

CSE451 Winter 2013 Project #2

Out: January 16, 2013

Due: January 30, 2013 by 11:59 PM (late assignments will lose 20% per day)

Objectives

In the first project you learned how to build and debug the Windows kernel, and how to enhance a system call. This second project builds on the first one by requiring more information be safely managed and returned via the query call. The principal objectives of this second project are:

1. Learn how to use synchronization primitives to ensure your code is properly synchronized for a multi-processor and preemptive kernel environment.
2. Learn how to manage dynamic data in a kernel.
3. Learn about buffering data and dealing with unexpected overflow conditions

Getting Started

Some of the new tools that you will need for the project are:

1. **Mutex** - Defined in `base\ntos\inc\ex.h` are a set of routines to manipulate a structured called a `FastMutex`. A `FastMutex` is a basic synchronization mechanism used throughout the kernel. It is essentially a binary semaphore but optimized for single ownership. The `FastMutex` calls to use are `ExInitializeFastMutex`, `ExAcquireFastMutex` and `ExReleaseFastMutex`. You can ignore the other versions (i.e., `Ki...`, `xx...`)
2. **Pool** - Dynamically allocated memory in the kernel is called pool. There are two types of pool “`PagedPool`” and “`NonPagedPool`.” `PagedPool` is memory that is allowed to be paged out to disk, while `NonPagedPool` must remain in memory at all times. As a rule of thumb allocation should be from `PagedPool` unless the code accessing the data must not take a page fault. For example, the paging code itself can never take a page fault, otherwise the data can never be read. Another rule of thumb in Windows is to never take a page fault holding a spinlock or with interrupts disabled. The routines to allocate and free memory are called `ExAllocatePool` and `ExFreePool`. They are declared in `base\ntos\ex.h` and their usage is pretty straightforward.
3. **Time** - Time in NT is stored in the kernel as a 64 bit integer. The resolution is 100ns. Kernel mode code calls `KeQuerySystemTime` to get the current time. There are also routines in `base\ntos\rtl\time.c` to help manipulate times.

This is just a partial list to help you get started, there are definitely more things in the kernel that you can use to help complete this project.

Your assignment:

Start by copying your own solution for Project 1 to a new directory called Project 2.

1. Most likely your solution for project 1 did not fully synchronize access to your kernel data structures and therefore the code will not always yield the correct answer on a preemptive kernel or on an MP system. The first part of this assignment is to correct this deficiency by protecting access to all the data structures that you added to project 1. You should use this using the Mutex provided in NT. Granularity (or scope) of the lock is up to you, but choose wisely.
2. We want to enhance your code from project 1 to include the order in which the actual calls and returns take place. To do this you will need to have the system keep a history buffer of these events that will be returned to the user on the query call. Each event needs to have a timestamp and specify an action. Actions are either "Call", "Return", or "Buffer Overflow". Each Call and Return action must also specify its appropriate NT API. In addition each return event must specify its returns status classification. The "Buffer Overflow" will denote holes in the timeline where the system was unable to keep a full history because of memory limitations.

Extend the SystemCSE451Information class to enable the user to retrieve the history of events. The user can supply any sized buffer and the system will need to return as much data as the buffer can hold or until there are no more events to return. If the buffer is empty the system returns STATUS_NO_MORE_ENTRIES.

Use PagedPool to store the event history in 4KB buffers that you will dynamically allocate and free as necessary. Do not consume more than 64KB worth of buffers. This limit does not include any ancillary data that you need to manage the buffers. It is probably convenient to visualize each NT API call as producing events and the query as consuming events. If an event occurs and all 64KB is already in use then the system will insert a buffer overflow event. The overflow condition will exist until a query is used to drain some of the buffer. The system must only keep as many 4KB buffers as necessary to keep the history. That is, it needs to allocate and free buffers as needed.

3. Modify your test program to take as input a buffer size. The buffer size is the number of bytes to use when calling NtQuerySystemInformation. The new output can now look like:

```
Z:\Project2\test>test 5000
```

API	Total Calls	Success	Info	Warn	Error	Bytes
NtCreateFile	132	77	0	33	22	
NtOpenFile	100	60	0	0	40	
NtReadFile	777	555	0	0	222	123,456
NtWriteFile	44	22	11	0	11	234,456
NtQueryInformationFile	123	123	0	0	0	
NtSetInformationFile	34	30	3	0	1	
NtQueryDirectoryFile	78	67	0	0	11	

```
History
```

```
-----
```

```
yyyy/mm/dd HH:MM:SS.sss NtOpenFile Call
```

```
yyyy/mm/dd HH:MM:SS.sss  NtOpenFile Return Success
yyyy/mm/dd HH:MM:SS.sss  NtReadFile Call
                          Buffer Overflow
yyyy/mm/dd HH:MM:SS.sss  NtWriteFile Return Warning
```

Turn-in:

Be prepared to turn in the following on Catalyst:

1. Executable images of your test program and the modified kernel.
2. Source code for your test program and the modules that you have modified or added to the kernel.
3. Submit also a concise one page summary of your design and the modifications you added to the sources

Grading Criteria:

Your project will be graded on the following scale.

10 points Correctness
20 points Readability
10 points Elegance
10 points Write-up

Late assignments will lose 20% per day.

Addendum:

Instead of putting all your new code into an existing file like sysinfo.c, you might find it convenient to add a new file in the ex directory. You do this by adding the filename to ex\BUILD\makefile.

You will probably need to add some initialization code that runs once at boot time. Finding the right place to add this code is always tricky. For this project you want to ideally run your initialization code after it is valid to call ExAllocatePool but before any NT API has been called. We have already located this place. The location in the code where you can safely insert your initialization code is right after line 584 in ntos\init\initos.c