code makes impossible to write process code. So, a program must be written and then executed, namely the process created. The interpreted languages mostly ignored prohibiting to change the code. But mainly this means a possibility to write new parts of code and not erase of old code.

The answer for other stated question must be more than yes. It means that it's not only possible to modify code, but it must be used. It is clear, that this approach carry heavy difficulties, but there are no other way, then to solve them.

Here duality among the code and the data need to be clarified.

5. The Code and Data Duality

The another of the main principles of current computer architecture states that the code and the data can be placed in the common memory. The only part of computer that can distinct what is code and what is data is processor. But the same processor can write data and treat them as code (if this isn't forbidden just in case). So, it happened that data may be code and code may be data. Let look at simple arithmetic expression 1+2. Mostly everyone says, that 1 and 2 are data, and + is command. Lets write this in polish notation: +,1,2. In this case one can say that command 1 and then command 2 applied to data +. Of course, this is a trick. Bat take a look at HTML [9]. This is objects description. Does object description is a program? Many programmers say no for HTML. But why C++, or Java, object is a program? Because method or function main are present? And if they hidden the object becomes clear data? The main aim to disable code changing is to simplifying the process of the program correctness proving. This was helpful at the beginning of the compute age. But like AC is become more suitable then DC, like dynamic systems overtakes static system, like a virtual memory system displace physical RAM, code changing will take advantage over solid code. This is the main law of nature.

Other example of the code/data duality is data streams in channel. For TTY channel commonly used ESC symbol to transfer command. Why ESC symbol is present in data transfer mode and is not used in command mode? Why command may be mixed with data without requiring ESC or similar symbol? The answer is simple: command code is such symbol. In programming languages numerical data commonly not separated form operations (commands). But character or textual data are separated by any kinds of quotes. The only reason for that is impossibility to recognize what is data text and what is program text (command). So for code and data we use special marks to distinguish that such bit-sequence is data or code.

6. The Virtual Process

I can find for now only one publication with the term 'virtual process' in close sense [10]. But the authors limits the virtual process only for one (personal:) computer. In the [11] extending of virtual process notion proposed. That extension made with approach close to the Antony Hoares' approach [12] of the communicating sequential processes and other similar approaches. This approach concentrated on communication between processes or in other words interaction without the paying attention to the internal structure of such a process. Now such structure is the aim of this research.

As stated, the data is the main part of the virtual process. The process created when initial set of data created. At any step of data modification code attached to this data. This code perform next step modification. At any step code may be changed for other code or modified.

Where this data can be stored? This data stored at any memory system. The simplest way is to store them in files. Other possibility for today is data bases. In both cases code, that give possibility to store data is not part of virtual process. To explain this let look at the usual operating system process. The part of code, that implements input/output or other system operation is not a part of process code. Processor, which is now mix of code and hardware, also is not part of any process on computer.

In the present time cloude services also can be used to store data. Also any future data storage can be used.

As mentioned above the atomic process data stored in address space. Virtual process uses amount of different address spaces, but still need to have unified method for data identifying. As every address space usually uses the same set of addresses other method must be used. Naming is such an other method. Naming provide name space. The difference between address space and name space is like difference between natural and rational numbers. For address space neighbor relation exist as equivalence of next (previous) relation for natural numbers. There is no such relation for the rational numbers, because the rational number exist between any pair of them. The similar situation is for names. The word *a* less then the word *b* by lexicographical order. The word *ab* grater then the word *a* and less then the word *b*. Let suppose that character set limited to small Latin characters. The is a problem for a and aa. The is no any word between

them. Let use the extension of alphabet to solve this problem. All known to me alphabet at present time have fixed number of symbol. May be logographic script can produce infinite character set, but if we suppose usual coding for such system this can't help solve the problem. To solve the problem both end infinite character set must be used. One possible solution is to extend code set for symbols with codes like integer numbers. Other solution is to extend code set by 1 divided by symbol code. The second approach is good because all codes are positive. But it is impossible to generate the other reverse code for zero and one coded symbols. This can be solving by changing formula to 1/(2+code). It's work, but not very nice.

The first approach is correlates with complement code. This means that the extending of computer architecture size makes possible to extend the character code space in both direction. In fact it doesn't matter what approach is used because standard HTML notation for symbols can be extended and used [13]. It can be ## to determine that other code used. What that symbol looks like? Is it necessary? Are you know what looks symbol €? Can you work with this symbol without knowing what it looks like? It is not important what symbol looks like if we can use it in another way.

Let return to *a* and *aa*. Now new symbol can be added to alphabet that less then *a,* and so a&##(a-1); is less then *aa* and grater then *a*. Strange symbol &##(a-1); used? Can you understand what I mean? If yes, there is no problem. If no, this is one more extension to symbol writing. Arithmetic expression used to generate new symbol code. This is one more demonstration of code/data duality.

Now it can be stated that we have name space. Each constant of such name space must have a part that determine way in which data stored (like URL[14]) and a part with name for data. Current URL specification provide this possibility. This is the way to name data. When access to the uninitialized data happened zero value returned.

The code for virtual process is the usual data for all system that support virtual process. As a result, it can be written, modified and executed. Transmition of code among different executing system is usual data transferring in the network. So the network is become the native environment for the virtual process.

7. Finishing and Pausing of the Virtual Process

If the virtual process is created, it is very hard to destroy it. Moreover, it is possible to multiply any existing virtual process to any number of it. Virus-like technologies very clearly demonstrate this. To comparison let look at usual process. To stop it, it is enough to turn computer off (since no magnetic ram used). This is the last, but very effective possibility. Programmers for many decades try to find way to prevent this. The first step was to swap inactive process. The next one is to sleep processor and, as a result, all processes. The last achieved step is hibernating. But hibernating does not provide the same environment for process. For example, all network connections losts. Not only for drivers problem, but also for timeout problem.

Operating system can be cloned by copying its data. Viruses can mutate – modifying their code. The only way to destroy virtual process is to destroy all copies of all its data. Pausing or hibernating is not a problem for virtual process.

8. Result

To solve the research task internal structure of virtual process investigated. It is determined that the main part of the virtual process is data sets for it. The name space is proposed as a mechanism to store such data. Concrete data structure is a material for the father investigation.

9. Conclusion

In this paper brief history of process notion present. Virtual process, name space, both end infinite character set, multi-language program, process code-equivalence, behavior-equivalence, conditional behavior-equivalence, equivalence-by-result, conditional equivalence-by-result, code generalization, and code simplification notion are discussed. Simple example of behavior-equivalence and conditional behavior-equivalence proposed. Virtual process termination problem is also discussed.

Acknowledgements

I want to say my gratitude to all my teachers, even those who never now that they teach me. I want to say my gratitude to all my students, even those who never ask my permission to read my paper. The first gives me knowledge and the second gives me inspiration. In common we produce the virtual process of science.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, OPERATING SYSTEM CONCEPTS with JAVA, 6th ed., JOHN WILEY & SONS, INC., USA, p. 1251, 2004.

[2] K. Thompson, D. M. Ritchie, UNIX PROGRAMMER'S MANUAL, Bell Labs, USA, p. 194, November 1971, https://www.bell-labs.com/usr/dmr/www/1stEdman.html.

[3] E. L. Post, Finite Combinatory Processes - Formulation 1, The Journal of Symbolic Logic, Vol. 1, No. 3. (Sep., 1936), pp. 103-105, http://www.jstor.org/stable/2269031.

[4] A. M. Turing, On computable numbers, with an application to the entscheidungsproblem, Proceedings of the London Mathematical Society, 2nd series, vol. 42, pt. 3 (November 30, 1936): pp. 230-240; 2nd series, vol. 42, pt. 4 (December 23, 1936): pp. 241-265; 2nd series, vol. 43, pt. 7 (December 30, 1937): pp. 544-546.

[5] Programming Research Group, The IBM Mathematical FORmula TRANslating System, FORTRAN, Preliminary Report, IBM, New York, USA, p. 29, November 10, 1954.

[6] D. M. Ritchie, C Reference Manual, Bell Telephone Laboratories, Murray Hill, New Jersey, USA, p. 31, 1974, https://www.bell-labs.com/usr/dmr/www/cman.pdf.

[7] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, The Java® Language Specification Java SE 7 Edition, Oracle America, Inc., USA, p. 670, 2013-02-28, http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf.

[8] A. A. Letichevsky, J. V. Kapitonova, V. A. Volkov, A. Chugajenko, V. Chomenko, and V. Peschanenko, Algebraic programming system APS (user manual), Glushkov Institute of Cybernetics, National Acad. of Sciences of Ukraine, Kyiv, Ukraine, p. 43, May 22, 2008.

[9] HTML5 Tutorial, https://www.w3schools.com/html/.

[10] D. Brugali, M. Torchiano, Software development: case studies in Java, Addisson-Wesley, p. 653, 2005.

[11] Iu. V. Krak, Iu. V. Koval, and A. B. Stavrovskyi, Virtual process: definition and application for gestures interface system creation, Bulletin of Taras Shevchenko National University of Kyiv, Series Physics & Mathematics, vol. 1, pp. 141-144, 2015.

[12] C. A. R. Hoare, Communicating Sequential Processes, p. 260, May 18, 2015, http://www.usingcsp.com/cspbook.pdf.

[13] HTML Symbols, https://www.w3schools.com/html/html_symbols.asp.

[14] HTML Links, https://www.w3schools.com/html/html_links.asp.