An Operating System Model with the Process Control at the Level of Programming Statements

Juraj Štefanovič*
stefanovic@fiit.stuba.sk

Abstract: Our goal is to create easy modeling framework of operating system, which bears the possibility to simulate various control of parallel running processes and various system architectures, without taking in mind the bottom system hardware and the management of main and secondary (filesystem) memory. The modeling philosophy here is to have one type of building block, the virtual machine which interprets higher programming language statements instead of the simulator/emulator of hardware processor. The control of processes is shifted from the level of discrete atomic machine instructions to the level of higher programming language statements.

Key Words: Operating system modeling, process control, virtual machine

1 Introduction

Several years we use Nachos [1,2], the model of operating system for student's practices and case studies. This model in C++ bears an easy environment to build, modify and understand the control and work of concurrent processes, memory management and file system. This kind of model is able to exist as an user process, because the real kernel of operating system is not user-accessible (on public domains) and some accessible real kernels (Linux) are for more experienced people only to cope with.

Besides the Nachos, there exist various more or less favorable models/kits of operating systems with different levels of theoretical or practical use. The Flux OS Toolkit [3,4] was created with a goal to get a set of reusable components with well defined function and interface (boot loader, kernel code, drivers...). This should help to the skilled experimentators they don't need to write everything from scratch or copy and paste from many other implementations. The 2K system [5,6] was created with serious goal to bear an open–source distributed adaptible operating system, which replaces the idea of classical "machine–centric" management with the "network–centric" model. The possibility of distribution is an important feature in the contemporary world of networks and mobile computing. The Plan9 system [7,8] was created with similar purposes having building blocks, pervasive computing possibility and unified access to the files and resources.

The MOSES2 [9,10] environment is a simple model of shared hardware and user processes, where the kernel emulator programs can be easy created in C-language by students. The MTOPS [11] is another different model, which simulates an easy CPU with small instruction set and connected memory. Here the user programs are written in those assembly instructions

* Faculty of Informatics and Information Technology, Slovak University of Technology, Ilkovičova 3, SK-842 16 Bratislava, http:/www.fiit.stuba.sk/~stefanovic

and the kernel of operating system is to be extended by student homeworks. The article [12] describes another hardware simulator with the model of operating system; it is prepared in 8 various levels of complexity for teaching purposes. The article [13] describes another model of "System/161" – hardware simulator with debugging support and "OS/161" instructional operating systems running in it.

This short introduction shows that besides the "habitual" operating systems (allthough they evolve too) there exist an amount of various implementation projects of special purposed operating systems in range from pure experimental toy–models to the perspective prototypes. The mentioned examples show that modeling and prototyping of the operating systems is focused (not only) to these features:

- an easy environment to build and understand the model of operating system, running as an user process in real computer, independent on any fixed hardware structure
- virtual processor - an emulator of Central Processing Unit hardware, to model the control of the machine instructions flow
- clear interfaces of system components, to be easy understood

Our attempt is to create a new model, which will try to satisfy these features above and it will be useful for study and experimental purposes. Main idea is to use the virtual processor working not at the level of machine instructions, but on the higher level of instruction set, interpreting the scripts they are written in higher language – the C-language here.

2 Placing the Context Switch to the Higher Level

The classical notion of operating system is a software extension of some hardware (processor, memory and interfaces), hence the operating systems were built around the underlying hardware principles, having the legs in wired architecture and the head in user interface. One of the basic low-level functionalities in the operating system kernel is the management of concurrent processes, based on the context switch between them, Fig.1.

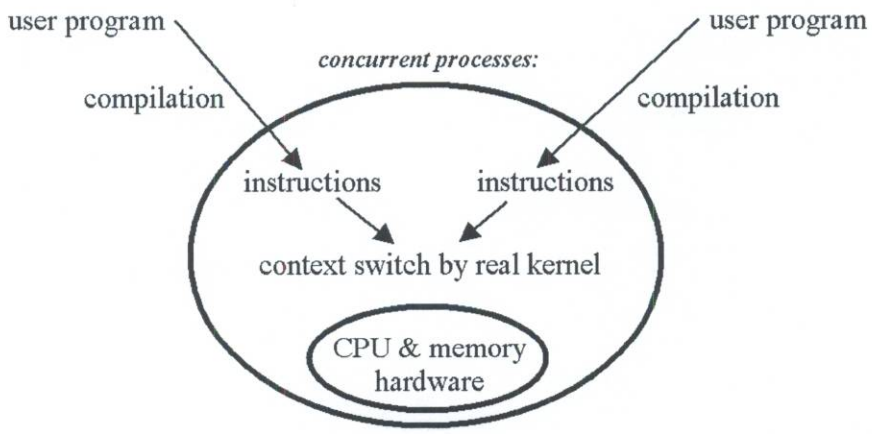

Fig.1. The core of process management in operating system.

The creation of Nachos and other similar modeling frameworks of operating systems had to cope first with the problem how to make this low-level functionality being accessible to the study and experiments. The realisation of Nachos is using MIPS simulator of RISC processor,

which is performing the specially prepared program instructions (compiled from C-language). In this case, the experimentator can model and perform an explicit context switch of the processes in his own user environment only, Fig.2.

The processes are composed from low-level machine instructions, but at the higher level they are composed from programming-language instructions/statements. Instead of the processor simulator/emulator, the interpreter of higher-level statements (scripts) can be used and the concurrent management of the processes can be performed and studied at this level. Hence our idea is to model the control and switching of the concurrent processes (simple user programs) to this higher abstraction level, Fig.3.

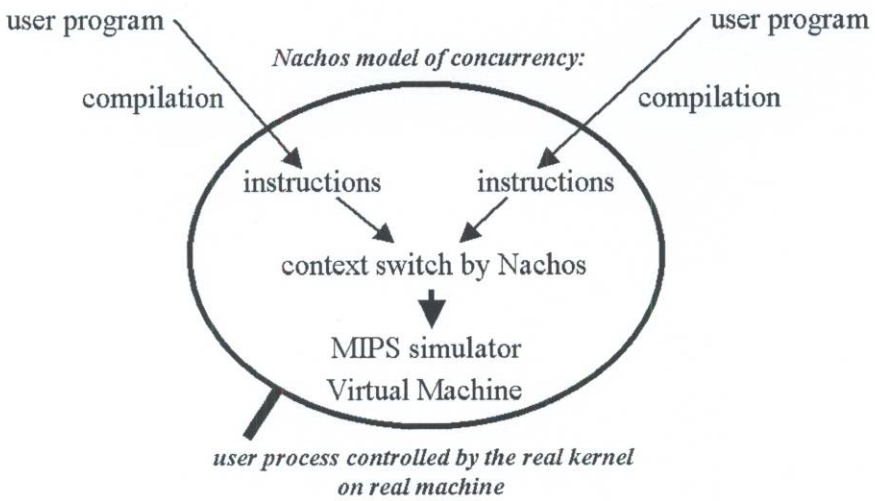

Fig.2. The process management simulated by Nachos.

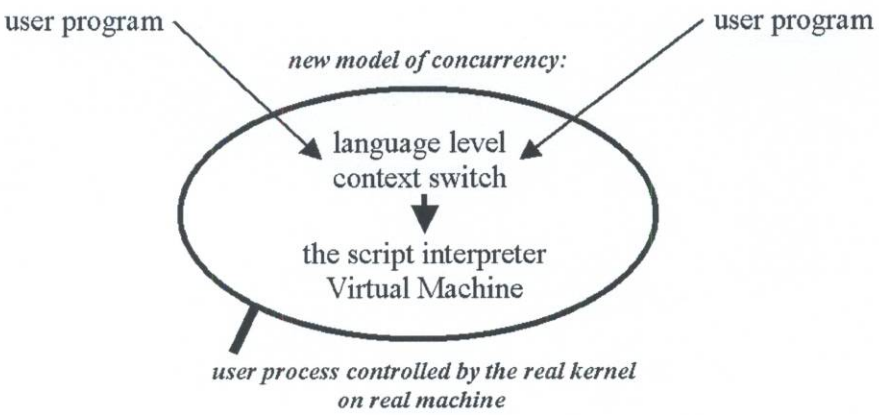

Fig.3. The proposed higher level to place context switch.

The advantages of this mentioned modeling paradigm:

- the model/the system is independent on the hardware bottom (no CPU emulation)
- the compiling stage of user programs (scripts) does not exist anymore, the shell-level programming and user process-level programming can be more closer than usual

- the relationship of this model to the hardware is lost: a possibility to create distributed system modules on various communicating platforms

The disadvantages:

- the granularity of process control is not very fine (atomic instructions are here the whole program statements or the parts of scripts and not the machine instructions)
- the relationship of this model to the hardware is lost
- the compilation is hidden in the interpreter engine, the step-by-step interpretation of single statements (of C-language indeed) can be complicated and constrained
- slower run of processes
- by using not the compiler but the interpreter of statements, the code of controlled processes must be rewritten to get run-able structure

Instead of the hardware central processing unit (CPU) simulator in Nachos, we are going to create a virtual machine, which will interpret the higher programming language statements so the processes can be controlled at the statement-timestep granularity.

3 The Virtual Machine Framework

The basic framework of our proposed model is the virtual machine, which is able to interpret some set of programming statements, Fig.4. The one question is what should be this set of statements and the other question is the relationship to the system memory. An interface of the virtual machine can be easy:

```
iostream *run_VM_oneStep (char *statement, int thread);
```

The user lets to run the machine for one step to read and to interpret the program `statement` and to make some behavior at the input and output data streams. The `thread` integer number is a possibility, that the statements can belong to several different threads or processes they run on the one virtual machine. This easy model lets the control of processes to the user (`fork, yield, ...`), or this model can be more complicated and it can incorporate some control inside. The set of statements/instructions is an reduced subset of C-language to fulfill our modeling goals:

- statements to create/delete data objects (variables, arrays, files)
- control structure statements (if/then/else)
- loop statements – their interpretation needs some control behavior outside
- data manipulation statements (a=b+c)
- input/output statements (create a stream, read/write to it)

The relationship to the system memory can be solved as an memory encapsulation into the virtual machine entity. There is no work with the notion "address", the machine is working with named and typed variables. The one type of variable can be a file, hence the difference between main memory and filesystem is removed:

```
run_VM_oneStep ("openfile("myfile","rw")", 1);
run_VM_oneStep ("openvariable("x","int")", 1);
```

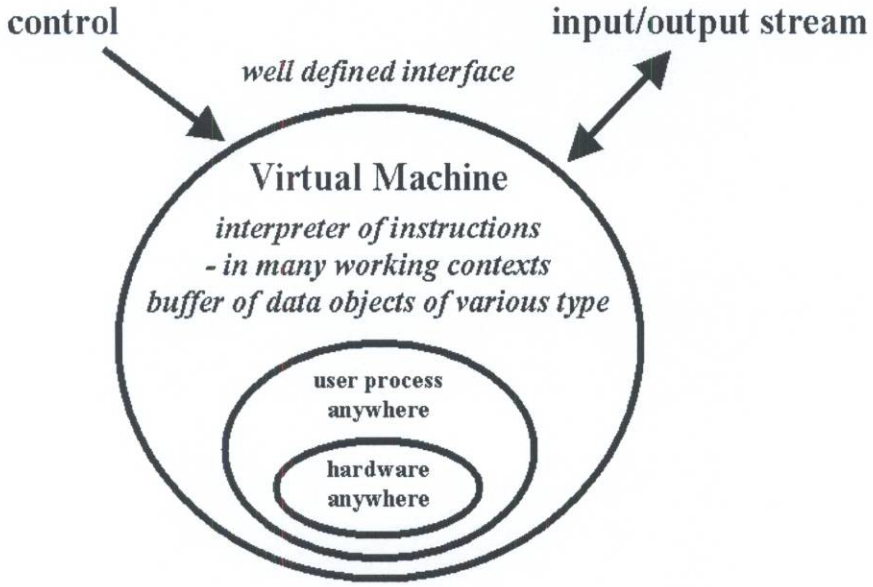

Fig.4. The concept of virtual machine at the level
of programming language statements/instructions.

As an example, here are two running threads on the virtual machine, they are synchronised with semaphores. They were written in C-language:

```
Thread1()
 {
 for(int i=0;i<2;i++){
   x = i;
   up(a);  // semaphore a: signal
   down(b);// semaphore b: wait
   }
 }
```

```
Thread2()
 {
 for(int i=0;i<2;i++){
   down(a);
   y = x + 1;
   up(b);
   printf("%d",y);
   }
 }
```

Using these two threads as the concurrently running processes by our modeling framework they are rewritten, still in C-language, to the form of statements they can be interpreted line by line by the mentioned virtual machine–processor:

```
i=0;              // Thread1
while(1){         // label-statement
x = i;
up(a);
down(b);
i++;
if(i==2)break; // ignore until '}'
}                 // goto label
```

```
j=0;    // Thread2
while(1){
down(a);
y = x + 1;
up(b);
print(y);
j++;
if(j==2) break;
}
```

Some user can write his own process scheduler, which will read the process statements and put them to the virtual machine model (**run_VM_oneStep**). The work of this scheduling algorithm is shown in following example:

```
initialisation:

  stream = run_VM_oneStep ("stream(st)",2); // return pointer
  signal = run_VM_oneStep ("control",1);    // return pointer
  signal = run_VM_oneStep ("control",2);

introductory statements:

  run_VM_oneStep ("int i;",1); // create this variable, thread1
  run_VM_oneStep ("int j;",2); // create this variable, thread2
  run_VM_oneStep ("int x;",2);
  semaphore(a) initialise it   // my control mechanism outside
  semaphore(b) initialise it

concurrent (switched) running of two threads 1,2:

  1  run_VM_oneStep ("i=0;",1);
  1  run_VM_oneStep ("while(1){",1);     // save label, thread1
  2  run_VM_oneStep ("j=0;",2);
  2  run_VM_oneStep ("while(1){",2);     // save label, thread2
  2  down(a) stop the second thread
  1  run_VM_oneStep ("x=i;",1);
  1  up(a) run the second thread
  2  run_VM_oneStep ("y=x+1;",2);
  1  down(b) stop the first thread
  1  run_VM_oneStep ("i++;",1);
  ....
  2  run_VM_oneStep ("print(st,y);,2");
  1  run_VM_oneStep ("if(i==2)break;,1");// ignore next statements
  1  run_VM_oneStep ("}",1);     // goto label from stack, thread1
  ....                           // ignore off, thread1
```

Using the **run_VM_oneStep** model of C-statement interpreter is shown in schematic in Fig.5. The **underlined** statements belong to the implementation of the user, which creates his own model of system control. The *italic* statements are producing easy signals for the control flow outside.

Fig.5. Using the model of Virtual Machine.

4 Teaching and Experimental Purposes

An easy virtual machine entity, as shown in previous text, can be used as a building block to make various models of synchronisation and architecture composition, Fig.6. Besides the Nachos and some partial C-language based models, we prepare a set of assignments for our students to work practices with this presented system philosophy. The practices in labs with many individual students will help to determine an optimal reduced set of statements the virtual machine entity will perform. The building blocks of system (Fig.6) can be generic entities (virtual machines as shown above), or they can bear specialised functionalities (performing database statements, or VRML visualisation/rendering statements, or otherwise). By the experiments, distributable virtual machine entities can be implemented within network servers or can be incorporated to the www browsers.

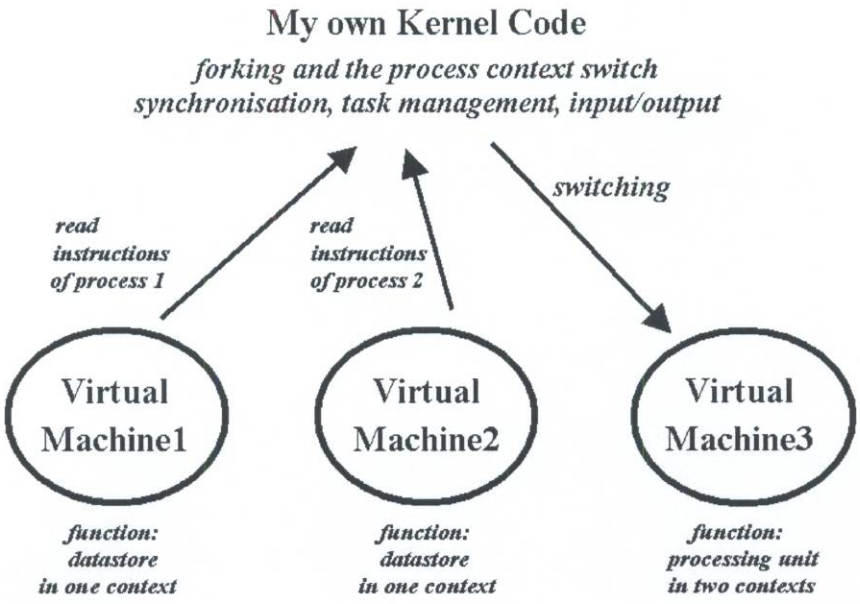

Fig.6. An example of joining the building blocks.

Using the Nachos model in our teaching of the operating systems programming has the one disadvantage: the course is 12 weeks long only and the students have various level of programming abilities. Beginning to study the whole system together (process control, memory management and filesystem) is difficult, allthough the study can be well prepared in partial tasks. The Nachos can be well installed to run and the student gets full information where to and what, but a bulk of C-code and files around him is very noisy at the beginning of the work. The encapsulation of this noise into the one kind of entity (the virtual machine) with clear defined interfaces is user friendly. Moreover, this philosophy is perspective at all, because the paradigm of operating system is not only the extension of bottom hardware, but the encapsulation of the whole system into a box with easy and clear user interface.

5 Conclusions

The virtual machine entity is presented here as an easy one type of building block to create simple models of process control and parallelism. The main idea is to control the process flow at the higher level of its programming statements, which causes raw granularity of process control and losen relationship of our model to the hardware, but the model has well defined interface at the abstract behavioral level. An application of this modeling philosophy will be in our teaching work, with the experiments to create portable and distributable building blocks and incorporate the building blocks with specialised behavior.

This work has been supported by the Grant Agency of Slovak Republic, grant No. VG1/3103/06.

References

1. Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson: *The Nachos Instructional Operating System.* Technical Report UCB//CSD-93-739, University of California, Berkeley. April 1993
2. This is the planet where Nachos rule: http://www.cs.washington.edu/homes/tom/nachos/ (accessible in February 2006)
3. Bryan Ford, Kevin Van Maren, Jay Lepreau, Stephen Clawson, Bart Robinson, and Jeff Turner: *The Flux OS Toolkit: Reusable Components for OS Implementation.* In Proc. of Sixth Workshop on Hot Topics in Operating Systems. May 1997, pp. 14-19
4. Utah: The OS Kit Project: http://www.cs.utah.edu/flux/oskit/ (accessible in February 2006)
5. Fabio Kon, Roy H. Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros: *2K: A Distributed Operating System for Dynamic Heterogeneous Environments.* Department of Computer Science, University of Illinois at Urbana-Champaign, 1999
6. 2K: A Component-Based Network-Centric Operating System for the Next Millenium http://srg.cs.uiuc.edu/2K/ (accessible in February 2006)
7. Francisco J. Ballesteros, Gorka Guardiola, Enrique Soriano, and Katia Leal: *Traditional Systems can Work Well for Pervasive Applications. A Case Study: Plan9 from Bell Labs Becomes Ubiquitous.* Third IEEE International Conference on Pervasive Computing and Communications PerCom'05, 2005, pp. 295-299
8. Plan9 from Bell Labs: http://www.cs.bell-labs.com/plan9/ (accessible in February 2006)
9. Robert E. England: *The virtual machine and user process model used in MOSES2: a microcomputer operating system environment simulator.* Journal of Computing Sciences in Colleges, Volume 17, Issue 2, December 2001, pp. 301 – 309
10. Robert E. England: *Teaching concepts of virtual memory with the MOSES2 microcomputer operating system environment simulator.* Journal of Computing Sciences in Colleges, Volume 20, Issue 6, June 2005, pp. 84-91
11. Suban Krishnamoorthy: *An experience teaching operating systems course with a programming project.* Journal of Computing Sciences in Colleges. Volume 17 Issue 6, May 2002, pp. 25 - 38
12. John Dickinson: *Operating systems projects built on a simple hardware simulator.* ACM SIGCSE Bulletin, Proceedings of the thirty-first SIGCSE technical symposium on Computer science education SIGCSE '00, Volume 32, Issue 1, ISSN:0097-8418, year 2000, pp. 320-324
13. David A. Holland, Ada T. Lim, Margo I. Seltzer: *A new instructional operating system.* ACM SIGCSE Bulletin, Proceedings of the 33rd SIGCSE technical symposium on Computer science education SIGCSE '02, Volume 34, Issue 1, ISSN:0097-8418 , year 2002, pp. 111-115