Project Title	Project Specifications	Additional Links
A "Better" Malloc	* implement a memory allocator for the heap of a user-level process * Malloc - functions similar to traditional malloc, but free can free memory from any point in the allocated memory implementations of Mem_Alloc (int size) and Mem_Free (void *ptr) are identical, except the pointer passed to Mem_Free does not have to have been previously returned by Mem_Alloc; instead, ptr can point to any valid range of memory returned by Mem_Alloc. Will need to have a more sophisticated data structure than the traditional malloc to track the regions of memory allocated by Mem_Alloc. Specifically, this data structure will allow you to efficiently map any address to the corresponding memory object or to determine that there is no corresponding object.	http://pages.cs.wisc.edu/~dusseau/Classes/CS537-F07/Projects/P3/index.html
A "Slower" File System	* build a user-level library, libFS, which implements a good portion of a file system * Your file system will be built inside of a library that applications can link with to access files and directories. Your library will in turn link with a layer that implements a "disk" * Create a complete File Access API supporting functions for file creation, file open, file read, write, seek, close etc. * Same functions for directories	http://pages.cs.wisc.edu/~dusseau/Classes/CS537-F07/Projects/P4/index.html
Building peer-to-peer systems with chord, a distributed lookup service	The core problem facing peer-to-peer Systems is locating documents in a decentralized network and propose Chord, a distributed lookup primitive. Chord provides an efficient method of locating documents while placing few constraints on the applications that use it.	http://ieeexplore.ieee.org/document/990065 http://nms.lcs.mit.edu/papers/chord.pdf
Google File System	Implementation of the Google File System, a scalable distributed file system for large distributed data-intensive applications. Possible file system interface extensions.	http://www.cs.cornell.edu/courses/cs614/2004sp/papers/gfs.pdf
Implementation of Inode Based File System (Simple file system)	The goals of this project are to implement a simple file system on top of a virtual disk and to understand implementation details of file systems	http://www.cs.ucsb.edu/~chris/teaching/cs170/projects/proj5.html
Implement mounting Google Drive as a File-System using libfuse (File System in User Space)	Design a user space C/C++ application that links against libfuse and maps the appropriate kernel calls to HTTP requests to Google Drive API and vice-versa.	https://github.com/libfuse/libfuse https://developers.google.com/drive/v3/web/about-sdk
The Slab Allocator: An Object-Caching Kernel Memory Allocator	This allocator is based on a set of object-caching primitives that reduce the cost of allocating complex objects by retaining their state between uses. These same primitives prove equally effective for managing stateless memory (e.g. data pages and temporary buffers) because they are space-efficient and fast. Slab allocator works along with Buddy allocation system. Can you implement one on any old Unix kernel with only support for buddy allocator?	http://srl.cs.jhu.edu/courses/600. 418/SlabAllocator.pdf
A Multithreaded Server	* First task is to design, implement, and test an abstraction called a thread pool. Your thread pool implementation should consist of two files: threadpool.h and threadpool.c. Implementation shouldn't contain BUSYWAITING or DEADLOCKS or RACE CONDITIONS * Use your thread pool to turn the single-threaded server into a multithreaded server. * Finally test the performance of your server	https://homes.cs.washington.edu/~zahorjan/homepage/Tools/LinuxProjects/MTServer/proj2.html

Modifying the Blitz Operating System	BLITZ is a collection of software, designed to streamline the process of learning about, and experimenting with, operating system kernel code. By using BLITZ students are able to study, in detail, the low-level operating system code that interacts with the hardware, as well as design, code and test their own modifications to the operating system. The project comprises of multiple components: - Understanding the OS code, performing threading and inter-process communication, kernel resource management and user processes.	http://web.cecs.pdx.edu/~walpole/class/cs333/spring2012/home. html
Build a Distributed Key-Value Store - Piazza	Implement a distributed key-value store that runs across multiple nodes. Data storage is durable, i.e., the system does not lose data if a single node fails. You will use replication for fault tolerance. The key-value store is to be built optimizing for read throughput. Accessing data concurrently from multiple replicas of the data will be used to improve performance. Operations on the store should be atomic. i.e., either the operation should succeed completely or fail altogether without any side effects. You will use the Two-Phase Commit protocol to ensure atomic operations.	https://inst.eecs.berkeley.edu/~cs162/fa12/
Nachos Operating System - Threading and Multiprogramming implementation	Stock Nachos has an incomplete thread system. Your job is to complete it, and then use it to solve several synchronization problems. The second phase of Nachos is to support multiprogramming and system calls. In a real operating system, the kernel not only uses its procedures internally, but allows user-level programs to access some of its routines them via system calls.	https://inst.eecs.berkeley.edu/~cs162/fa12/
Pastry	A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.	http://research.microsoft.com/en-us/um/people/antr/PAST/pastry.pdf
Map Reduce Implementation	MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. As a part of project, students need to implement map reduce parallel implementation for various problems like word count, inverted index creation on wiki dump, Facebook common friend finder etc. Handling node failures, network failures, data replication would be few of the challenges. Simultaneous execution of multiple map reduce jobs would require handling of synchronization, socket communication, race conditions.	https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
Version Control System (git)	Implement a git like client system with limited capabilities. The program should be able *To create/initialize a git repository, creating a .git folder with the necessary details. *To be able to add files to index and commit, maintaining the commit-ids so that retrieving them back could be done. *To be able to see the status, diff, checkout previous commits, as shown by the git utility. *Bonus: To be able to communicate the GitHub server and upload files, getting update of only the changed files	http://shafiulazam.com/gitbook/index.html
Snapshot Of File System	Add snapshots to a file system, so that a user can look at the file system as it appeared at various points in the past. You'll probably want to use some kind of copy-on-write for disk storage to keep space consumption down.	https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.3/com.ibm.spectrum.scale.v4r23.doc/bl1adv_logcopy.htm
Distributed Ledger	The aim is to design a bitcoin system. Unlike Bitcoin, we will not use crypto- graphic primitives, or other hashing mechanisms	http://www.cse.iitd.ac. in/~mcs162658/cop701/A1.pdf