Assignment 1

"Memory Management"

(20 Points)

Referring to Operating Systems architecture, you are to write an assignment on the "Memory Management" which is part of the computer's physical memory or Random-Access Memory (RAM) organization.

Among other issues, you should provide a complete explanation regarding the following issues;

- **Process Address Space**
- **Symbolic addresses**
- **Relative addresses**
- **Physical addresses**
- **Static vs Dynamic Loading**
- **Static vs Dynamic Linking**
- **Swapping**
- **Memory Allocation**
- **Single-partition allocation**
- **Multiple-partition allocation**
- **Fragmentation**
- **External fragmentation**
- **Internal fragmentation**
- **Paging**

· Address Translation

· Segmentation

The assignment is between 25 to 30 pages and should be written in accordance with the specifications given by the instructor,

APA style **(see www.apa.org).**

Notes on Assignment 1 Writing

All assignments will be carefully looked at and appropriate marks are given.

These marks are given in the following fashion;

1. **How close is the written assignment to the actual theme of the given assignment title?**
2. **How well the idea(s) are presented in the assignment.**
3. **The amount of plagiarism does not exceed 15%.**

The marks are spread out in accordance with the above rubrics.

You should *include figures* in your discussion. And when you do, these should be *labeled and referred* to in your discussion correctly, not leaving things for the reader to figure out what's what.

Also, always *check* your text for writing mistakes and never refer to a URL inside your text, URL placement is in the *references* at the end of your assignment.

Assignment due date must be abided with.

Two points will be deducted for each day being late.

Most sound operating systems books will/should deal with the following issues too.

These are;

- Subdividing memory to accommodate multiple processes.
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.
- Preparing a Program for Execution.
- Program Transformations.
- Logical-to-Physical Address Binding.
- Memory Partitioning Schemes.
- Fixed Partitions.
- Variable Partitions.
- Allocation Strategies for Variable Partitions.
- Dealing with Insufficient Memory.

Possible References. (33 in Total)

The following original references constitutes some of the main source of knowledge gained by current technologies as applied to memory management.

[1] Benjamin G. Zorn. The measured cost of conservative garbage collection. Software Practice and Experience, 23(7):733–756, 1993.
[2] William S. Beebee and Martin C. Rinard. An implementation of scoped memory for Real-Time Java. In EMSOFT, pages 289–305, 2001.
[3] Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe, and Paul R. Wilson. Hoard: A scalable memory allocator for multithreaded applications. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), pages 117–128, Cambridge, MA, November 2000.
[4] Emery D. Berger, Benjamin G. Zorn, and Kathryn S. McKinley. Composing high-performance memory allocators. In Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 114–124, Snowbird, Utah, June 2001.
[5] Greg Bollella, James Gosling, Benjamin Brosgol, Peter Dibble, Steve Furr, and Mark Turnbull. The Real-Time Specification for Java. Addison-Wesley, 2000.
[6] Gilad Bracha and William Cook. Mixin-based inheritance. In Norman Meyrowitz, editor, Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA) / Proceedings of the European Conference on Object-Oriented Programming (ECOOP), pages 303–311, Ottawa, Canada, 1990. ACM Press.
[7] Dov Bulka and David Mayhew. Efficient C++. Addison-Wesley, 2001.
[8] Trishul Chilimbi. Efficient representations and abstractions for quantifying and exploiting data reference locality. In Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 191–202, Snowbird, Utah, June 2001.
[9] Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. Cache-conscious structure layout. In Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 1–12, Atlanta, GA, May 1999.

[10] Margaret A. Ellis and Bjarne Stroustrop. The Annotated C++ Reference Manual. Addison-Wesley, 1990.
[11] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. Lecture Notes in Computer Science, 986, 1995.
[12] Christopher W. Fraser and David R. Hanson. A Retargetable C Compiler: Design and Implementation. Addison-Wesley, 1995.
[13] Mark Weiser, Alan Demers, and Carl Hauser. The Portable Common Runtime approach to interoperability. In Twelfth ACM Symposium on Operating Systems Principles, pages 114–122, December 1989.
[14] David Gay and Alex Aiken. Memory management with explicit regions. In Proceedings of the 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 313 – 323, Montreal, Canada, June 1998.
[15] David Gay and Alex Aiken. Language support for regions. In Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 70 – 80, Snowbird, Utah, June 2001.
[16] Wolfram Gloger. Dynamic memory allocator implementations in Linux system libraries. http://www.dent.med.uni-muenchen.de/˜ wmglo/malloc-slides.html.
[17] Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, and James Cheney. Region-based memory management in cyclone. In Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 282–293, Berlin, Germany, June 2002. 11
[18] Sam Guyer, Daniel A. Jimenez, and Calvin Lin. The ´ C-Breeze compiler infrastructure. Technical Report UTCS-TR01-43, The University of Texas at Austin, November 2001.
[19] David R. Hanson. Fast allocation and deallocation of memory based on object lifetimes. In Software Practice & Experience, number 20(1), pages 5–12. Wiley, January 1990.
[20] David R. Hanson. C Interfaces and Implementation. Addison-Wesley, 1997.
[21] Reed Hastings and Bob Joyce. Purify: Fast detection of memory leaks and access errors. In Proceedings of the Winter USENIX 1992 Conference, pages 125–136, December 1992.
[22] Mark S. Johnstone and Paul R. Wilson. The memory fragmentation problem: Solved? In International Symposium on Memory Management, pages 26–36, Vancouver, B.C., Canada, 1998.
[23] Kiem-Phong Vo. Vmalloc: A general and efficient memory allocator. In Software Practice & Experience, number 26, pages 1–18. Wiley, 1996.
[24] Scott Meyers. Effective C++. Addison-Wesley, 1996.
[25] Scott Meyers. More Effective C++. Addison-Wesley, 1997.
[26] Bartosz Milewski. C++ In Action: Industrial-Strength Programming Techniques. Addison-Wesley, 2001.
[27] Dan N. Truong, Franc¸ois Bodin, and Andre Seznec. ´ Improving cache behavior of dynamically allocated data structures. In International Conference on Parallel Architectures and Compilation Techniques, pages 322–329, October 1998.
[28] Jeffrey Richter. Advanced Windows: the developer's guide to the Win32 API for Windows NT 3.5 and Windows 95. Microsoft Press.
[29] Gustavo Rodriguez-Rivera, Mike Spertus, and Charles Fiterman. Conservative garbage collection for general memory allocators. In International Symposium on Memory Management, Minneapolis, Minnesota, 2000.
[30] D. T. Ross. The AED free storage package. Communications of the ACM, 10(8):481–492, 1967.

[31] Colin Runciman and Niklas Rojemo. Lag, drag and postmortem heap profiling. In Implementation of Functional Languages Workshop, Bastad, Sweden, September 1995.
[32] SGI. The Standard Template Library for C++: Allocators. http://www.sgi.com/tech/stl/Allocators.html.
[33] Ran Shaham, Elliot K. Kolodner, and Mooly Sagiv. Heap profiling for space-efficient Java. In Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 104–113, Snowbird, Utah, June 2001.