International Journal of Research (IJR) Vol-1, Issue-10 November 2014 **ISSN 2348-6848**

Multithreading In Java

Kuldeep Panwar & Devanshu Sharma
Electonics and Computer Science
Kuldeep.15715@Ggnindia.Dronacharya.Info, Devanshu.15715@Ggnindia.Dronacharya.Info

ABSTRACT
In this paper we discuss the use of multithreading, its types and difference from other multitasking processes. This is just a basic research on multithreading. This doesn't conclude the new research paradigm. The basic details of multithreading used in java is defined here.
We have discussed our basic approach that is needed for the betterment of multithreading in our context.
There is a little concusion that came out of this research.

Keywords—
About four key words ;phrases in alphabetical order; separated by commas.

I. INTRODUCTION

Java is a*multithreaded programming language* which means we can develop multithreaded program using Java. A multithreaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition multitasking is when multiple processes share common processing resources such as a CPU.[20] Multithreading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.[2]

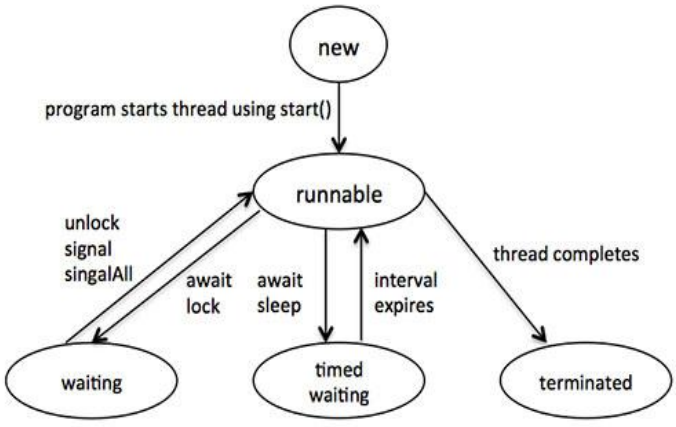
Multithreading enables you to write in a way where multiple activities can proceed concurrently in the same program.

formatted further at IJEMR. Define all symbols used in the abstract.

II. LIFE CYCLE OF A THREAD

Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.[4]



III. LIFE CYCLE EXPLAINED

Above-mentioned stages are explained here:

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the

program starts the thread. It is also referred to as a born thread.

- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.[3]
- **Waiting:** Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.[6]

Thread Priorities:[8]

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).[5]

Threads with higher priority arė more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and very much platform dependentant.[5]

IV. USE OF MULTITHREADING

The use of multi-threading programming is the key to take advantage of the increasing number of processing cores in central processing units in each new generation of processors; it will [18] be necessary if we want simulators to continue developing in features and performance, while supporting larger number of simultaneously simulated robots. Multithreaded software applications – programs that run multiple tasks (threads) at the same time to increase performance for heavy workload scenarios, are already positioned to take advantage of multi-core processors [10]. Our goal is to try to program the simulator in a way that takes benefit of current processors,[19] while at the same time avoid implementing them in the same way as discussed in past paragraph in order to avoid their drawbacks.[17] So the main idea of our approach is to use multi-threading programming as a component of the simulator architecture design, and not just in control method implementations where it is left to the programmer to decide whether to implement multithreading or not. [20]

A thread of execution by definition is a "fork" of a computer program into two or more concurrently running tasks [16]. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process.[18] Multiple threads can exist within the same process and share resources such as memory, while different [12] In the non threaded approach when dealing with simulating multi robot systems, the robots are put in an array, and then sequentially the CPU resources are passed to each of them, in their turn, to calculate their actions, and when each of them finishes, then the available resources are passed to the next robot, and so on, until all robots in the array do their jobs.[16] Then this cycle starts again, from the first robot to decide its next step, and so on.[14] The performance of this approach is acceptable

when all robots are of the same type, or if they need similar time to complete their decision making
process. in which we are simulating four [14] robots at the same time (each of them needing a different time to accomplish its control strategy calculations), we encounter the following situation: the Robot3 will need 50 milliseconds to finish
the calculations in order to take a decision, and this will affect the simulated Robot4, since it cannot
start its own calculations until Robot3 finishes. This situation causes a lag that the Robot4 is not responsible of.[9] This is one shortcoming of not using threads in the programming. Threading will
eliminate this problem even when we only have single core processor, because if we use
multithreading and assign a separate thread to each simulated robot, then the slow robot (Robot3)
will not slow down all the simulation and other robots till it finishes its calculations. Instead, each [3]
robot will only affect itself, and the robot with easy calculations will be simulated normally as it should be, while the complex robot will stay in its place until finishing its calculations with[12]

V. CONCLUSION

Implementing multithreaded Java code is reasonably straightforward. Even converting existing single-threaded Java GUI code to a multithreaded format is not difficult. The results are actually quite impressive: Your GUI is freed up from long waits and ready to process other user actions. However, your application might or might not permit the interleaving of operations; imagine an application that fetches customer names from a database before printing out invoices. It wouldn't make much sense to put the printing code in a separate thread and then try to print invoices before customer names had been read from the database!

For application operations where it's okay to interleave user actions, making a GUI multithreaded can be very useful. It can even help to improve end user productivity by allowing a number of application operations to proceed in parallel.

However, the power that multithreaded programming provides should be used with care. If you use shared data across threads, then it's quite easy to get into some very difficult debugging scenarios.The same is true of locking shared resources between threads. The rule of thumb is to follow the guidelines and start out being conservative in your multithreaded designs. As you get more experienced with the use of multithreaded programming, you can then start to use the more powerful features.

REFERENCES

1. Nilsson, Theodor, "KiKS is a Khepera Simulator", (2001-03-14), Umea University.
2. Mondada, Francesco, Franzi, Edoardo and Ienne, Paolo, (1993), "Mobile robot miniaturisation:
A tool for investigation in control algorithms." Proceedings of the 3rd International Symposium on Experimental Robotics, Kyoto, Japan.
3. Michel, Olivier, (2004), "WebotsTM: Professional Mobile Robot Simulation", International
Journal of Advanced Robotic Systems, Volume 1, Number 1, pp. 39-42.
4. Liu, Jiming, Wu, Jianbing, (2001), "Multi-Agent Robotic Systems", CRC Press International
Series on Computational Intelligence.
5. Katsumi Kimoto, Shinichi Yuta, "Autonomous mobile robot simulator - a programming tool for sensor-based behavior Autonomous Robots", Autonomous Robots, Springer Netherlands,
Volume 1, Number 2 / June, 1995, pages 131-148.

6. Fishwick, Paul, (1995), "Simulation Model Design & Execution: Building Digital Worlds", Prentice Hall.
7. Intel®, (January 2003), "Hyper-Threading Technology Technical User's Guide".
8. The Rossum Project, Open-Source Robotics Software, Retrieved July 20th 2009 from http://rossum.sourceforge.net/
9. "Microprocessor": Retrieved July 20th, 2009, from: http://en.wikipedia.org/wiki/Microprocessor.
10. Advanced Micro Devices, Inc. "Multi-core White Paper": Retrieved July 20th, 2009, from: http://www.sun.com/emrkt/innercircle/newsletter/0505multicore_wp.pdf.
11. Microsoft MSDN http://msdn.microsoft.com
12. Halfhill, Tom R., (12/31/07), "The Insider's Guide to Microprocessor Hardware, The Future of Multicore Processors". Reed Electronics Group.www.MPRonline.com
13. Akhter, Shameem, Roberts, Jason, (2006), "Multi-Core Programming Increasing Performance through Software Multi-threading", Intel Press; 1ST edition.
14. Gerkey, Brian P., Vaughan, Richard T., Howard, Andrew, (2003), "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". Proceedings of the International Conference on Advanced Robotics (ICAR) pages 317-323, Coimbra, Portugal.
15. Distributed computing: Retrieved July 20th, 2009, from: http://en.wikipedia.org/wiki/Distributed_computing
16. Lewis, Bill: (1995), "Threads Primer: A Guide to Multithreaded Programming", Prentice Hall.
17. Thread. sleep: Retrieved July 20th, 2009, from: http://www.javamex.com/tutorials/threads/sleep.shtml
18. Guz, Zvika, Bolotin, Evgeny, Keidar, Idit, (2009), "Many Core vs. Many-Thread Machines: Stay Away From the Valley". IEEE Computer Architecture Letters, vol. 8, no. 1, pp. 25-28.
19. Gravinghoff, Andreas, Keller, Jorg, "How to Emulate Fine-Grained Multithreading", Fern University Hagen, Germany, retrieved on 20th July 2009 from: http://www.fernunihagen.de/imperia/md/content/fakultaetfuermathematikundinformatik/forschung/berichte/bericht_227.pdf
20. Fedy Abi-Chahla (09/16/2008), "Multi-Threaded Rendering" Retrieved on 20th July 2009 from: http://www.tomshardware.com/reviews/opengl-directx,2019-6.html

_____