AV Search & Monitoring Program

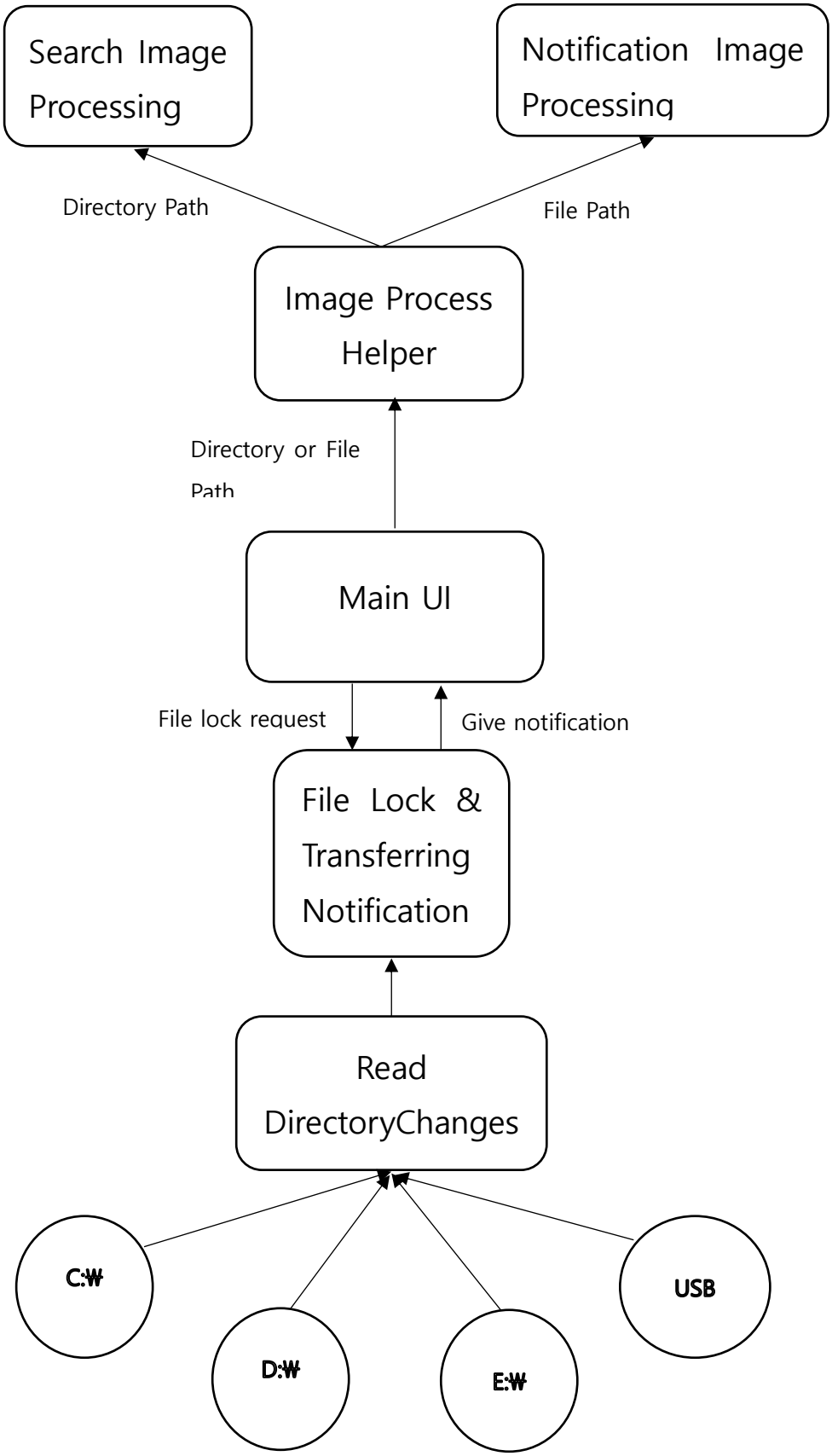
Seunghwan Han

- AV search & monitoring program

1. Primary function

- Find AV video in the hard disk
- Monitor the whole hard disk if there are added Adult video
- Cannot play the video seemed to be that kind of video
- Can react to USB insertion
- Self protect function (Task Manager -> Process terminate)

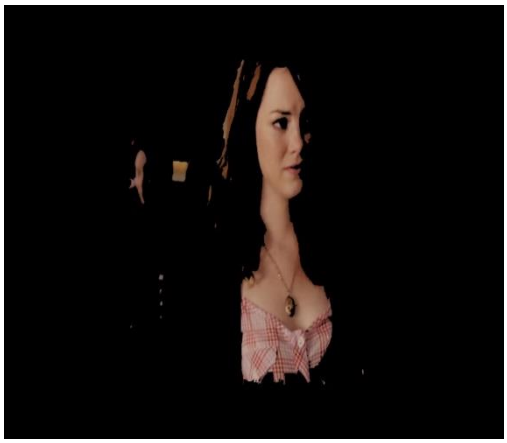
2. Structure of program



3. The way of implemation

- Process of judging the target video

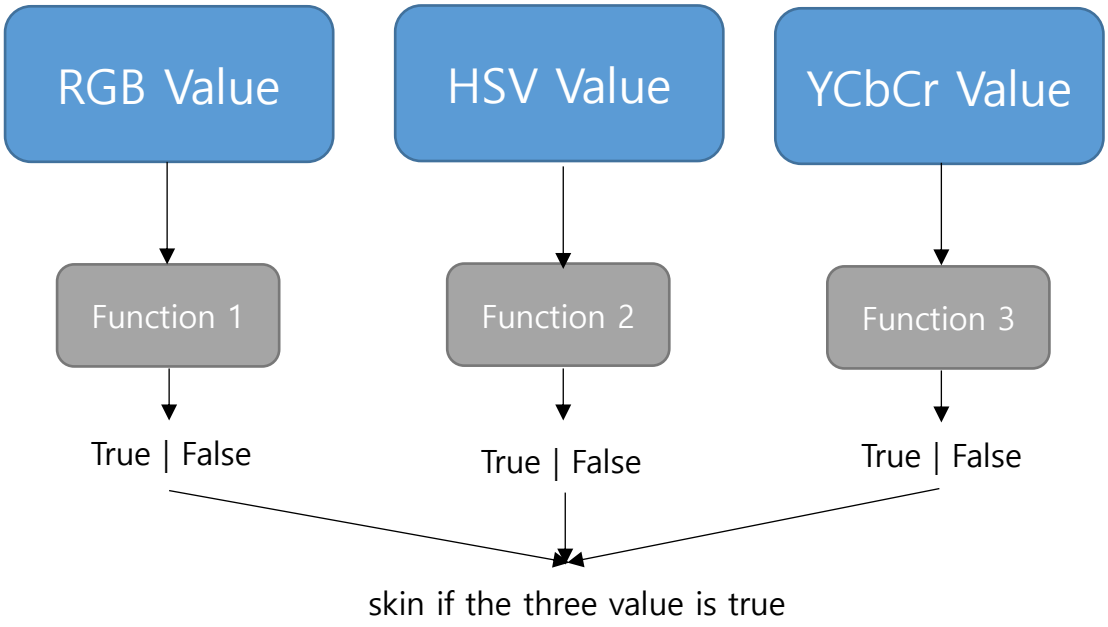
1. Use Grabcut function(OpenCV) - divide Human & things



```
GrabcutHelper helper;
bool isFound = false;

helper.setImage(frame, winName);
helper.setRect();

int iterCount = helper.getIterCount();
int newIterCount = helper.doGrabcut();
if (newIterCount > iterCount)
    helper.setTransformedImage();
```

```
int GrabcutHelper::doGrabcut(){
    if (isInitialized)
        grabCut(*image, mask, rect, bgdModel, fgdModel, 1);
    else{
        if (rectState != true)
            return iterCount;

        grabCut(*image, mask, rect, bgdModel, fgdModel, 1, GC_INIT_WITH_RECT);
        isInitialized = true;
    }
    iterCount++;

    return iterCount;
}
```

2. Skin check algorithm - translate the RGB to HSV, YCbCr and compare the each value if they are in the range.



Ex)

```
bool R1(int R, int G, int B) {
    bool e1 = (R>95) && (G>40) && (B>20) && ((max(R, max(G, B)) - min(R, min(G, B)))>15) && (abs(R - G)>15) && (R>G) && (R>B);
    bool e2 = (R>220) && (G>210) && (B>170) && (abs(R - G) <= 15) && (R>B) && (G>B);
    return (e1 || e2);
}

bool R2(float Y, float Cr, float Cb) {
    bool e3 = Cr <= 1.5862*Cb + 20;
    bool e4 = Cr >= 0.3448*Cb + 76.2069;
    bool e5 = Cr >= -4.5652*Cb + 234.5652;
    bool e6 = Cr <= -1.15*Cb + 301.75;
    bool e7 = Cr <= -2.2857*Cb + 432.85;
    return e3 && e4 && e5 && e6 && e7;
}

bool R3(float H, float S, float V) {
    return (H<25) || (H > 230);
}
```

```
for (int i = 0; i < src.rows; i++) {
    for (int j = 0; j < src.cols; j++) {

        Vec3b pix_bgr = src.ptr<Vec3b>(i)[j];
        int B = pix_bgr.val[0];
        int G = pix_bgr.val[1];
        int R = pix_bgr.val[2];
        // apply rgb rule
        bool a = R1(R, G, B);

        Vec3b pix_ycrcb = src_ycrcb.ptr<Vec3b>(i)[j];
        int Y = pix_ycrcb.val[0];
        int Cr = pix_ycrcb.val[1];
        int Cb = pix_ycrcb.val[2];
        // apply ycrcb rule
        bool b = R2(Y, Cr, Cb);

        Vec3f pix_hsv = src_hsv.ptr<Vec3f>(i)[j];
        float H = pix_hsv.val[0];
        float S = pix_hsv.val[1];
        float V = pix_hsv.val[2];
        // apply hsv rule
        bool c = R3(H, S, V);

        if (!(a&&b&&c))
            dst.ptr<Vec3b>(i)[j] = cblack;

    }
}
```

```
Mat tempImage = helper.getConvertedImage();
int totalCount = getCount(tempImage);

Mat skin = GetSkin(tempImage);
int skinCount = getCount(skin);

double skinRatio = (double)((double)skinCount / (double)totalCount);

if (skinRatio >= pInfo->choiceLevel)
{
    isFound = true;
    break;
}
```

- Realtime monitoring & blocking the play

1. Realtime monitoring - Use ReadDirectoryChangesW API

Retrieves information that describes the changes within the specified directory.

```
BOOL WINAPI ReadDirectoryChangesW(
  _In_        HANDLE                          hDirectory,
  _Out_       LPVOID                          lpBuffer,
  _In_        DWORD                           nBufferLength,
  _In_        BOOL                            bWatchSubtree,
  _In_        DWORD                           dwNotifyFilter,
  _Out_opt_   LPDWORD                         lpBytesReturned,
  _Inout_opt_ LPOVERLAPPED                    lpOverlapped,
  _In_opt_    LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

```
DWORD dwBytes = 0;

// This call needs to be reissued after every APC.
BOOL success = ::ReadDirectoryChangesW(
    m_hDirectory,                       // handle to directory
    &m_Buffer[0],                       // read results buffer
    m_Buffer.size(),                    // length of buffer
    m_bChildren,                        // monitoring option
    FILE_NOTIFY_CHANGE_FILE_NAME,       // filter conditions
    &dwBytes,                           // bytes returned
    &m_Overlapped,                      // overlapped buffer
    &NotificationCompletion);           // completion routine
```

```
VOID CALLBACK CReadChangesRequest::NotificationCompletion(
    DWORD dwErrorCode,                              // completion code
    DWORD dwNumberOfBytesTransfered,                // number of bytes transferred
    LPOVERLAPPED lpOverlapped)                      // I/O information buffer
{
    CReadChangesRequest* pBlock = (CReadChangesRequest*)lpOverlapped->hEvent;

    if (dwErrorCode == ERROR_OPERATION_ABORTED)
    {
        ::InterlockedDecrement(&pBlock->m_pServer->m_nOutstandingRequests);
        delete pBlock;
        return;
    }

    // Can't use sizeof(FILE_NOTIFY_INFORMATION) because
    // the structure is padded to 16 bytes.
    _ASSERTE(dwNumberOfBytesTransfered >= offsetof(FILE_NOTIFY_INFORMATION, FileName) + sizeof(WCHAR));

    // This might mean overflow? Not sure.
    if (!dwNumberOfBytesTransfered)
        return;

    pBlock->BackupBuffer(dwNumberOfBytesTransfered);

    // Get the new read issued as fast as possible. The documentation
    // says that the original OVERLAPPED structure will not be used
    // again once the completion routine is called.
    pBlock->BeginRead();

    pBlock->ProcessNotification();
}
```

-> Call the completion routine that passed to the API, and then do some work for judging and re-register the ReadDirectoryChanges when changes occur.

2. Blocking the play

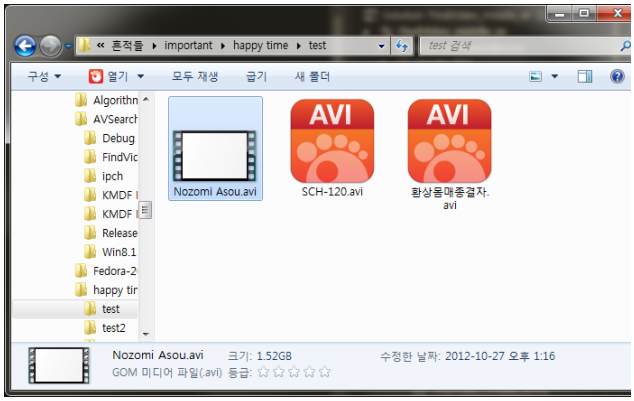
Locks the specified file for exclusive access by the calling process.

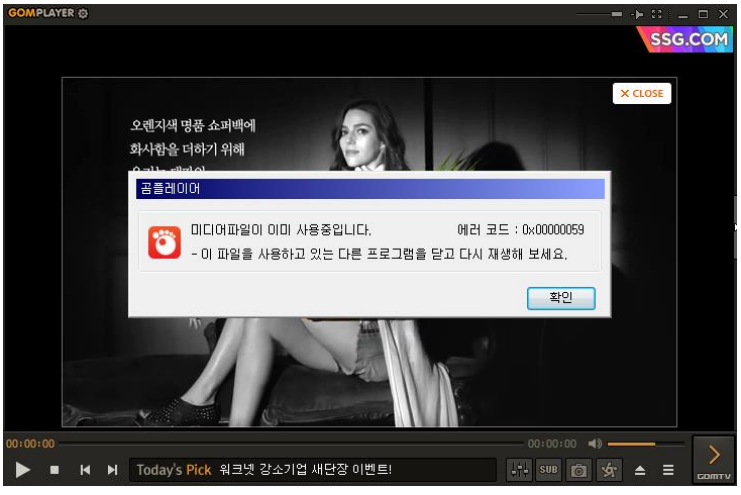
```
BOOL WINAPI LockFile(
  _In_ HANDLE hFile,
  _In_ DWORD  dwFileOffsetLow,
  _In_ DWORD  dwFileOffsetHigh,
  _In_ DWORD  nNumberOfBytesToLockLow,
  _In_ DWORD  nNumberOfBytesToLockHigh
);
```

```
BOOL FileLockThread::MyFileLock()
{
    for (auto it = fileHandles.begin(); it != fileHandles.end(); it++)
        LockFile(*it, 0, 0, GetFileSize(*it, 0), 0);
    return TRUE;
}
```

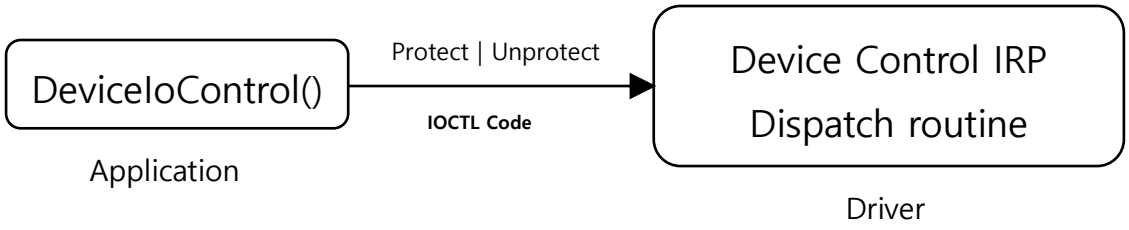
-> Use LockFile API with file handle

Ex)





- Self protect function - using kernel-level programming



-> Communicate between application and driver using IOCTL code

Sends a control code directly to a specified device driver, causing the corresponding device to perform the corresponding operation.

Syntax

C++

```
BOOL WINAPI DeviceIoControl(
  _In_        HANDLE       hDevice,
  _In_        DWORD        dwIoControlCode,
  _In_opt_    LPVOID       lpInBuffer,
  _In_        DWORD        nInBufferSize,
  _Out_opt_   LPVOID       lpOutBuffer,
  _In_        DWORD        nOutBufferSize,
  _Out_opt_   LPDWORD      lpBytesReturned,
  _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

Use this API to communicate from application to driver.

```
Result = DeviceIoControl (
    TcDeviceHandle,
    TD_IOCTL_PROTECT_NAME_CALLBACK,
    &ProtectNameCallbackInput,
    sizeof(ProtectNameCallbackInput),
    NULL,
    0,
    &BytesReturned,
    NULL
);
```

```
BOOL Result = DeviceIoControl (
    TcDeviceHandle,
    TD_IOCTL_UNPROTECT_CALLBACK,
    &UnprotectCallbackInput,
    sizeof(UnprotectCallbackInput),
    NULL,
    0,
    &BytesReturned,
    NULL
);
```

```
switch (Ioctl)
{
case TD_IOCTL_PROTECT_NAME_CALLBACK:

    Status = TdControlProtectName (DeviceObject, Irp);
    break;

case TD_IOCTL_UNPROTECT_CALLBACK:

    Status = TdControlUnprotect (DeviceObject, Irp);
    break;
```

The **ObRegisterCallbacks** routine registers a list of callback routines for thread, process, and desktop handle operations.

Syntax

C++

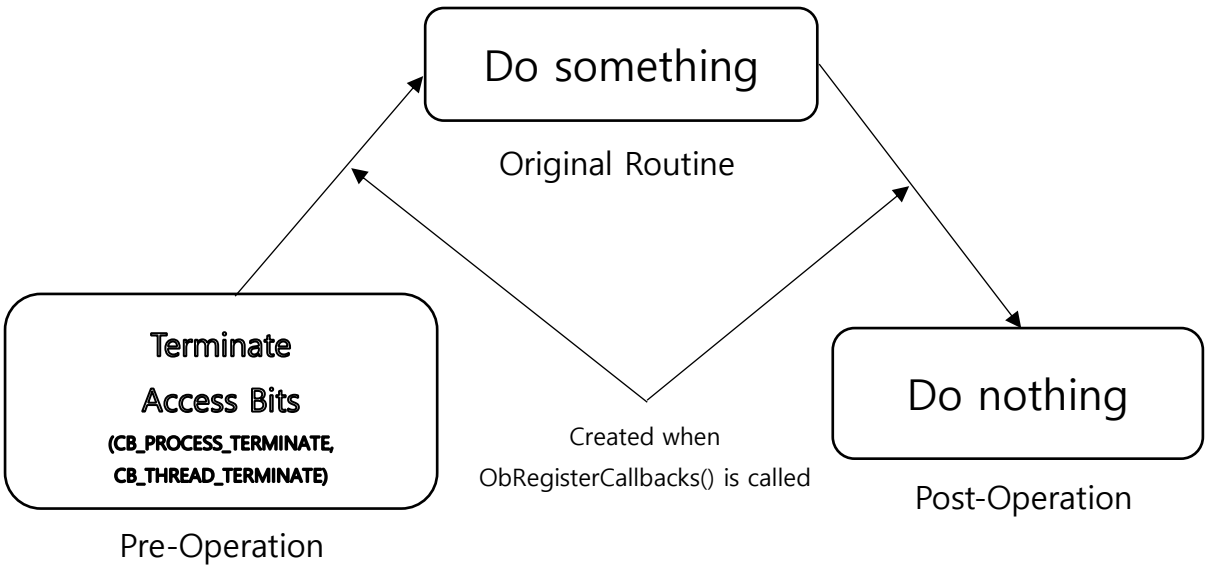
```
NTSTATUS ObRegisterCallbacks(
  _In_  POB_CALLBACK_REGISTRATION CallBackRegistration,
  _Out_ PVOID                     *RegistrationHandle
);
```

Parameters

CallBackRegistration [in]
A pointer to an OB_CALLBACK_REGISTRATION structure that specifies the list of callback routines and other registration information.

RegistrationHandle [out]
A pointer to a variable that receives a value that identifies the set of registered callback routines. The caller passes this value to the ObUnRegisterCallbacks routine to unregister the set of callbacks.

-> Create Filtering routine by using this API



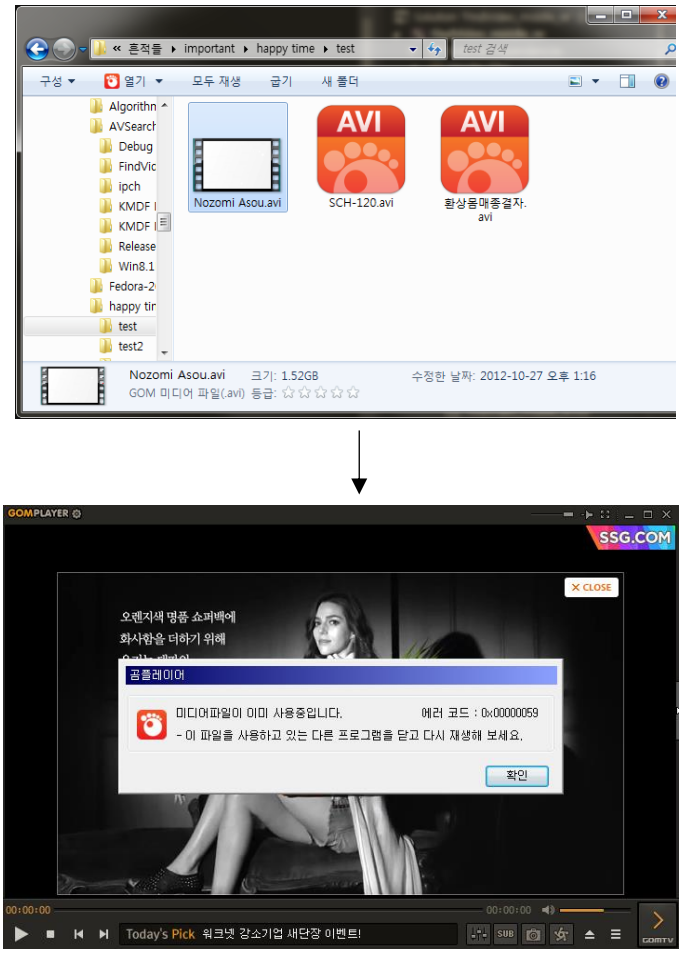
```
CBOperationRegistrations[0].ObjectType = PsProcessType;
CBOperationRegistrations[0].Operations |= OB_OPERATION_HANDLE_CREATE;
CBOperationRegistrations[0].Operations |= OB_OPERATION_HANDLE_DUPLICATE;
CBOperationRegistrations[0].PreOperation = CBTdPreOperationCallback;
CBOperationRegistrations[0].PostOperation = CBTdPostOperationCallback;

CBOperationRegistrations[1].ObjectType = PsThreadType;
CBOperationRegistrations[1].Operations |= OB_OPERATION_HANDLE_CREATE;
CBOperationRegistrations[1].Operations |= OB_OPERATION_HANDLE_DUPLICATE;
CBOperationRegistrations[1].PreOperation = CBTdPreOperationCallback;
CBOperationRegistrations[1].PostOperation = CBTdPostOperationCallback;


RtlInitUnicodeString (&CBAltitude, L"1000");

CBObRegistration.Version                    = OB_FLT_REGISTRATION_VERSION;
CBObRegistration.OperationRegistrationCount = 2;
CBObRegistration.Altitude                   = CBAltitude;
CBObRegistration.RegistrationContext        = &CBCallbackRegistration;
CBObRegistration.OperationRegistration      = CBOperationRegistrations;


 Status = ObRegisterCallbacks (
     &CBObRegistration,
     &pCBRegistrationHandle
 );
```

-> Register Pre-Operation & Post-Operation routine

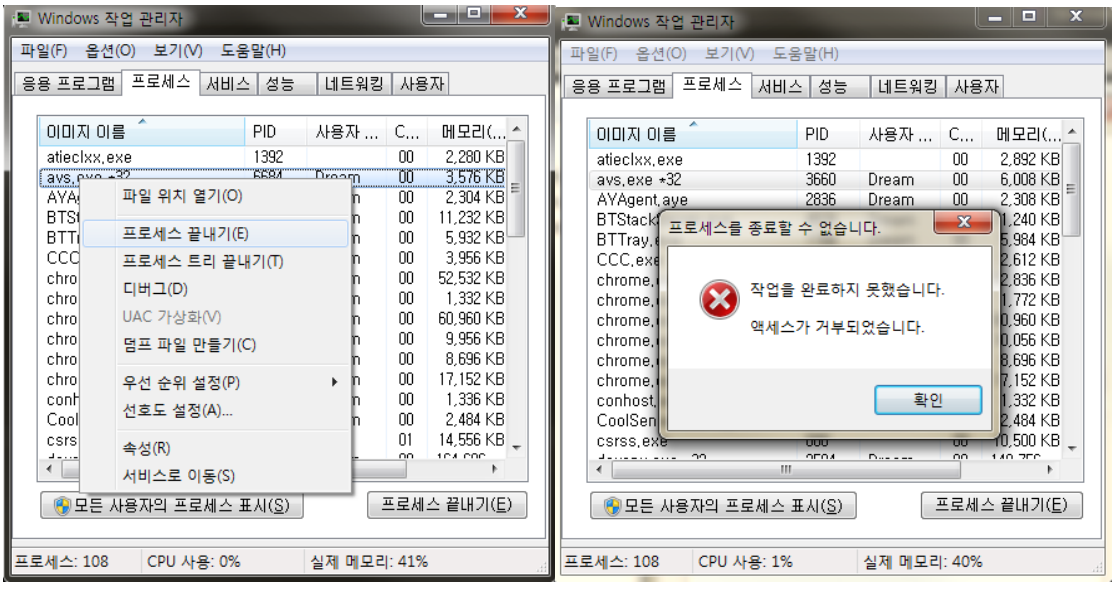
```
OB_PREOP_CALLBACK_STATUS
CBTdPreOperationCallback (
    _In_ PVOID RegistrationContext,
    _Inout_ POB_PRE_OPERATION_INFORMATION PreInfo
])
```

```
// Filter only if request made outside of the kernel
if (PreInfo->KernelHandle != 1) {
    *DesiredAccess &= ~AccessBitsToClear;
    *DesiredAccess |=  AccessBitsToSet;
}
```

-> remove the termination flag in Pre-Operation Routine when our program's thread or process handle is called



<Cannot play the AV>



<Can do only normal program exit - cannot terminate the program abnormaly>