NON-DETERMINISM SUPPORT IN THE FIBEROS EXOKERNEL

Milen Loukantchevsky,
Assoc. Prof., PhD, IEEE Member, ACM Member
Department of Computing, University of Ruse, BG
E-mail: mil@ieee.org

Nikolay Kostadinov, PhD
Department of Computing, University of Ruse, BG
E-mail: nkostadinov@ecs.uni-ruse.bg

Hovanes Avakyan, PhD
Department of Computing, University of Ruse, BG
E-mail: havakian@ecs.uni-ruse.bg

Abstract: Non-determinism is a fundamental concept in automata theory, algorithms and parallelism. Dijkstra’s guarded commands as well as Hoare’s CSP alternative primitives are means for overcome of the intrinsic for parallel systems non-determinism.

The fiberOS is educational non-preemptive cooperative exokernel. Represents an implementation of a simple CSP machine using Windows fibers. The key CSP objects such processes and channels are supported as well as CSP parallel, alternative and communications primitives.

This paper presents implementation of CSP alternative command in fiberOS. 2-channel and n-channel versions of this command are supported. The first version is for introductory purposes, while the second one is STL (C++ Standard Library) based. Both are truly non-deterministic and could be used to profound study of that fundamental concept.

Keywords: Alternative command, C++, CSP, Exokernel, Non-determinism, Parallelism.

INTRODUCTION

Non-determinism is a fundamental concept well known in automata and algorithm theories [1, 6]. Traditionally in these areas non-determinism is considered as result of missing predefined transitions between system states. But in parallel systems where non-determinism is an intrinsic feature, it is a result of random choice of transitions [6]. Dijkstra’s guarded command is a mean to control and effectively use of the non-determinism in parallel systems [2, 3]. Guarded command is evolved further by C.A.R. Hoare in his theory of Communicating Sequential Processes (CSP). The CSP equivalent of Dijkstra’s guarded command is alternative command [4, 7].

Let us have as an example the next alternative command containing two alternatives

\[ \{G_1 \rightarrow P_1 \triangledown G_2 \rightarrow P_2\}, \]

where \( G_1 \) and \( G_2 \) are guards of the processes \( P_1 \) and \( P_2 \) respectively. In case the events \( e_1 \) and \( e_2 \) do not occur simultaneously, then both guards will not evaluate as true in the same time. And we got the deterministic machine transition diagram (fig. 1).

![Fig. 1. Deterministic Machine Transition Diagram](attachment://image.png)

Suppose our guards use CSP communications command \(<?>\) as follows

\[ \{P_1 ? x \rightarrow SKIP \triangleleft P_2 ? x \rightarrow SKIP\}. \]
In this case the event $e_1$ corresponds to receipt of a message by process $P_1$ and $e_2$ – by process $P_2$. Hence these events are not mutually exclusive in contrast with the deterministic case and we have got the non-deterministic machine transition diagram (fig. 2).

![Fig. 2. Non-deterministic Machine Transition Diagram](image)

From the above follows that a process with two or more input channels included in one common alternative command is non-deterministic.

### I. CSP COMMUNICATIONS PRIMITIVES AND THEIR FIBEROS SUPPORT

CSP has two communications primitives – receive of message marked with `<?>` and send of message marked with `<!>`. Their fiberOS equivalents [7] are respectively

\[
\text{bool RECV(CHAN* chanIn, CHAN_MSG* dst);} \\
\text{bool SEND(CHAN* chanOut, CHAN_MSG* src);} \\
\]

The CSP communication primitives and their fiberOS implementations use strong bilateral synchronization called rendezvous.

![Fig. 3. Rendezvous of communicating processes](image)

In fig. 3 are shown two scenarios we could have with rendezvous. In both, the first process which came to the point of communication is blocked until another come to that point [5]. As result we have got the simplest possible communication channel – 1:1, unidirectional without buffering.

### II. CSP ALTERNATIVE COMMAND IMPLEMENTATION IN FIBEROS

The general form of CSP alternative command has $n$ alternatives

\[
\{G_1 \rightarrow P_1\square G_2 \rightarrow P_2\square \cdots G_n \rightarrow P_n\} 
\]

and can be considered as generalization of deterministic `<if-else>` operator. Only if all guards are mutual exclusive we will have `<if-else>` deterministic behavior, otherwise the behavior observed will be non-deterministic.

As distinct from `<if-else>` all guards (conditions) are with equal priority and are checked not successively but at once. The fiberOS implementation of CSP alternative command follows its semantic and is shown in fig. 4.

First we should build the list $L$ with all guards evaluated true.
Secondly, it is checked if that list $L$ is empty or not. If the list $L$ is not empty, the index $k$ of the selected guard contained in $L$ is determined by uniform distributed random function $\mathcal{U}$. Then the control is given to the process $P_k$. Otherwise, if the list $L$ is empty, the alternative command terminates.

For educational purposes there are two fiberOS primitives (2-channel and n-channel):

\begin{equation*}
\begin{align*}
\text{int } ALT&(CHAN* chanIn1, CHAN* chanIn2, CHAN_MSG* dst); \\
\text{int } ALT&(vector < CHAN* > & vChanIn, CHAN_MSG* dst);
\end{align*}
\end{equation*}

The 2-channel $ALT()$ is appropriate for educational purposes and introduction to the matter. It is used in Section 3 of this paper. The $n$-channel $ALT()$ as generalized form is preferable for practical purposes. It gets the list of input channels $vChanIn$ in STL (C++ Standard Library) container of vector type.

**III. FIBEROS TESTBED OF NON-DETERMINISM**

In this section is presented an exemplary fiberOS testbed for non-determinism exploration. The 2-channel $ALT()$ is selected according to common sequence of consideration.

We choose the simplest parallel system with non-determinism. It is a 2:1 "producer-consumer" system and consists of three processes – the producers $P_1$ and $P_2$ and the consumer $Q$:

\begin{equation*}
\begin{align*}
\frac{\{P_1||P_2||Q\},}{P_1 = P_2 = P = \ast \{Q!msg \rightarrow \Delta \rightarrow \text{SKIP}\}_N}, \\
Q = \ast \{P_1?x \rightarrow \text{write}(x) \rightarrow \text{SKIP}\Box P_2?x \rightarrow \text{write}(x) \rightarrow \text{SKIP}\}_{2\times N}.
\end{align*}
\end{equation*}

The producers $P_1$ and $P_2$ are identical and execute $N$ cycles before termination. In each working cycle they send to $Q$ the message $msg$. The consumer $Q$ is non-deterministic by specification – contains two alternatives which guards are commands for communication. It executes $(2 \times N)$ cycles which equals to the total number of messages sent by two producers.

Two scenarios are investigated: deterministic and non-deterministic one.

First scenario assumes deterministic implementation of CSP alternative command by the means of <if-else> operator. Let suppose $Q$ checks first for message from $P_1$ and after that for messages from $P_2$.

The result we eventually get in the first scenario is more than strange for a newcomer. The consumer $Q$ communicates only with $P_1$ (fig. 5) ignoring any messages sent from $P_2$. And the parallel system finally encounters mutual deadlock - the consumer $Q$ continues to wait for message...
from $P_1$ even after its completion and still ignores the messages from $P_2$. Thus, only process $P_1$ stops but $P_2$ and $Q$ do not.

![Diagram of processes $P_1$, $P_2$, and $Q$.]

Fig. 5. Left channel preferable under deterministic case

In result, the parallel system $\{P_1|P_2||Q\}$ do not terminates (fig. 6).

![Diagram of mutual deadlock.]

Fig. 6. Mutual deadlock under deterministic case

The situation will turn just reflective if $Q$ checks first for message from $P_2$ and after that for messages from $P_1$. And the result will be the same – mutual deadlock of our parallel system of three processes. The reason in both cases is in unconformity of deterministic <if-else> with process communication of type rendezvous.

The second scenario assumes usage of fiberOS primitive ALT() which implements non-deterministic semantics of CSP alternative command (fig. 4).

![Diagram of deadlock free communication.]

Fig. 7. Deadlock free communication under non-determinism

In this case the consumer $Q$ receives regularly at equality basis all messages sent by the two producers. And eventually the system completes its execution deadlock free (fig. 7).

![Exemplary execution of non-deterministic parallel machine.]

Fig. 8. Exemplary execution of non-deterministic parallel machine under fiberOS
A snapshot of the second scenario execution under fiberOS is shown in fig. 8. The processes Task 1, Task 2 and Task 3 correspond to P₁, P₂ and Q respectively. At the instant given P₁ and P₂ produced already 33 messages. And Q consumed 33 from P₁ and 32 from P₂ yet. Because of random-driven nature of decision there is not and would not be any guarantee that messages from P₁ or P₂ will overtake.

The only guarantee is that Q will consume eventually all messages generated and no any deadlock will happen. And this is exactly what we need.

CONCLUSIONS

Leading methodological principle of science is to direct investigations to the essential contradiction of the subject domain. At the area of computer systems development this is the contradiction between the explicit parallelism of current computer architectures and the sequential thinking.

The fiberOS non-preemptive cooperative exokernel is developed in the Department of Computing of Rousse University as an educational tool. It is used at the first phase in the course on “Parallel Computer Systems” along with CSP as fundamental mathematical model for specification of parallel systems.

The authors adhere to the next phases in practical teaching of the base concepts and mechanisms in the field of parallel systems:
- One node machine with multitasking (Windows/x86 platform, fiberOS executive, C programming language);
- Conventional multi node machine with multitasking (X51/MCS51 platform, X51mp executive, C programming language);
- Multi node machine with direct support of the parallelism at architectural level (SMT/DLP core xCORE, no executive required, XC parallel programming language).

At the first of above phases the goal set up is to reach well understanding of processes as active objects, channels as means of communications, parallel command, synchronization and alternative command as specific for parallelism exploitation. This paper presents authors’ approach of introduction of means to control the intrinsic for parallel systems non-determinism.

REFERENCES


