Nicholas Castorina

0822759

Project 2 Summary

First in base/ntos/init/initos.c, I initialized a mutex (instance in sysinfo.c; extern in base/ntos/inc/ex.h) for use when incrementing the status counts each of the CSE451 APIs and updating the history. I only used a single mutex because when any of the APIs are called the history has to be updated. Thus, it makes sense that when incrementing a status code and adding an event to the history to only use a single mutex.

I then moved the incStatus function (from project 1) to sysinfo.c because I felt that the users didn’t need to know about this function definition since it was for internal use only. Thus, I removed the SYSTEM\_CSE451\_INFORMATION argument and made it so it only modified the instance of the struct in sysinfo.c that is keeping track of all of the system data. I also used the mutex to make it thread safe. I also added in a BytesUsed argument to the function, so that it would update the bytes consumed by a particular function upon a success status, so as to make updating the BytesUsed threadsafe.

Next I created the HIST\_LIST struct (ntexapi.h) which contains an array of pointers to chunks of event lists. This allows me to have pointers to each of the 4kB event lists, allowing them to easily allocate and free as necessary. This is a struct only used internally. When passed to the user in SYSTEM\_CSE451\_INFORMATION, they are simply given a list of events back for simplicity sake. Because I manually handle the chunks and am making it transparent to the user, I added an instance of HIST\_LIST to sysinfo.c (extern in ntosdef.h).

I then created a function addToHistory() (sysinfo.c) that adds a function (i.e. NtOpenFile, NtCreateFile, etc.), an action (i.e. Call, Return, etc.), a Status (whole status code), and a time (in 100ms) to the history. If there isn't enough room in the history buffer to add the call, it overwrites the last entry with a BufferOverflow event. If the last entry in the last buffer is a BufferOverflow when the last BufferOverflow occurs, it keeps the old timestamp. This addToHistory() function only modifies the internal chunked history representation.

I then made a threadsafe version of addToHistory() (called addHistCall()) that only adds the Call actions of a function to the history. I placed an instance of addHistCall() to the beginning of all of the CSE451 APIs. I also added an instance of addToHistory() inside incStatus, so that it would log all of the return events for all of the functions.

In /public/sdk/inc/ntexapi.h, I modified the definition of the SYSTEM\_CSE451\_INFORMATION struct. In addition to all of the information that I had for project 1, I added an array of events (History) and a number of events (NumEvents).

I also modified the NtQuerySystemInformation function in the sysinfo.c file. In addition to copying the function-status count information that it did for project 1, it now copies over the oldest NumEvents (parameter in SYSTEM\_CSE451\_INFORMATION) to the struct. When it copies the NumEvents into SYSTEM\_CSE451\_INFORMATION.History, it removes them from the system buffer. It frees all of the completely removed blocks, and moves the pointers to the unfreed blocks up in the system chunked history accordingly. If a buffer is only partially used up, it then performs memcpy on all of the remaining chunks to move the events up the history buffer so that there isn’t in offset in the first chunk or any subsequent chunks. That way there is no wasted space due to an offset on the first chunk of history.

Please note that the history buffer holds the oldest N events, and places a BufferOveflow event at the end if it reaches its memory limit. In this instance, the oldest N-1 events remain unchanged until NtQuerySystemInformation is called.

I also modified test.c, so that it takes in an argument regarding the number of bytes to allocate for history. It then prints out the history in a very clear and readable fashion.