Journal of Computer Science 5 (7): 493-500, 2009
ISSN 1549-3636

Multi Microkernel Operating Systems for Multi-Core Processors

Rami Matarneh
Department of Management Information Systems,
Faculty of Administrative and Financial Sciences,
Al-Isra Private University, Amman, P.O. 11622, Jordan

Abstract: Problem statement: In the midst of the huge development in processors industry as a response to the increasing demand for high-speed processors manufacturers were able to achieve the goal of producing the required processors, but this industry disappointed hopes, because it faced problems not amenable to solution, such as complexity, hard management and large consumption of energy. These problems forced the manufacturers to stop the focus on increasing the speed of processors and go toward parallel processing to increase performance. This eventually produced multi-core processors with high-performance, if used properly. Unfortunately, until now, these processors did not use as it should be used; because of lack support of operating system and software applications. **Approach:** The approach based on the assumption that single-kernel operating system was not enough to manage multi-core processors to rethink the construction of multi-kernel operating system. One of these kernels serves as the master kernel and the others serve as slave kernels. **Results:** Theoretically, the proposed model showed that it can do much better than the existing models; because it supported single-threaded processing and multi-threaded processing at the same time, in addition, it can make better use of multi-core processors because it divided the load almost equally between the cores and the kernels which will lead to a significant improvement in the performance of the operating system. **Conclusion:** Software industry needed to get out of the classical framework to be able to keep pace with hardware development, this objective was achieved by re-thinking building operating systems and software in a new innovative methodologies and methods, where the current theories of operating systems were no longer capable of achieving the aspirations of future.

Key words: Microkernel, multi-microkernel, multi-core processors, inter-process communication

INTRODUCTION

During the past decades there have been significant developments for the operating systems, began with simple structure and end with large and complex structure, although the design and implementation of operating system, not solvable, but some approaches have proven successfully[1].

As the kernel is the fundamental part of an operating system which implements a set of hardware abstractions that provide a clean interface to the underlying hardware, all developments focused on its design which is vary in three broad categories: Monolithic kernels, Microkernel and Exokernels[2,3]. Monolithic kernels are a mixture of everything the OS needed: Inter-process Communication (IPC), file systems, memory management, without much of an organization (Fig. 1). Newer monolithic kernels have a modular design, in which kernel runs in kernel mode and the processes run in user mode on top of the kernel. Such design offers adding and removal of services at run-time.

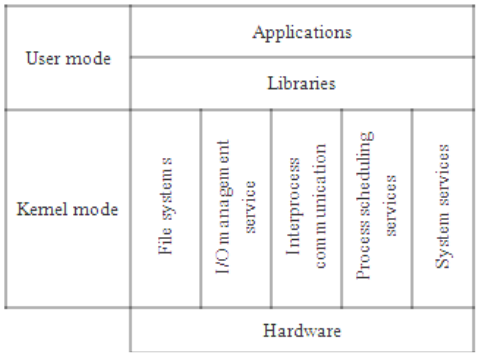

Fig. 1: Structure of monolithic kernel

Microkernel design usually provides only minimal services by putting a lot of operating system services such as file systems, device drivers (Fig. 2), user interface and protocol stacks in separate processes running on top of the microkernel and can be started or stopped at runtime to makes the kernel smaller and flexible.

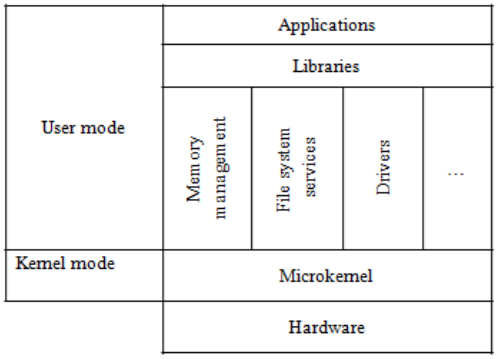

Fig. 2: Structure of Microkernel

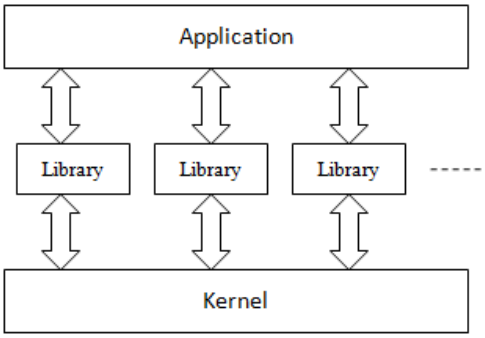

Fig. 3: Structure of Exokernel

Exokernel accompanied by library operating systems, which provide application developers with the conventional functionalities of a complete operating system (Fig. 3). This approach lets user programs override the standard code exported by the system and the kernel and leads to very fast operation but weak safety.

Microkernel design challenge: The big idea of microkernel is that the kernel can be split up into independent parts called servers, which communicate with each other and applications through Inter-Process Communication (IPC) via message passing. This architecture is actually a client-server; processes (clients) can call operating system services by sending requests through IPC to server processes[4].

But it seems that the reality is slightly different, where the developers of microkernels have not agreed on what services the microkernel should provide, every developer has its own perspective. These different perspectives led to have different versions of microkernels. For example Windows NT allows device drivers to run in kernel mode for reasons of efficiency, while Mach and Chorus microkernels keep the device drivers outside the kernel[5,6].

Such change in Windows NT led to replace message passing by system call, which means a fundamental change in microkernel architecture, because of this Windows NT considered not a true microkernel.

The main goal of a microkernel system is to keep it small as possible by following the pure microkernel doctrine which holds that all nonessential services should run in the processor's non-privileged mode[7]. To achieve this goal we must determine which services should be contained within the kernel that cannot be placed elsewhere, or that its presence outside the kernel would be costly.

In general the following represent essential but not definitive list of services that should be contained within the microkernel:

- Short-term scheduling
- Low-level memory management
- Inter-process communication via message passing
- Low level Input/Output
- Low level network support

Microkernel bottlenecks: Highly effective communications between processes is inevitable and the problem of microkernels performance revolves around the extra work to copy data between servers and application programs and the necessary inter-process communication between processes results in extra context switch operations[8].

QNX microkenel performs all inter-process communication by direct copying to reducing complexity and code size which may cause some extra copying costs, in contrary L4 microkernel improves performance by using registers mechanism if the amount of data being passed is small. Anyway, to avoid the mentioned problems different techniques were used by different microkernels-based operating systems. One of the most popular techniques known as co-location, which based on allowing the operating system to optionally run specific programs inside the kernel in particular servers. Although this technique leads to some complexity in the kernel's scheduler however, it significantly reduces the number of context switches because inter-process communication overhead is reduced to normal system call[9].

Microkernel performance in general, is often poor due to switching between kernel and user mode, switching between address spaces and context switching between threads[10] in addition to complicate

implementation that is why most operating systems are using monolithic kernel.

Modern microprocessors architecture: As a result of the growing demand for more high-speed processors, CPU manufacturers began the competition by increasing parallelism at the instruction level to get more performance out of additional transistors on a chip.

This technique eventually led to complex and hard to manage processors, in addition these processors consume large amount of power emitted in the form of high temperature and the problem is become worse the greater the speed of the processor[11]. To resolve this problem, it was necessary to reduce the speed of processors and combine multiple cores on the same chip[12,13].

Multi-core processors came to solve the deficiencies of single core processors, by decreasing power consumption while increasing bandwidth. In a multi-core configuration, an integrated circuit contains two or more complete computer processors, Fig. 4 represents a generic diagram of multi-core processor. Usually, these identical processors are manufactured so they reside side-by-side on the same die. Each of the physical processor cores has its own resources architectural state, registers and execution units. Processor technology trend follows Moore's Law, which states that the number of transistors per a certain area on the chip will double approximately every 18 months[13-15], which is mean that the number of processor cores in one integrated circuit chip will continue to increase and this is also confirmed by processor manufacturers[15]. Multi-core technology requires the development of operating system that capable of dealing with such processors.

Operating systems and software challenge with modern microprocessors architecture: Despite the significant progress in building high-speed processors, new high-speed hardware is not reflected at the same rate on the operating system performance[16] and always bottlenecks were found which prevented using the computer resources to their fullest capacity.

Multi-core processors are built to support parallelism, so, to make use of such processors, operating system must support multithreading and the software must have Simultaneous Multithreading Technology (SMT)[17] written into the application software, otherwise the software will only recognize and run through a single core which leads to significantly decrease the efficiency[18,19].

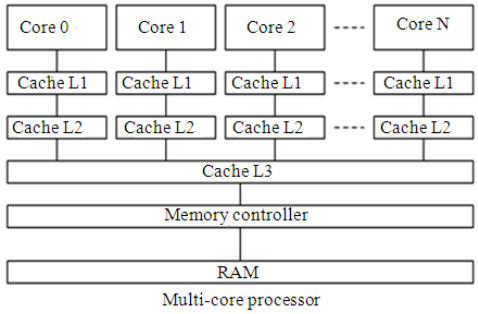

Fig. 4: Generic diagram of multi-core processor

Coding for simultaneous multithreading technology it's not trivial at all[20] because of some issues such as interleaving shared data can slow performance and create errors, in addition, it is not easy to write correct multithreaded programs and if we assumed that we were able to write such program we still facing another serious problem is how to parallelize the threads in the program[21]. This problem can be described as follows: If we have a program with two threads one handle heavy computations while the other perform simple computation, such case would not lead to significant speed in execution because the huge part assigned to single core while the other cores will almost sit idle, which will result in application bottleneck[22]. So, getting significant increase in performance needs optimal conditions[23-25], which may be difficult to achieve in most cases.

There is another problem lies in that most programs do not support multithreading feature, this means re-designing and re-write these programs which will require time, effort and knowledge which is probably still is not available to many programmers. All of this does not mean reaching a final solution, because the operating systems that support dual-cores do not support the Quad-core and which support Quad-core do not support the Octa-core.

This will lead to rebuild the operating system in case of emergence of new processors that contain more number of cores and this fully applied to the software applications[26,27].

MATERIALS AND METHODS

Architecture of proposed model: The proposed model is oriented to multi-core processors and consists of kernels equal to the number of processor cores. The model assumes that kernels divided into two categories:

- One master microkernel and
- Many slave microkernels

The master kernel invoked first and it is responsible for creating all aspects of the system, after that, it creates slave-microkernels and assigns each one of them to one and distinct of the processor cores. That is if we have a processor with N cores the master microkernel assigns itself to Core 0 and create N-1 microkernels corresponding to processor's cores. Figure 5 shows the relationship between microkernels and processor cores.

Master microkernel responsible for the management of the system, that is, it is the only one capable of dealing with all resources of system and directly communicates with servers and in addition it guarantees the communication between salve microkernels with each other and with servers (Fig. 6), while the job of slave microkernels is limited to the execution of user's programs and it can manage and directly access its processor core and its caches(L1 and L2), the other system's resources it can access only through master microkernel.

The proposed model assumes that each processor's core support hyper threading technology[28,29] and has large enough L1 and L2 caches. The first characteristic will enable user programs that support multithreading to make use of hyper threading technology resulting in good responsiveness and performance, but if user program doesn't support multithreading it will run as a single threaded process on single core, while the other cores simultaneously executing other user programs,

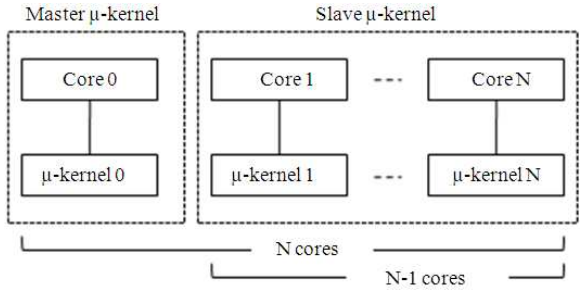

Fig. 5: Relationship between master-slave microkernels

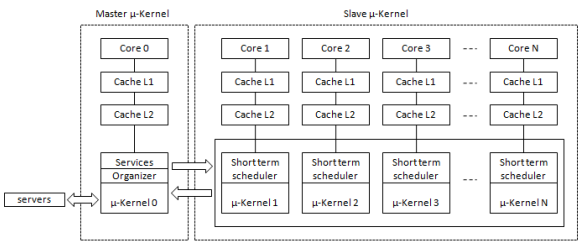

Fig. 6: Model structure and communication between master-slave microkernels and servers

here we can notice the deference between this model and other models when running single threaded process on multi-core processor, the process will be assigned to one core and the other cores will set idle.

The second characteristic will give us the ability to load master, slave microkernels and nearly all user programs completely into their caches to reduce the need for the main memory RAM, which will significantly increase and enhance the performance[30] and one can imagine the difference between cache speed and RAM speed.

Getting cache with large size is an achievable target; because caches chips tend to get larger with each new generation of processors as transistors become smaller which means there will be more area on the die for additional cache[14,15,31].

Description of the proposed model mechanism: We can describe model's mechanism through the following different cases; assume we have a processor with four cores:

Case 1: Four processes arrived, respectively, to the master microkernel P_1, P_2, P_3 and P_4. After the completion of P_3, arrived P_5.

In this case, master microkernel organizer find idle cores and maintains information about each process such as its ID and on which kernel-core running for future use when necessary, then respectively assigns processes to the cores. Accordingly, it will assign P_1 to core1, P_2 to core2, P_3 to core4, since there are no more idle cores and all cores nearly with the same load it starts counting from the beginning and assign P_4 to core1. After a specific time P_3 finished execution, immediately arrived P_5, in this case master microkernel will assign it to core4; because it is the only idle core at the current time.

Case 2: P_1 needs to communicate with P_4: In this case, as both P_1 and P_4 are running in the same microkernel, this means that slave microkernel1 will establish the communication link between the two processes without interference from master microkernel

Case 3: P_1 needs to communicate with P_2: P_1 and P_4 are running in different microkernel, in this case, slave microkernel1 sends a request to master microkernel that it needs to communicate with process P_2 as follows:

Link (P_1@microkernel1, P_4@masterkernel)

As each process has a unique ID the master microkernel directly locate where P_4 is now running

and send request to its microkernel, after that master microkernel establish the communicating link and works as intermediate between the two slave microkernels.

Case 4: P_1 needs file systems service: In this case:

- P_1 sends the request to the master microkernel send (P_1@microkernel1, File system service)
- Master microkernel sends the request to file system service
- File system service performs the request and return the result to the master microkernel
- Master microkernel sends back the result to send (P_1@microkernel1, result)
- In this model, the master microkernel is responsible for all of the following:
- Assigning processes to one of idle or low-load kernels
- memory management
- Inter-process communication between slave microkernels
- Input/output management
- Network support

RESULTS

To evaluate the proposed model in term of performance, let's assume a group of scheduled processes as in Table 1 and a multi-core processor with 4 cores.

For simplicity we will use simple round robin algorithm with quantum time (Q) 20 and it should be mentioned here that the selection of the algorithm does not play an important role, because the evaluation focuses on the throughput of the system in term of its organization and inter-process communication not on the algorithm itself.

For the purpose of performing some calculation to compare multi microkernel model with classical single-kernel model, assume the following variables:

Lost time for each core LST_i: this time considered when $core_i$ idle while other cores busy with a specific process.

Total lost time is TLT: which is representing the ration of idle states to the busy states during all rounds.

Table 1: Group of scheduled processes

Process	Burst time	Type
P_1	40	Multithreaded
P_2	60	Not multithreaded
P_3	38	Not multithreaded

Mathematically we can represent LST_i, in the following form:

$$LST_i = \sum_{1}^{n}\sum_{1}^{i} Q_i \quad (1)$$

Where:

$$Q_i = \begin{cases} 0, & \text{if the core busy} \\ Q, & \text{if the core idle} \end{cases} \quad (2)$$

n = The number of rounds
i = The number of cores

And therefore the value of the total lost time TLT will be:

$$TLT = \frac{\text{number of idle states}}{\text{number of all states}} \quad (3)$$

By applying the algorithm with appropriate parameters for a single-kernel operating systems we will get the result as shown in Fig. 7.

Using formula 1 and 2, to get the value for TL_i and TLT:

$$LST_1 = 0+0+0+0+0+0+0+0 = 0$$
$$LST_2 = 0+20+20+0+20+18+20 = 100$$
$$LST_3 = 0+20+20+0+20+18+20 = 100$$

$$TLT = \frac{10}{21} = 0.48$$

The obtained results show that core1 is busy all the time while core2 and core3 busy only one third of the time, which means bad distribution of loads between cores, leading to loss of time due to poor exploitation of microprocessor's cores. Figure 8 represents utilization ratio for each core.

Now we will use the same data for the proposed model (multi microkernel model), this will give us the results shown in Fig. 9.

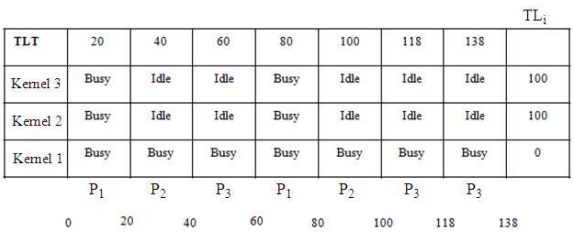

Fig. 7: Lost time for single-kernel operating systems

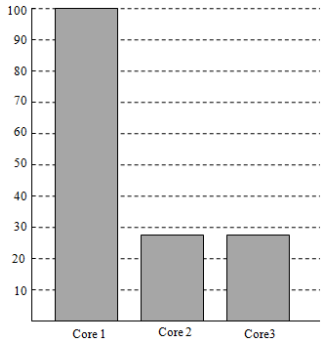

Fig. 8: Utilization ratio for each core in single-kernel operating systems

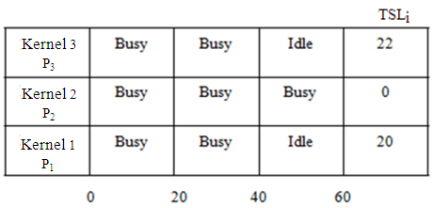

Fig. 9: Lost time for multi microkernel operating systems

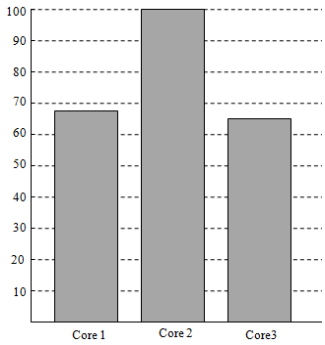

Fig. 10: Utilization ratio for each core in multikernel operating systems

$$LST_1 = 0 + 0 + 20 = 20$$
$$LST_2 = 0 + 0 + 0 = 0$$
$$LST_3 = 0 + 0 + 22 = 22$$

$$TLT = \frac{2}{9} = 0.22$$

The obtained results show that core1 is busy all the time while core2 and core3 busy almost two-thirds of the time, which means that the new proposed model is better in term of distribution of loads between cores, leading to minimize loss of time due to good exploitation of microprocessor's cores. Figure 10 represents utilization ratio for each core.

In addition TLT for single-microkernel is almost twice the time of multi microkernel:

$TLT_{single\text{-}microkernel} = 0.48$
$TLT_{multi\ microkernel} = 0.22$

This means that we can save twice or may be more than twice the time using the new proposed model, leading to high performance due to good utilization of microprocessor cores.

DISCUSSION

The proposed model based on multikernel approach shows through the obtained results that the performance of multi-microkernel-based operating system is much better than single-microkernel-based operating system.

By making the master microkernel only responsible for the fundamental services in the system running on a separate core with large caches this will lead to eliminate all classical problems related to microkernel-based operating systems such as bottleneck, switching between user and kernel mode. On the other hand, assigning the task of executing user programs to the slave microkernels, by dividing the total number of process as subsets between the slave microkernels this will enhance the performance, increase the throughput and decrease turnaround time, waiting time and response time of the system.

Creating a number of slave microkernels depending on the number of processor cores make the operating system independent of the microprocessor architecture which gives it the ability to behave dynamically, which is mean that the operating system can deal with processors with any number of cores without the need for rebuilding it.

CONCLUSION

In this study, new model for operating system presented to solve the bottlenecks problem and operating systems and software challenge with modern microprocessors architecture of classical single-microkernel-based operating system, the proposed model shows high performance compared with the classical model, because of its dynamic nature and independency of microprocessor architecture, in addition to its ability to adapt with both multithreaded and single threaded process.

REFERENCES

1. Silberschatz, A., P. Baer Galvin and G. Gagne, 2004. Operating System Concepts. 7th Edn. John Wiley and Sons, Hoboken, New Jersey, ISBN: 10: 0471694665, pp: 944.

2. Heiser, G., K. Elphinstone, G. Klein, I. Kuz and M.S. Petters, 2007. Towards trustworthy computing systems: Taking microkernels to the next level. Operat. Syst. Rev., 41: 3-11. http://portal.acm.org/citation.cfm?id=1278901.1278904
3. Jochen, L., 1995. On µ-kernel construction. Proceeding of 15th ACM Symposium on Operating System Principles, (SOSP'95), pp: 237-250. http://www.citeulike.org/user/rahul/article/364430
4. Brett, D. Fleisch, Mark Allan A. Co, 1999. Workplace microkernel and OS: A case study. Software: Pract. Exp., 28: 569-591. http://www3.interscience.wiley.com/journal/1798/abstract?CRETRY=1&SRETRY=0
5. Andrew S. Tanenbaum, 1995. Distributed Operating Systems. Prentice Hall, ISBN: 0132199084, pp: 648.
6. Mark E. Russinovich and David A. Solomon, 2003. Microsoft Windows Internals. 4th Edn., Microsoft Press, ISBN: 0-7356-1917-4, pp: 976.
7. Andrew, S., 2001. Modern Operating Systems, 2/E, Tanenbaum. Prentice Hall, ISBN: 13: 9780130313584, pp: 976.
8. Hidaka, S., K. Kodama, Y. Ji and K. Maruyama, 2002. A file server optimization using scatter/gather IPC on L4 based multi-server operating system. Proceedings of the 6th World Multi Conference on Systemics, Cybernetics and Informatics, (SCI'02), Tokyo, pp: 184-189. http://research.nii.ac.jp/H2O/SCI-2002.pdf
9. Rajkumar, R., 1999. Operating Systems and Services. Springer, ISBN: 0792385489, pp: 204.
10. Härtig, H., M. Hohmuth, J. Liedtke, J. Wolter and S. Schönberg, 1997. The performance of µ-kernel-based systems. Operat. Syst. Rev., 31: 66-77. http://direct.bl.uk/bld/PlaceOrder.do?UIN=038779461&ETOC=RN&from=searchengine
11. Peng, L. *et al.*, 2007. Memory performance and scalability of intel's and AMD's dual-core processors: A case study. Proceeding of the IEEE International Conference on Performance, Computing and Communications, Apr. 11-13, IEEE Xplore Press, New Orleans, LA., pp: 55-64. DOI: 10.1109/PCCC.2007.358879
12. Knight, W., 2005. Two heads are better than one. IEEE. Rev., 51: 32-35. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1540008
13. Geer, D., 2005. Chip makers turn to multi-core processors. Computer, 38: 1-13. DOI: 10.1109/MC.2005.160
14. Gordon Moore, E., 1965. Cramming more components onto integrated circuits. Electronics, 38: 114-117. http://www.citeulike.org/user/sjanusz/article/814762
15. Access My Library, 2005. Intel, Innovation more important than ever in platform era. http://www.accessmylibrary.com/coms2/summary_0286-18980112_ITM
16. Muneer, H. and K. Rashid, 2006. SPE architecture for concurrent execution OS kernel and user code. Inform. Technol. J., 5: 192-197. http://scialert.net/asci/ascidetail.php?doi=itj.2006.192.197&kw=
17. Eggers, S.J., J.S. Emer, H.M. Levy, J.L. Lo, R.L. Stamm and D.M. Tullsen, 1997. Simultaneous multithreading: A platform for next-generation processors. IEEE Micro, 17: 12-19. DOI: 10.1109/40.621209
18. Ron Kalla, Balaram Sinharoy and J.M. Tendler, 2004. IBM power5 chip: A dual-core multithreaded processor. IEEE Micro, 24: 40-47. DOI: 10.1109/MM.2004.1289290
19. Van Roy Peter, 2008. The challenges and opportunities of multiple processors: Why multi-core processors are easy and internet is hard. Proceeding of the International Computer Music Conference, (ICMC'08), Belgium, pp: 1-2. http://www.info.ucl.ac.be/~pvr/vanroy-mc-panel.pdf
20. Artho, C. and A. Biere, 2001. Applying static analysis to large-scale, multi-threaded java programs. Proceeding of the 13th Australian Software Engineering Conference, Aug. 27-28, IEEE Computer Society Washington DC., USA., pp: 68-68. http://portal.acm.org/citation.cfm?id=872575
21. STAR Watch, 2005. Double the performance: Dual-core CPU's make their debut. http://www.wnylc.net/pdf/star-watch/MayJunel05.pdf
22. Visser, W., K. Havelund, G. Brat and S. Park, 2000. Model, checking programs. Proceeding of the International Conference on Automated Software Engineering, pp: 1-10. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.3362&rep=rep1&type=pdf
23. Akhter, S. and J. Roberts, 2006. Multi-Core Programming: Increasing Performance through Software Multithreading. 1st Edn., Intel Press, ISBN: 13: 978-0976483243, pp: 360.
24. Stewart Taylor, 2007. Optimizing Applications for Multi-Core Processors, Using the Intel® Integrated Performance Primitives. 2nd Edn., Intel Press, ISBN: 13: 978-1934053010, pp: 600.

25. Hughes, C. and T. Hughes, 2008. Professional Multicore Programming: Design and Implementation for C++ Developers. Wrox, ISBN: 13: 978-0470289624, pp: 648.
26. Tullsen, D.M., S.J. Eggers and H.M. Levy, 1995. Simultaneous multithreading: Maximizing on-chip parallelism. Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 22-24, IEEE Xplore Press, USA., pp: 392-403. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=524578
27. Tullsen, D.M., S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo and R.L. Stamm, 1996. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. Comput. Architect., 24: 191-202. http://portal.acm.org/citation.cfm?id=232974.232993
28. Koufaty, D. and D.T. Marr, 2003. Hyperthreading technology in the netburst microarchitecture, Micro IEEE., 23: 56-65. DOI: 10.1109/MM.2003.1196115
29. Lin Chao, 2002. Hyper-threading technology. http://www.buzzle.com/editorials/7-31-2004-57330.asp
30. Geer, D., 2007. For programmers, multicore chips mean multiple challenges. Computer, 40: 17-19. http://portal.acm.org/citation.cfm?id=1301953
31. Alameldeen, A.R. and D.A. Wood, 2007. Interactions between compression and prefetching in chip multiprocessors. Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Feb. 10-14, IEEE Xplore Press, Scottsdale, AZ., pp: 228-239. DOI: 10.1109/HPCA.2007.346200