최종보고서

SMT
(System Monitoring Tool
: 시스템자원 모니터링 툴)

팀 명 : Point of C++
팀 장 : 한소라(11학번)
팀 원 : 이재익(09학번)
한우영(09학번)
김은지(11학번)

지도교수 : 양정모 교수님

목 차

1 . 서론

1-1 연구 배경 및 목적

우리는 현재 어떤 일이든 컴퓨터를 통해 처리하고 있고 생활의 거의 대부분이 컴퓨터와 연관 되어있다고 생각합니다. 그런데 최근 각종 악성코드로 인해 시스템 자원을 고갈시키는 등의 방법으로 시스템의 운영을 방해하거나 정보자료를 훼손하여 많은 사용자들에게 피해를 주고 있습니다. 따라서 본 프로젝트를 통해 시스템 자원의 과부하 등 시스템 운영상태를 실시간으로 감시하고, 각종 침해를 실시간으로 통제함으로써 피해를 최소화하고, 시스템의 안정적인 운영을 도모하려고 합니다. 더불어 MEMORY, CPU에 관한 시스템함수에 대한 학습을 통해 실력을 쌓고 체계적인 진행을 통해 C프로그램의 전반적인 학습으로 프로그래밍 실력의 향상을 목표에 두고 임할 것입니다.

1-2 연구 내용

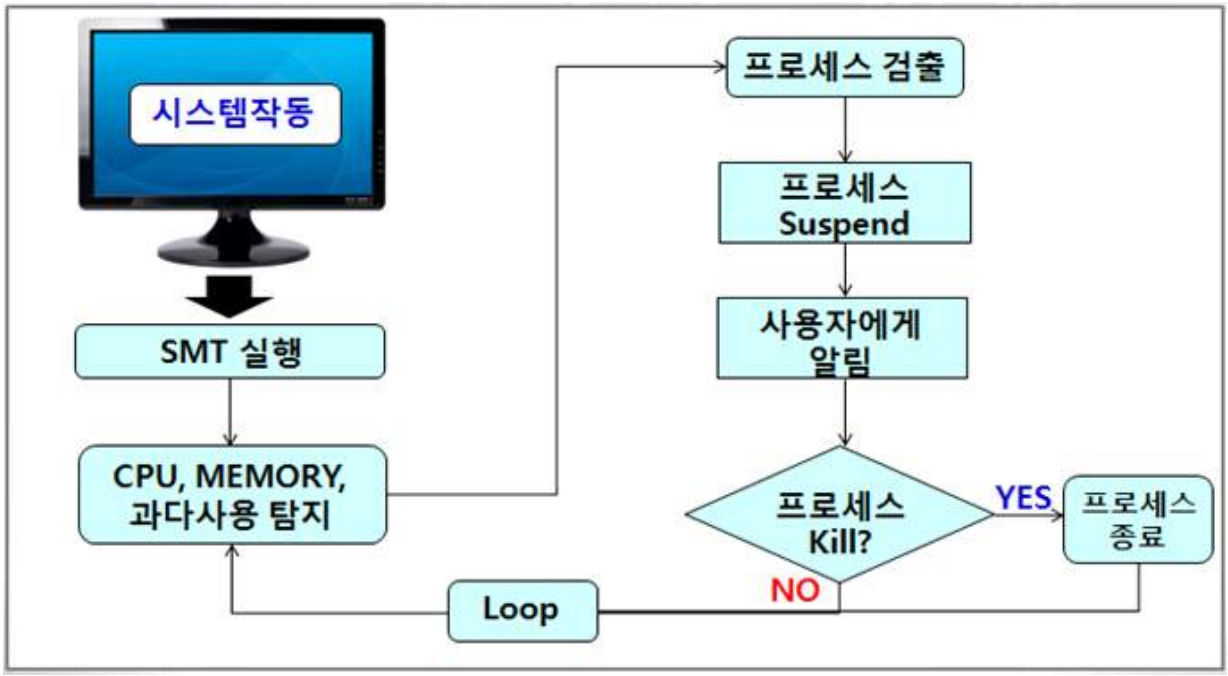
우리의 연구목적은 실시간으로 과다 사용되고 있는 프로세스를 검출하여 시스템자원을 통제하여 시스템운영의 피해를 최소화하기 위한 프로그램인 시스템 모니터링 툴(System Monitoring Tool : SMT) 제작입니다. SMT에서는 시스템 함수인 GlobalMemoryStatusEx( )와 GetCpuUsage( )를 사용하여 시스템 모니터링을 실행합니다. 컴퓨터가 동작하고 SMT를 실행시켜 시스템자원인 MEMORY, CPU의 프로세스별 사용량을 모니터링하여 사용량이 70%이상으로 과다사용이 되고 있는지를 탐지합니다. 사용량이 70%이상에 해당되는 프로세스를 검출하여 자동으로 Suspend하며, 사용자에게 메시지 박스와 알림음을 통해 경보를 하고 해당하는 프로세스의 상세정보를 출력합니다. 사용자에게 해당 프로세스를 종료 시킬 것인지 질의를 통해 사용자의 의사를 묻고, 계속 실행할 경우 프로세스의 Suspend상태를 풀어주고 프로세스의 과다사용을 다시 탐지합니다.

1-3 개발 환경

- 운영체제 : Windows7
- 프로그래밍 언어 : C/C++
- 개발도구 : Microsoft Visual Studio 2008/2010

2 . 본론

2-1 프로그램구성



< 그림 1 > 시스템 개요도

본 시스템 모니터링 툴(SMT)은 컴퓨터가 동작하고 SMT를 실행시켜 시스템자원인 MEMORY, CPU의 프로세스별 사용량이 70%으로 과다사용이 되고 있는지를 탐지합니다. 해당되는 프로세스를 검출하여 자동으로 Suspend하며, 사용자에게 메시지 박스를 통해 경보를 하고 프로세스의 상세정보를 출력합니다. 사용자에게 해당 프로세스를 종료 시킬 것인지 질의를 합니다. 계속 실행할 경우 프로세스의 Suspend상태를 풀어주고 프로세스의 과다사용을 다시 탐지합니다.

2-2 MEMORY값 측정 함수

시스템의 메모리 값을 얻어오기 위하여 GlobalMemoryStatusEx( ) API를 사용하였습니다.

▶ API 소개

GlobalMemoryStatusEx API는 가상메모리, 페이지파일, 실제 메모리(물리메모리) 등의 정보를 얻고자 할 때 사용합니다. 구조체는 MEMORYSTATUSEX가 사용됩니다.

▶ API 선언

Public Dexlare Sub GlobalMemoryStatusEx Lib "kernel32"(lpBuffer As MEMORYSTATUSEX)

▶ 구조체 선언

Public Type MEMORYSTATUSEX

dwLength	As Long	· 구조체의 크기
dwMemoryLoad	As Long	· 메모리의 사용량(백분율)
ullTotalPhys	As Currency	· 실제 메모리의 총 크기
ullAvailPhys	As Currency	· 실제 메모리의 남은 크기
ullTotalPagefile	As Currency	· 페이지 파일의 총 크기
ullAvailPageFile	As Currency	· 페이지 파일의 남은 크기
ullTotalVirtual	As Currency	· 가상메모리의 총 크기
ullAvailVirtual	As Currency	· 가상메모리의 남은 크기
ullAvailExtendedVirtual	As Currency	· 확장된 가상 메모리의 남은 크기

▶ 설명

GlobalMemoryStatusEx( ) API를 사용해 얻어온 값들의 단위는 메가바이트가 아니므로, 단위를 맞추기 위해 102.4로 나누고 거기에 1000을 곱한 다음 또다시 1024로 나눕니다.
또, 구조체 MEMORYSTATUSEX 에서는 각 메모리의 크기와 사용 가능한 공간만을 구할 수 있는데 저 두 대를 응용하여 메모리 사용량을 구합니다.(메모리 크기 - 사용가능한 공간)
GlobalMemoryStatus( ) API는 메모리의 크기가 4GB 이상이 되면 값이 이상하게 나온다고 하여 GlobalMemoryStatusEx( ) API를 사용하였습니다.

2-3 CPU값 측정 함수

레지스트를 통한 HKEY_PERPOMANCE_DATA 키에 값을 가져오는 방법입니다. 실제 보이지 않는 키이기 때문에 regedit로 해서 HKEY_PERFORMANCE_DATA를 조회할 수는 없습니다.
HKEY_PERFORMANCE_DATA에 있는 값들을 통해서 성능에 관련된 정보를 획득할 수 있었습니다.

▶ 정의되는 함수

LPBYTE GetPerformanceData(LPTSTR src);

```
int GetCounterValue(const int&amp;amp; objId, const int&amp;amp; counterId, const char* instanceName,
                    PERF_DATA_BLOCK **dataBlock, LONGLONG &amp;amp;value);
double GetCpuUsage(const char* process);
```

정의되는 함수는 총 3개지만 실제 사용하는 함수는 GetCpuUsage( )입니다. 다른 함수는 통계정보를 얻기 위하여 사용합니다.

▶ GetCpuUsage( ) 함수

GetCpuUsage( ) 함수를 통해서 CPU 사용량을 얻을 수 있습니다. GetCounterValue( ) 함수에 의해서 HKEY_PERFORMANCE_DATA 키에 있는 원하는 카운터 객체 (PERF_COUNTER_BLOCK)를 얻을 수 있고, 그 곳에서 원하는 값을 얻을 수 있습니다. 이렇게 얻은 값을 통해서 CPU사용율을 계산할 수 있습니다.

▶ GetCounterValue( ) 함수

GetCounterValue( ) 함수는 HKEY_PERFORMANCE_DATA치에서 원하는 카운터 객체를 찾고 거기에서을 얻는 것입니다. 사용되는 인자가 많습니다. 첫 번째 인자는 찾는 객체 ID이고 두 번째 인자는 찾고자하는 카운터 객체 ID이고 세 번째 인자는 찾고자하는 인스턴스 이름입니다. 네 번째 인자는 성능데이터 값(PERF_DATA_BLOCK)이 저장되고 다섯 번째 인자는 카운트 블록(PERF_COUNTER_BLOCK)의 값이 저장됩니다. 앞의 세 개 인자는 입력인자이고 나머지 두 개 인자는 출력인자입니다. 반환 값은 에러가 발생시 -1이 반환되고 성공이면 0이 반환됩니다.

▶ GetCounterValue( ) 함수 구조체

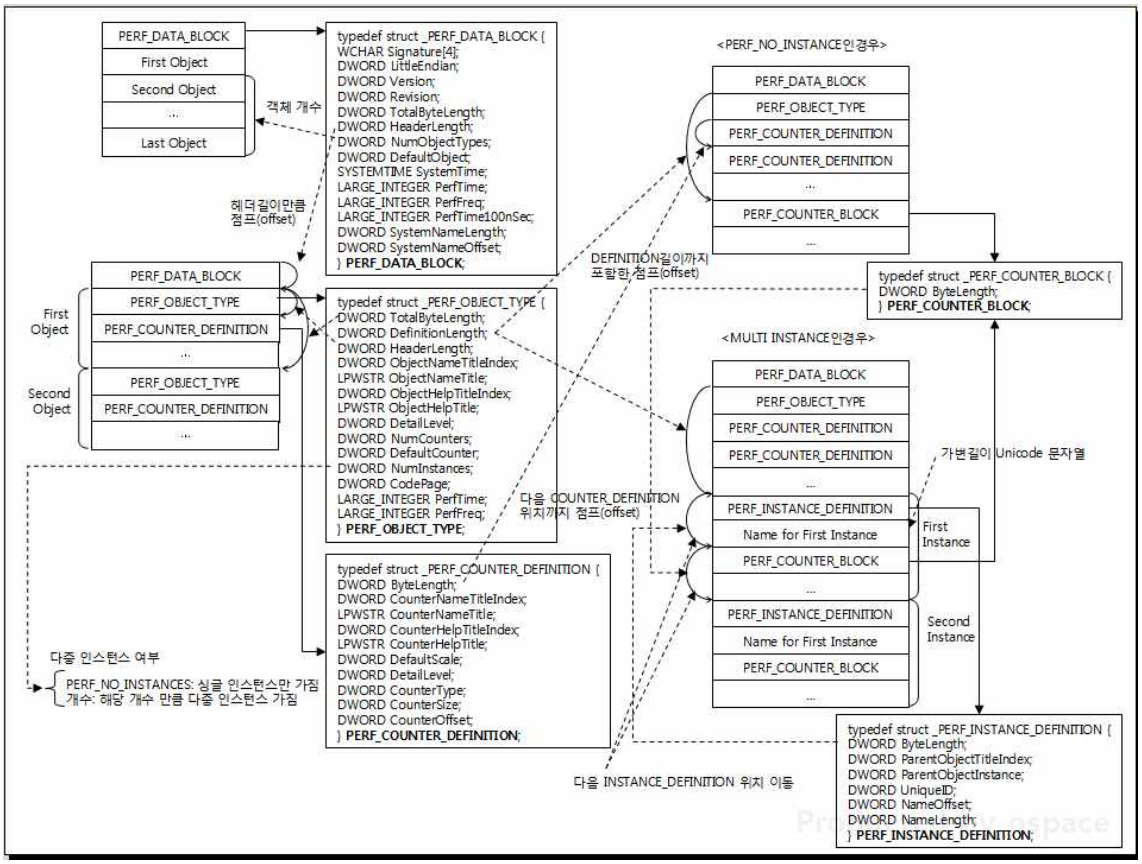
PERF_DATA_BLOCK

PERF_OBJECT_TYPE

PERF_COUNTER_DEFINITION

PERF_INSTANCE_DEFINITION

PERF_COUNTER_BLOCK



< 그림 2 > 구조체의 관계

기본적인 객체는 PERF_DATA_BLOCK에서 시작됩니다. 그 뒤로 여러 객체들이 따라오는데 각 객체는 PERF_OBFECT_TYPE를 헤더로 갖습니다. 그리고 각 객체에는 여러 카운터 객체가 포함됩니다. 그리고 각 객체에 인스턴스 개수에 따라서 여러 인스턴스 객체가 포함되어 있습니다.

PERF_DATA_BLOCK에서 다음 객체(PERF_OBJECT_TYPE)를 찾으려면 PERF_DATA_BLOCK의 HeaderLength만큼 이동하면 되고 여러 객체에서 다음 객체를 찾으려면 PERF_OBJECT_TYPE에서 TotalByteLength만큼 이동하면 됩니다. 그리고 각 객체(PERF_DATA_BLOCK)에서 카운터 정의 객체(PERF_COUNTER_DEFINITION)을 찾으려면 PERF_DATA_BLOCK에서 PERF+DATA_BLOCK의 HeaderLength만큼 이동하면 됩니다. 카운터 객체가 여러 개가 오는데 다음 카운터 객체(PERF_COUNTER_DEFINITION)를 찾으려면 PERF_COUNTER_DEFINITION의 ByteLength만큼 이동하면 됩니다.

그리고 다중 인스턴스라면 첫 번째 인스턴스 객체(PERF_INSTANCE_DEFINITION)를 찾으려면 PERF_OBJECT_TYPE에서 PERF_OBJECT_TYPE의 DefinitionLength만큼 이동하면 됩니다. 그리고 다음 인스턴스 객체를 찾으려면 PERF_INSTANCE_DEFINITION에서 PERF_INSTANCE_DEFINIRION의 ByteLength만큼 이동하면 PERF_INSTANCE_DEFINITION을 얻을 수 있습니다.

다중 객체가 아니면 간단합니다. 첫 번째 인스턴스 객체(PERF_INSTANCE_DEFINITION) 위치는 앞에서 구한

값으로 가지면 되고 다음은 PERF_COUNTER_BLOCK에서 ByteLength만큼 이동하면 됩니다.

▶ GetPerformanceData( ) 함수

GetPerformanceData( ) 함수는 레지스트리에 있는 HKEY_PERFORMANCE_DATA 키에서 원하는 객체 값을 얻어오는 것이다. 이 함수는 GetCounterValue( ) 함수에 비해서는 간단합니다. 입력되는 인자는 원하는 인스턴스 객체명을 받습니다. 반환되는 값은 PERF_DATA_BLOCK의 포인터를 반환합니다. 만약 에러가 발생한다면 NULL값이 반환됩니다.

2-4 프로그램 실행화면

▶ CPU 및 메모리 실시간 모니터링 구현부분 소스코드

```
// CPU 및 메모리 실시간 모니터링 소스
BOOL SystemMonitoring( ) {

        HANDLE hProcessSnap;
        HANDLE hProcess;
        PROCESSENTRY32 pe32;
        MEMORYSTATUSEX statex;
        statex.dwLength = sizeof ( statex );
        GlobalMemoryStatusEx ( &statex );
        PROCESS_MEMORY_COUNTERS pmc;
        DWORD Code = 0;
        int i=0, j=0, ProcessID, suspend;
        char * processname;
        DWORD dwPriorityClass;


        hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
        //32bit인 process의 정보를 가져옴.


        if( hProcessSnap == INVALID_HANDLE_VALUE ) {
                printError( TEXT("CreateToolhelp32Snapshot (of processes)") );
                return( FALSE );

        }

        pe32.dwSize = sizeof( PROCESSENTRY32 );                    //pe23 사이즈 초기화

        if( !Process32First( hProcessSnap, &pe32 ) ) {
```

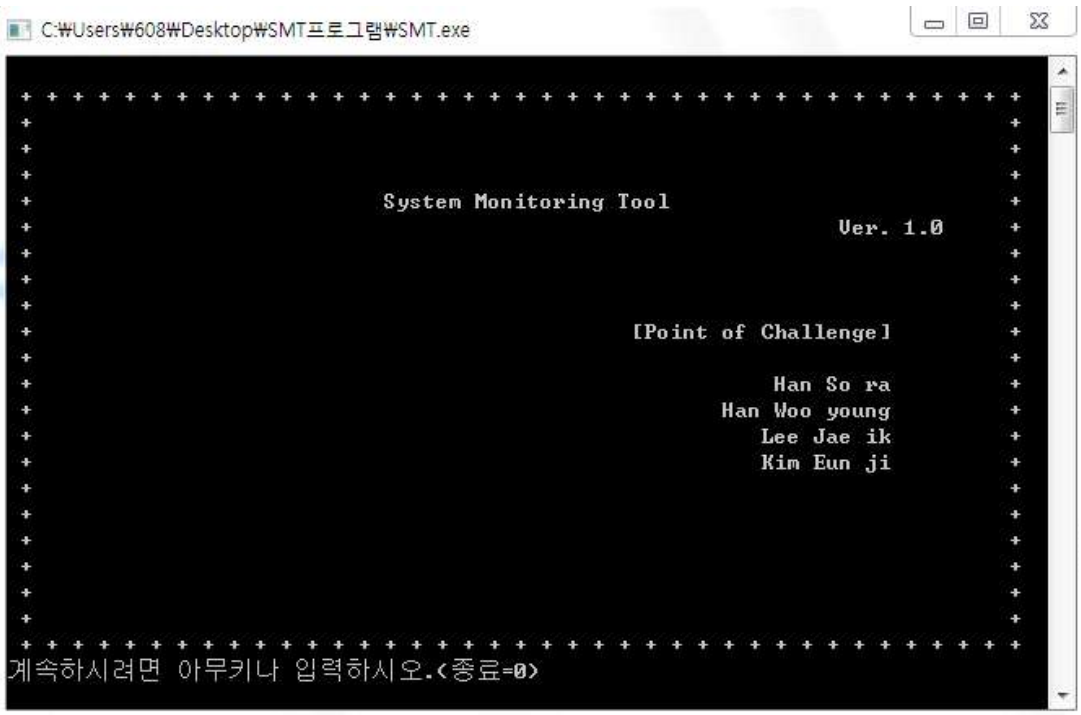
```
            printError( TEXT("Process32First") );
            CloseHandle( hProcessSnap );
            return( FALSE );
}

do {
            hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
// 프로세스의 Handle값을 얻어옴.
                                                PROCESS_VM_READ,
                                                FALSE, pe32.th32ProcessID  );
            dwPriorityClass = GetPriorityClass(hProcess);
            // 프로세스의 우선 순위값을 반환.

            processname=pe32.szExeFile;
            strtok(processname,".");

            if(0==i%3 || i==0) {
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );
                        printf("₩n                                  System Monitoring Tool₩n");
                        printf("₩n                                   [Point of Challenge]₩n");
                        printf("₩n                                    모니터링중 입니다.₩n");
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );
                        }
            else if(0==i%2) {
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );
                        printf("₩n                                  System Monitoring Tool₩n");
                        printf("₩n                                   [Point of Challenge]₩n");
                        printf("₩n                                    모니터링중 입니다...₩n");
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );}
            else {
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );
                        printf("₩n                                  System Monitoring Tool₩n");
                        printf("₩n                                   [Point of Challenge]₩n");
                        printf("₩n                                    모니터링중 입니다..₩n");
                        printf("₩n₩n₩n₩n₩n₩n================================
                        ========================₩n" );}

            int processID=0;
            CCpuUsage usageA;
            CCpuUsage usageB;
```

```
for( j=0;j<2;j++ )
{
        int SystemWideCpuUsage = usageA.GetCpuUsage();
//전체프로세스 CPU 사용량
int ProcessCpuUsageByName = usageB.GetCpuUsage( processname );
//프로세스 CPU 사용량

Sleep( 250 );

if( ProcessCpuUsageByName > 70 ) { // 프로세스 CPU 점유율 70%가 넘었을때
      ProcessID = pe32.th32ProcessID;
      suspend = DebugActiveProcess(ProcessID);
        //프로세스 일시중지

     printf("₩a"); printf("₩a"); printf("₩a");      //경고음
     MessageBoxW( NULL, L"CPU점유율이 최대 70%가 넘는 프로세스가 발견되었습니다.",
     L"Warning", 0 ); //경고메세지
     system("cls");

_tprintf(TEXT("₩n₩n₩n₩n₩n₩n================================
      ================" ) );
_tprintf( TEXT("₩n                    PROCESS NAME :  %s"), pe32.szExeFile );
_tprintf( TEXT("₩n                    Process ID   :  %u"), pe32.th32ProcessID );


if( j==1 ) printf("₩n                    전체 프로세스 CPU 사용량 :  %d", SystemWideCpuUsage);
if( j==1 ) printf("₩n                 프로세스 CPU 사용량      :  %d", ProcessCpuUsageByName);
        _tprintf(TEXT("₩n₩n₩n₩n₩n₩n================================
      ================" ) );

          printf("₩n                    프로세스를 중지하였습니다.");
          printf("₩n                    프로세스를 종료하시려면 1");
          printf("₩n                    다시 되돌리시려면 2를 입력해주세요.");
          scanf("%d",&suspend);

                if( suspend == 1 ) {
                      KillProcess(processname,ProcessID);
//프로세스kill함수 호출
                      printf("                    프로세스를 종료하였습니다.₩n");
                      Sleep( 1000 );
                      printf("                    모니터링을 다시 시작합니다.");
                      Sleep( 1000 );
                }
```

```
if( suspend == 2 ) {
                DebugActiveProcessStop(ProcessID);
//프로세스 일시중지 해제
                printf("                프로세스를 원래대로 되돌렸습니다.₩n");
                        Sleep( 1000 );
                        printf("                모니터링을 다시 시작합니다.");
                        Sleep( 1000 );
                }
        }
}

if( GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc)) ) {
// 프로세스 메모리정보 구하는 함수
if( (pmc.WorkingSetSize/DIV) > ((statex.ullAvailPhys/DIV)*0.75) ) {
//메모리 점유율 70%가 넘었을때
        ProcessID = pe32.th32ProcessID;
        suspend = DebugActiveProcess(ProcessID);
                                        //프로세스 일시중지
        printf("₩a"); printf("₩a"); printf("₩a");
                                //경고음
MessageBoxW( NULL, L"메모리점유율이 최대 70%가 넘는 프로세스가 발견되었습니다.",
    L"Warning", 0 );//경고메세지
                system("cls");
_tprintf(TEXT("₩n₩n₩n₩n₩n₩n==============================
    ================" ) );

_tprintf( TEXT("₩n                PROCESS NAME :  %s"), pe32.szExeFile );
_tprintf( TEXT("₩n                Process ID   :  %u"), pe32.th32ProcessID );
printf("₩n      최대작업메모리의 크기 : %*d KB₩n", WIDTH, statex.ullTotalPhys/DIV);
printf("₩n      프로세스의 작업메모리크기 : %d KB₩n", pmc.WorkingSetSize/DIV);
_tprintf(TEXT("₩n₩n₩n₩n₩n₩n==============================
    ================" ) );

        printf("₩n                프로세스를 중지하였습니다.");
        printf("₩n                프로세스를 종료하시려면 1");
        printf("₩n                다시 되돌리시려면 2를 입력해주세요.");
        scanf("%d",&suspend);
        if (suspend == 1) {
                KillProcess( processname, ProcessID );
                //프로세스kill함수 호출
                printf("                프로세스를 종료하였습니다.₩n");
                Sleep( 1000 );
                printf("                모니터링을 다시 시작합니다.");
                Sleep( 1000 );
                }
```

```
                        if (suspend == 2) {
                                DebugActiveProcessStop( ProcessID );
                //프로세스 일시중지 해제
                                printf("          프로세스를 원래대로 되돌렸습니다.₩n");
                                Sleep( 1000 );
                                printf("                    모니터링을 다시 시작합니다.");
                                Sleep( 1000 );
                        }
                    }
                }

                CloseHandle(hProcess);          // 프로세스의 핸들을 반환해주는 함수
                i++;
                system("cls");
        } while( Process32Next( hProcessSnap, &pe32 ) ) //엔트리 프로세스를 읽고 없을때까지 계속 읽음
        CloseHandle( hProcessSnap );        //스냅샷의 핸들을 반환해주는 함수
        return 0;
}
```

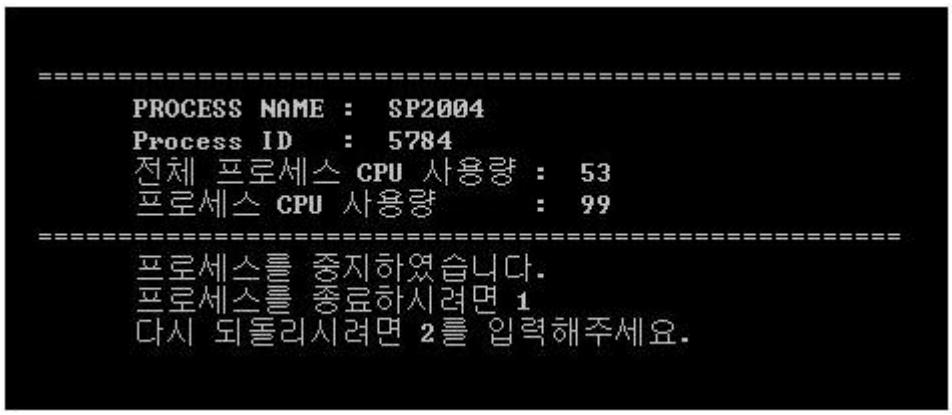
위의 프로그램 소스코드를 실행하게 되면 메인화면이 출력되고 0을 제외한 다른 값을 입력시켜 모니터링을 시작합니다.



< 그림 3 > 프로그램 메인화면

< 그림 4 > 프로그램 모니터링화면

프로그램이 프로세스별 CPU, MEMORY의 사용율을 모니터링하면서 과다 사용을 검출합니다. MEMORY, CPU의 프로세스별 사용량이 70%으로 과다사용이 확인되면 알림을 주고 프로세스를 종료를 원하면 프로세스를 종료시킵니다.



< 그림 5 > 프로세스Suspend 후 질의화면

3. 결론

3-1 결론 및 기대효과

시스템자원 모니터링 툴을 직접 프로그램으로 구현하는 과정에서 시스템 함수에 관한 지식을 습득하고 메모리 과부하 프로그램을 제작하여 컴퓨터에 직접 영향을 미치는 위험성들을 실험을 통해 프로그램 구동원리를 공부할 수가 있었으며 원도우 프로그래밍의 다양한 기법, 기술을 학습하여 좀 더 폭 넓은 기술개발 역량을 배량 하였습니다. 프로젝트를 진행하면서 처음에는 과연 수업시간에 배웠던 모든 것을 활용해서 우리가 할 수 있을까라는 의문점을 갖으면서도 계획을 세우고 끝까지 해보려는 노력과 팀워크가 힘을 더해 어려운 것도 잘 해결 할 수 있었습니다.

4. 부록

4-1. 소스코드

```
//include파일 및 전처리기 정의
#include "stdafx.h"
#include <windows.h>
#include <winuser.h>
#include <tlhelp32.h>
#include <tchar.h>
#include <stdio.h>
#include <string.h>
#include <Psapi.h>
#include <stdlib.h>
#include "CpuUsage.h"

#pragma comment( lib, "psapi" )
#define DIV 1024
#define WIDTH 7

//함수정의
BOOL KillProcess( char* processname, int a );  //프로세스kill 함수
BOOL SystemMonitoring( );                      //시스템 모니터링 함수
void printError( TCHAR* msg );                 //Error탐지 함수

//main함수
int main( void ) {

        int start;
        printf("₩n + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
        + + + +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                                              +₩n");
        printf(" +                  System Monitoring Tool                      +₩n");
        printf(" +                                              Ver. 1.0       +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                 [Point of Challenge]        +₩n");
        printf(" +                                                              +₩n");
        printf(" +                                        Han So ra             +₩n");
        printf(" +                                     Han Woo young            +₩n");
        printf(" +                                       Lee Jae ik             +₩n");
        printf(" +                                       Kim Eun ji             +₩n");
```

```
	printf("  +                                                          +₩n");
	printf("  +                                                          +₩n");
	printf("  +                                                          +₩n");
	printf("  +                                                          +₩n");
	printf("  +                                                          +₩n");
	printf("  +                                                          +₩n");
	printf("  + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
	 + + +₩n");
	printf("계속하시려면 아무키나 입력하시오.(종료=0)");
		scanf("%d",&start);
		if(start==0){system("cls"); printf("프로그램을 종료합니다.₩n"); exit(0);}
		system("cls");
	while( 1 ) {
		SystemMonitoring( );
	}
	return 0;
}

// CPU 및 메모리 실시간 모니터링 소스
BOOL SystemMonitoring( ) {

	HANDLE hProcessSnap;
	HANDLE hProcess;
	PROCESSENTRY32 pe32;
	MEMORYSTATUSEX statex;
	statex.dwLength = sizeof ( statex );
	GlobalMemoryStatusEx ( &statex );
	PROCESS_MEMORY_COUNTERS pmc;
	DWORD Code = 0;
	int i=0, j=0, ProcessID, suspend;
	char * processname;
	DWORD dwPriorityClass;


	hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
	//32bit인 process의 정보를 가져옴.


	if( hProcessSnap == INVALID_HANDLE_VALUE ) {
		printError( TEXT("CreateToolhelp32Snapshot (of processes)") );
		return( FALSE );

	}

	pe32.dwSize = sizeof( PROCESSENTRY32 );			//pe23 사이즈 초기화
```

```
if( !Process32First( hProcessSnap, &pe32 ) ) {
        printError( TEXT("Process32First") );
        CloseHandle( hProcessSnap );
        return( FALSE );
}

do {
        hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
// 프로세스의 Handle값을 얻어옴.
                                                PROCESS_VM_READ,
                                                FALSE, pe32.th32ProcessID  );
        dwPriorityClass = GetPriorityClass(hProcess);
        // 프로세스의 우선 순위값을 반환.

        processname=pe32.szExeFile;
        strtok(processname,".");

        if(0==i%3 || i==0) {
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );
                printf("₩n                              System Monitoring Tool₩n");
                printf("₩n                               [Point of Challenge]₩n");
                printf("₩n                                모니터링중 입니다.₩n");
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );
                }
        else if(0==i%2) {
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );
                printf("₩n                              System Monitoring Tool₩n");
                printf("₩n                               [Point of Challenge]₩n");
                printf("₩n                                모니터링중 입니다...₩n");
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );}
        else {
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );
                printf("₩n                              System Monitoring Tool₩n");
                printf("₩n                               [Point of Challenge]₩n");
                printf("₩n                                모니터링중 입니다..₩n");
                printf("₩n₩n₩n₩n₩n₩n₩n==============================
                ========================₩n" );}

        int processID=0;
```

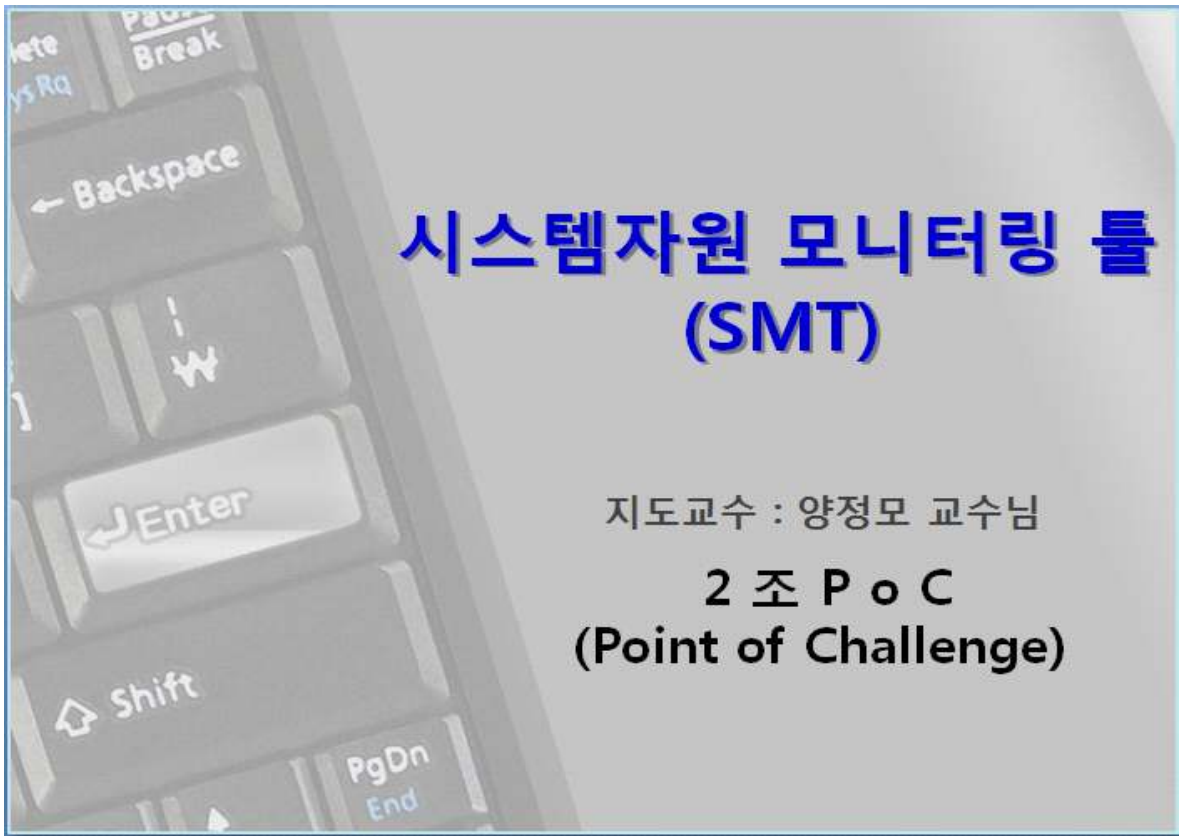
```
CCpuUsage usageA;
CCpuUsage usageB;

for( j=0;j<2;j++ )
{
        int SystemWideCpuUsage = usageA.GetCpuUsage();
//전체프로세스 CPU 사용량
int ProcessCpuUsageByName = usageB.GetCpuUsage( processname );
//프로세스 CPU 사용량

Sleep( 250 );

if( ProcessCpuUsageByName > 70 ) { // 프로세스 CPU 점유율 70%가 넘었을때
      ProcessID = pe32.th32ProcessID;
      suspend = DebugActiveProcess(ProcessID);
        //프로세스 일시중지

     printf("₩a"); printf("₩a"); printf("₩a");      //경고음
      MessageBoxW( NULL, L"CPU점유율이 최대 70%가 넘는 프로세스가 발견되었습니다.",
     L"Warning", 0 ); //경고메세지
     system("cls");

_tprintf(TEXT("₩n₩n₩n₩n₩n₩n================================
      ================" ) );
_tprintf( TEXT("₩n                    PROCESS NAME :  %s"), pe32.szExeFile );
_tprintf( TEXT("₩n                    Process ID    :  %u"), pe32.th32ProcessID );


if( j==1 ) printf("₩n                    전체 프로세스 CPU 사용량 :  %d", SystemWideCpuUsage);
if( j==1 ) printf("₩n                  프로세스 CPU 사용량       :  %d", ProcessCpuUsageByName);
_tprintf(TEXT("₩n₩n₩n₩n₩n₩n================================
      ================" ) );

        printf("₩n                    프로세스를 중지하였습니다.");
        printf("₩n                    프로세스를 종료하시려면 1");
        printf("₩n                    다시 되돌리시려면 2를 입력해주세요.");
        scanf("%d",&suspend);

              if( suspend == 1 ) {
                    KillProcess(processname,ProcessID);
//프로세스kill함수 호출
                    printf("                    프로세스를 종료하였습니다.₩n");
                    Sleep( 1000 );
                    printf("                    모니터링을 다시 시작합니다.");
                    Sleep( 1000 );
```

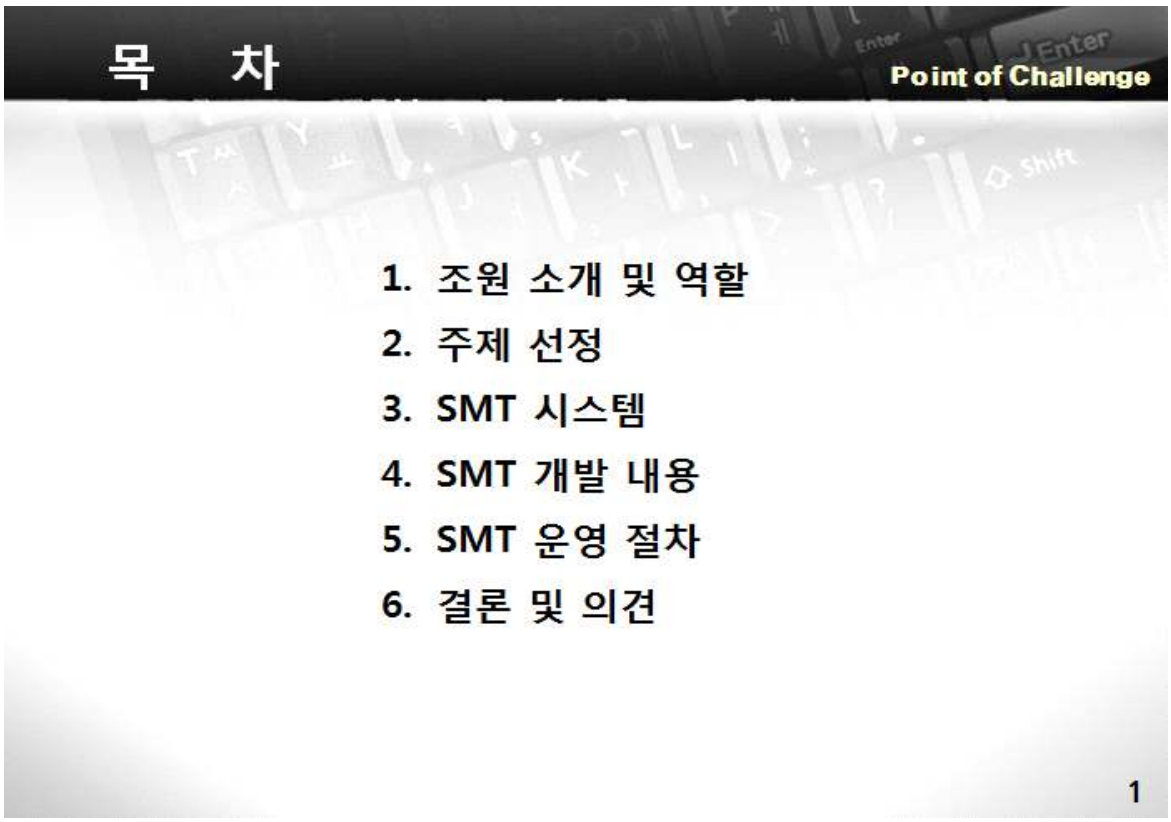
```
                    }

 if( suspend == 2 ) {
                    DebugActiveProcessStop(ProcessID);
//프로세스 일시중지 해제
                    printf("                    프로세스를 원래대로 되돌렸습니다.₩n");
                        Sleep( 1000 );
                        printf("                    모니터링을 다시 시작합니다.");
                        Sleep( 1000 );
                    }
            }
 }

 if( GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc)) ) {
 // 프로세스 메모리정보 구하는 함수
 if( (pmc.WorkingSetSize/DIV) > ((statex.ullAvailPhys/DIV)*0.75) ) {
 //메모리 점유율 70%가 넘었을때
            ProcessID = pe32.th32ProcessID;
            suspend = DebugActiveProcess(ProcessID);
                                        //프로세스 일시중지
            printf("₩a"); printf("₩a"); printf("₩a");
                                    //경고음
 MessageBoxW( NULL, L"메모리점유율이 최대 70%가 넘는 프로세스가 발견되었습니다.",
        L"Warning", 0 );//경고메세지
                    system("cls");
_tprintf(TEXT("₩n₩n₩n₩n₩n₩n₩n================================
        ================" ) );

 _tprintf( TEXT("₩n                    PROCESS NAME :  %s"), pe32.szExeFile );
 _tprintf( TEXT("₩n                    Process ID   :  %u"), pe32.th32ProcessID );
 printf("₩n      최대작업메모리의 크기 : %*d KB₩n", WIDTH, statex.ullTotalPhys/DIV);
 printf("₩n      프로세스의 작업메모리크기 : %d KB₩n", pmc.WorkingSetSize/DIV);
 _tprintf(TEXT("₩n₩n₩n₩n₩n₩n₩n================================
        ================" ) );

            printf("₩n                    프로세스를 중지하였습니다.");
            printf("₩n                    프로세스를 종료하시려면 1");
            printf("₩n                    다시 되돌리시려면 2를 입력해주세요.");
            scanf("%d",&suspend);
            if (suspend == 1) {
                    KillProcess( processname, ProcessID );
                    //프로세스kill함수 호출
                    printf("                    프로세스를 종료하였습니다.₩n");
                    Sleep( 1000 );
                    printf("                    모니터링을 다시 시작합니다.");
```

```
                                        Sleep( 1000 );
                                        }

                                        if (suspend == 2) {
                                                DebugActiveProcessStop( ProcessID );
                        //프로세스 일시중지 해제
                                                printf("                프로세스를 원래대로 되돌렸습니다.₩n");
                                                Sleep( 1000 );
                                                printf("                        모니터링을 다시 시작합니다.");
                                                Sleep( 1000 );
                                        }
                                }
                        }

                        CloseHandle(hProcess);                  // 프로세스의 핸들을 반환해주는 함수
                        i++;
                        system("cls");
                } while( Process32Next( hProcessSnap, &pe32 ) ) //엔트리 프로세스를 읽고 없을때까지 계속 읽음
                CloseHandle( hProcessSnap );            //스냅샷의 핸들을 반환해주는 함수
                return 0;
}

BOOL KillProcess(char * processname,int ProcessID) {    //프로세스kill 함수 선언

        HANDLE          hProcessSnap = NULL;
        DWORD           Return       = FALSE;
        PROCESSENTRY32  pe32         = {0};
        int result;
        char *exe = ".exe";
        strcat(processname,exe);
        char *Temp =  pe32.szExeFile;
        hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

        if(hProcessSnap == INVALID_HANDLE_VALUE) return (DWORD)INVALID_HANDLE_VALUE;
        pe32.dwSize = sizeof(PROCESSENTRY32);

        if(Process32First(hProcessSnap, &pe32)) {
                DWORD Code = 0;
                DWORD dwPriorityClass;

                do {
                        HANDLE hProcess;
                        hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pe32.th32ProcessID);
                        dwPriorityClass = GetPriorityClass(hProcess);
```

```
                        result=strcmp(Temp, processname);    //kill할 프로세스를 찾음

                        if(result==0) {                      //찾으면 kill
                                if(TerminateProcess(hProcess, 0))
                                        GetExitCodeProcess(hProcess, &Code);
                                else
                                        return GetLastError();
                        }
                        CloseHandle(hProcess);
                }

                while(Process32Next(hProcessSnap, &pe32));
                Return = TRUE;
        }
        else
                Return = FALSE;
        DebugActiveProcessStop(ProcessID);          //프로세스 일시중지 해제
        CloseHandle(hProcessSnap);
        return Return;
}



void printError( TCHAR* msg ) {             //Error 출력 함수 선언
        DWORD eNum;
        TCHAR sysMsg[256];
        TCHAR* p;

        eNum = GetLastError( );
        FormatMessage( FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
                                        NULL, eNum,
                                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                                        // Default language
                                        sysMsg, 256, NULL );
        p = sysMsg;
        while( ( *p > 31 ) || ( *p == 9 ) ) ++p;

        do { *p-- = 0; }
        while( ( p >= sysMsg ) && ( ( *p == '.' ) || ( *p < 33 ) ) );
        _tprintf( TEXT("₩n  WARNING: %s failed with error %d (%s)"), msg, eNum, sysMsg );
          //Error부분 출력
}
```

4-2. 발표 PPT 자료


시스템자원 모니터링 툴
(SMT)
지도교수 : 양정모 교수님
2 조 P o C
(Point of Challenge)


목 차
Point of Challenge
1. 조원 소개 및 역할
2. 주제 선정
3. SMT 시스템
4. SMT 개발 내용
5. SMT 운영 절차
6. 결론 및 의견
1

조원 소개 및 역할

Point of Challenge

한소라
조장
총 괄
CPU 자료조사
C++ 프로그래밍

김은지
조원
CPU 자료조사
C++ 프로그래밍

Point of Challenge

이재익
조원
MEMORY 자료조사
C++ 프로그래밍

한우영
조원
MEMORY 자료조사
C++ 프로그래밍

2

주제 선정

Point of Challenge

❖ 주 제

- CPU, 메모리 등 시스템자원의 운영상태를 실시간 감시
- 안정적인 시스템 운영을 지원하는 모니터링 툴 개발

⇨ 개발 시스템 : SMT(System Monitoring Tool) 시스템

선정 이유

✓ 각종 악성코드는 시스템 자원을 고갈시키는 등의 방법으로 시스템의 운영을 방해하거나 정보자료를 훼손

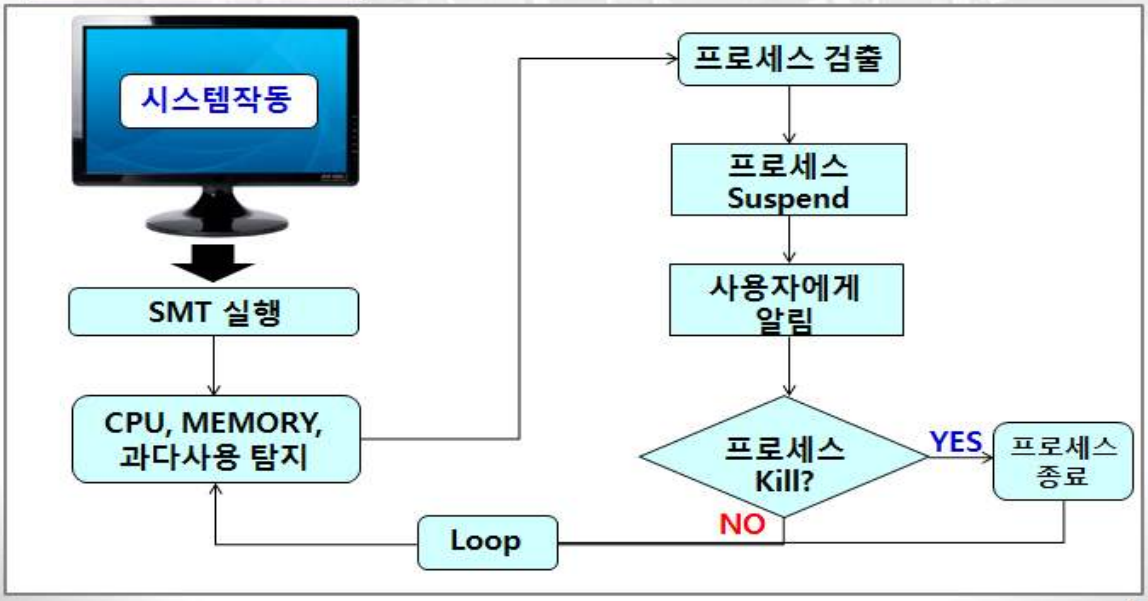
✓ 이에 대응, 시스템 자원의 과부하 등 운영상태를 실시간 감시, 각종 침해를 실시간 통제함으로써 피해를 최소화하고 시스템의 안정적인 운영을 도모

3

SMT 시스템

Point of Challenge

시스템 계통도



4

SMT 시스템

Point of Challenge

개발 환경

❖ 운영체제

- OS : Windows7

❖개발 환경

- 사용언어 : C/C++

- 개발도구 : Microsoft Visual Studio 2008/2010

5

SMT 시스템

Point of Challenge

탐지·경보 및 조치

❖ **메모리**

- 프로세스별 메모리 사용량이 70% 이상인 경우 탐지
- 해당 프로세스를 자동 Suspend하고 탐지상황을 경보

❖ **CPU**

- 프로세스별 CPU 사용량이 70% 이상인 경우 탐지
- 해당 프로세스를 자동 Suspend하고 탐지상황을 경보

6

SMT 개발 내용

Point of Challenge

메모리 사용량 검사

❒ GlobalMemoryStatusEx 함수로 메모리 사용용량을 검사, 과부하 여부 평가

```
do
{
  hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
                          PROCESS_VM_READ,
                          FALSE, pe32.th32ProcessID  );

  if(GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc))) {
  if((pmc.WorkingSetSize/DIV) > ((statex.ullAvailPhys/DIV)/2)) {

      printf("₩a");
      printf("₩a");
      printf("₩a"); //경고음
       MessageBoxW(NULL, L"메모리점유율이 최대 50%가 넘는프로세스가 발견되었
      _tprintf( TEXT("₩n₩n=============================================
      _tprintf( TEXT("₩nPROCESS NAME:  %s"), pe32.szExeFile );
      _tprintf( TEXT("₩n-------------------------------------------------
      _tprintf( TEXT("₩n  Process ID        = %u"), pe32.th32ProcessID );
      //  printf("%u 의 메모리 점유율이 최대메모리크기의 50%를 넘었음.", pro
          printf("₩n최대작업메모리의 크기 : %d KB₩n", WIDTH, statex.ullTot
          printf("₩n프로세스의 작업메모리크기 : %d KB₩n", pmc.WorkingSetSiz
      }
  }
  CloseHandle(hProcess);
} while( Process32Next( hProcessSnap, &pe32 ) );
CloseHandle( hProcessSnap );
return( TRUE );
```

메모리 사용용량을 계산하는 프로그램

프로세스별 메모리 소요량

```
PROCESS NAME:
-------------------------------------------------
  Process ID        = 4012
최대작업메모리의 크기 : 1833456 KB

프로세스의 작업메모리크기 : 126320 KB


=================================================
PROCESS NAME:  chrome.exe
-------------------------------------------------
  Process ID        = 968
최대작업메모리의 크기 : 1833456 KB

프로세스의 작업메모리크기 : 124288 KB


=================================================
PROCESS NAME:  devenv.exe
-------------------------------------------------
  Process ID        = 2616
최대작업메모리의 크기 : 1833456 KB

프로세스의 작업메모리크기 : 156324 KB
계속하려면 아무 키나 누르십시오 . . .
```

7

SMT 개발 내용

Point of Challenge

CPU 사용량 검사

❒ GetCpuUsage 함수를 이용하여 프로세스별 CPU 사용용량을 측정, 과부하 여부 평가

```
printf("SystemWide Cpu Usage     Explorer cpu usage    Cpu Usage
printf("********************     ******************    *********

do
{
  hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
                          PROCESS_VM_READ,
                          FALSE, pe32.th32ProcessID  );

  processeslist[i]=pe32.szExeFile;
  strtok(processeslist[i],".");
  printf("%d = %s\n",i, processeslist[i]);

  int processID=0;
  CCpuUsage usageA;
  CCpuUsage usageB;
  CCpuUsage usageC;

  for(j=0;j<2;j++)
  {
      // Display the system-wide cpu usage and the "Explore

      int SystemWideCpuUsage = usageA.GetCpuUsage();
      int ProcessCpuUsageByName = usageB.GetCpuUsage(proces
      int ProcessCpuUsageByID = usageC.GetCpuUsage(processID);
      if(j==1)printf("%19d%%%22d%%%31d%%%\n",SystemWideCpuUsage, ProcessCpuUsageByName, ProcessCpuUsageByID);
      Sleep(100);
  }
```

프로세스 CPU 사용량 측정용 프로그램

프로세스별 CPU 사용량

```
SystemWide Cpu Usage     process cpu usage     Cpu Usage for processID 0
********************     *****************     *************************
0 = [System Process]
                  36%                  0%                        259%
1 = System
                  30%                  0%                        311%
2 = smss
                  28%                  0%                        288%
3 = csrss
                  27%                  0%                        317%
4 = csrss
                  46%                  0%                        299%
5 = wininit
                   0%                  0%                        308%
```

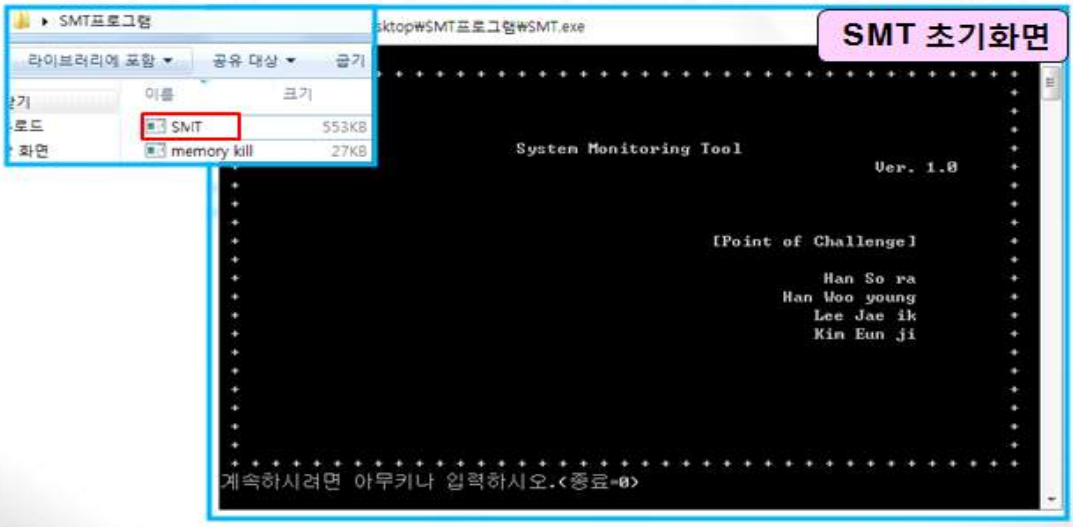
8

SMT 운영 절차

Point of Challenge

SMT 시스템 구동

❒ SMT 실행



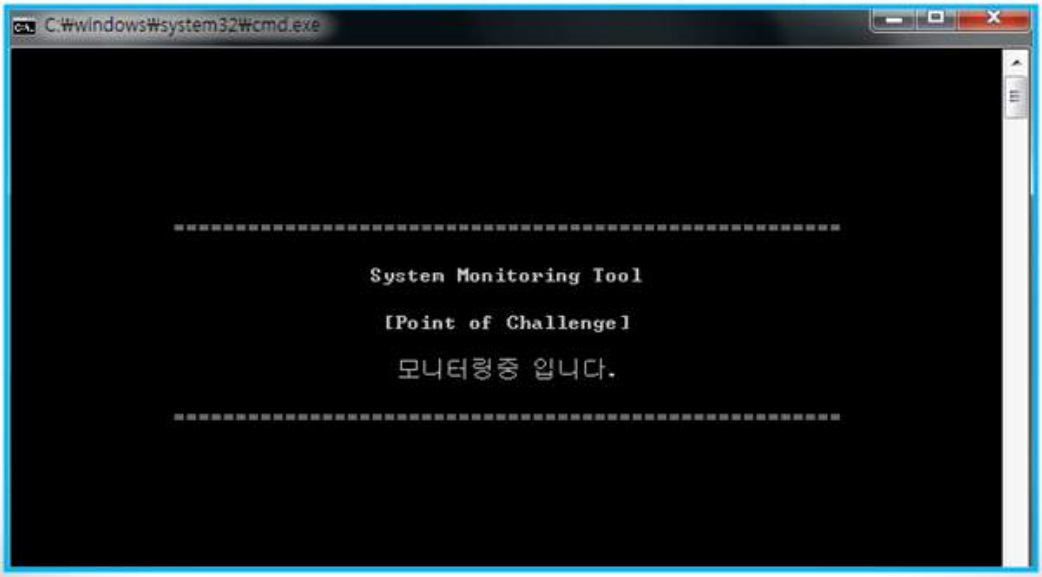
9

SMT 운영 절차

Point of Challenge

SMT 시스템 구동

❒ SMT 구동



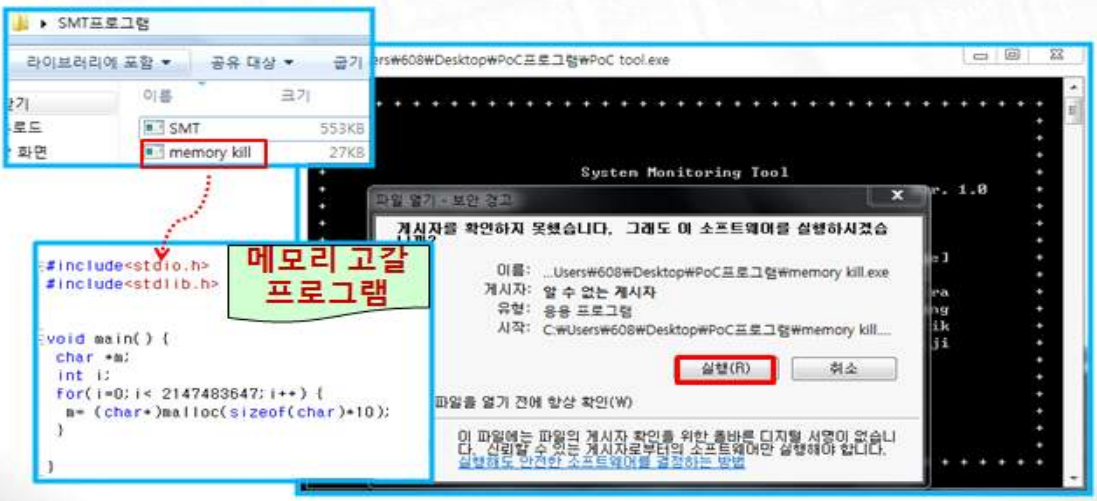
10

SMT 운영 절차

Point of Challenge

메모리 고갈 프로그램(memory kill.exe) 구동

❒ memory kill.exe을 실행



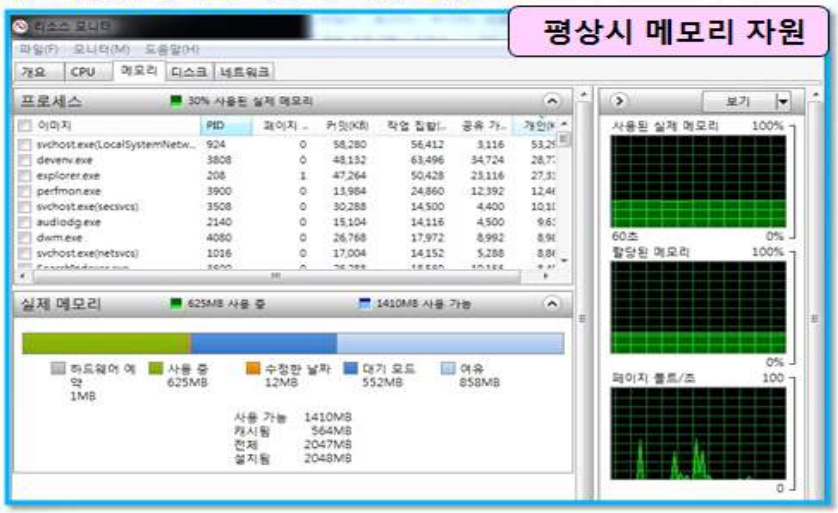
11

SMT 운영 절차

Point of Challenge

메모리 과부하 탐지

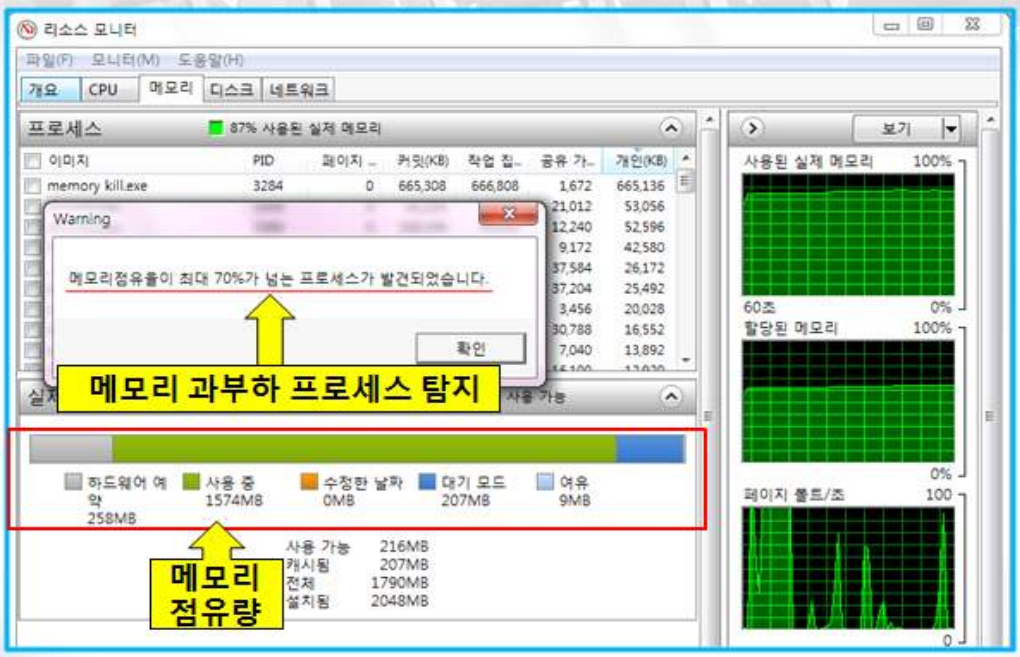
❒ 윈도우 작업관리자로 메모리 자원 확인



12

SMT 운영 절차

Point of Challenge



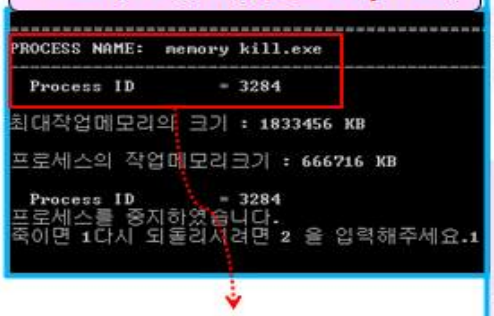
13

SMT 운영 절차

Point of Challenge

❒ 검출한 프로세스 상세정보 출력 ⇨ 사용자에게 프로세스 Kill 여부 질의

프로세스 정보(자동 Suspend)



프로세스 이름 : memory kill.exe
프로세스 ID : 3284

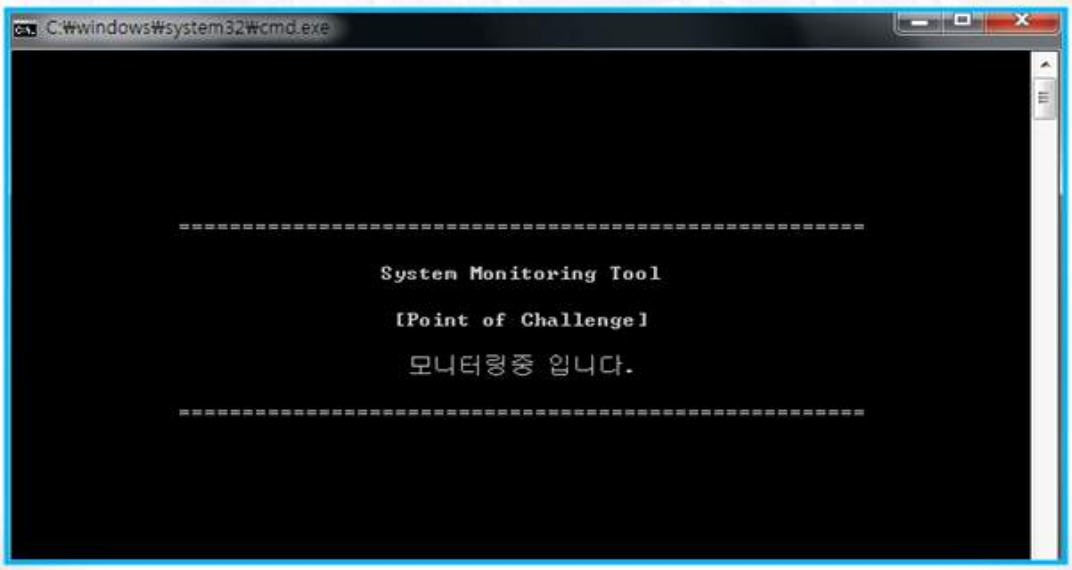
프로세스 Kill

14

SMT 운영 절차

Point of Challenge

프로세스 종료 후 재구동



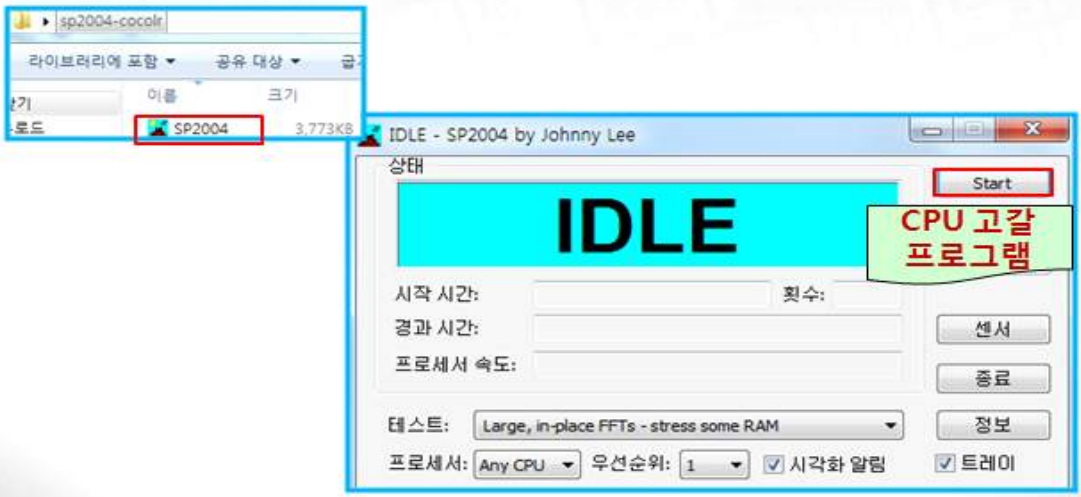
15

SMT 운영 절차

Point of Challenge

CPU 고갈 프로그램(SP2004.exe) 구동

❒ SP2004.exe을 실행



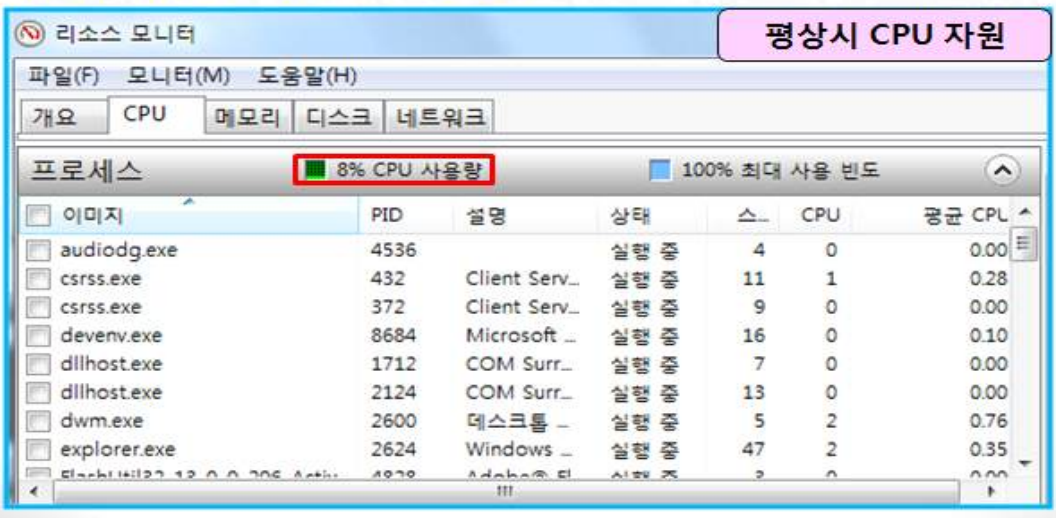
16

SMT 운영 절차

Point of Challenge

CPU 과부하 탐지

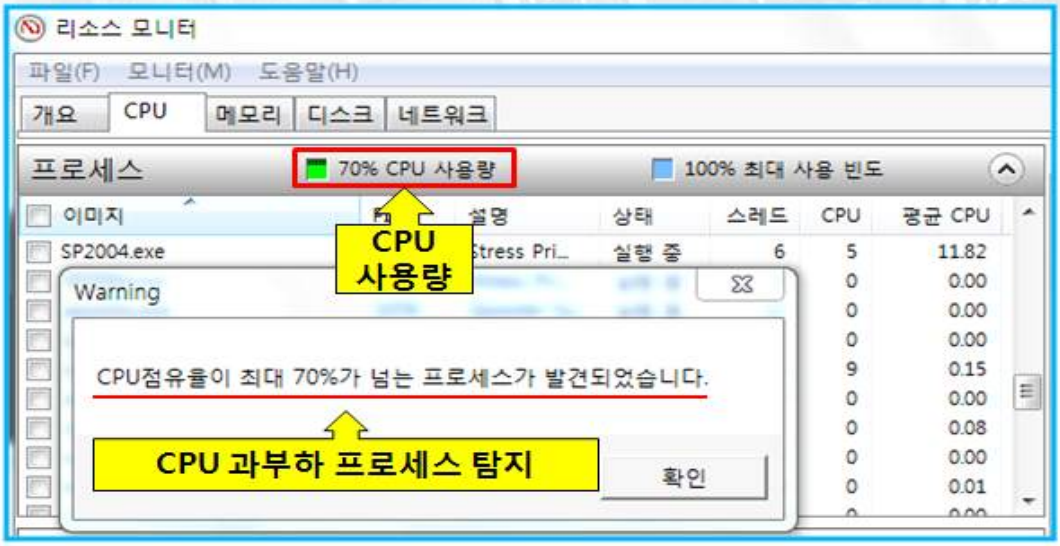
❒ 윈도우 작업관리자로 CPU 자원 확인



17

SMT 운영 절차

Point of Challenge

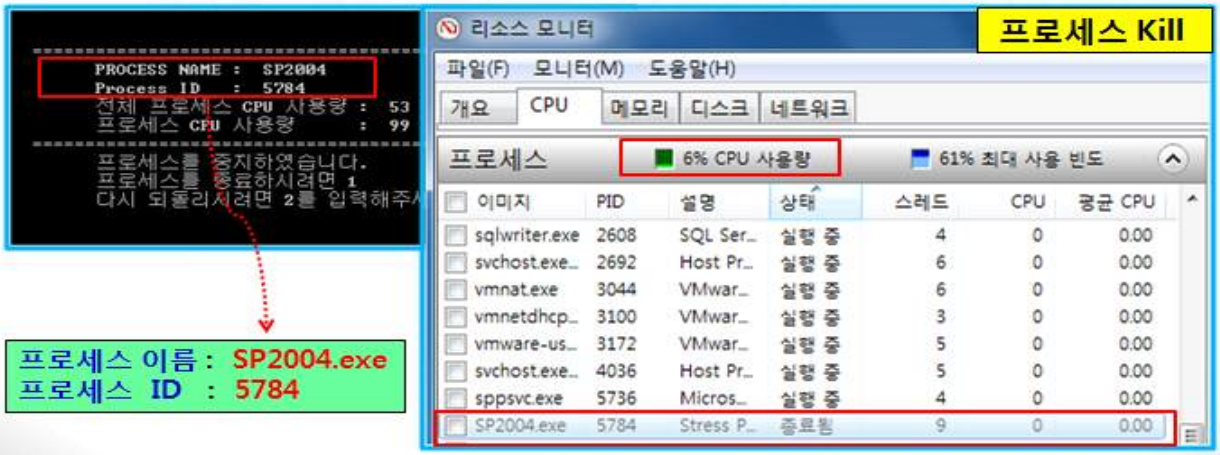


SMT 운영 절차

Point of Challenge

❒ 검출한 프로세스 상세정보 출력 ⇨ 사용자에게 프로세스 Kill 여부 질의

프로세스 정보(자동 Suspend)



결론 및 의견

Point of Challenge

❖ SMT 프로그램 개발 성과

① 시스템 자원의 과부하 등 운영상태를 실시간 감시

② 각종 침해를 실시간 통제 가능하게 함으로써 피해를 최소화하고 시스템의 안정적인 운영을 도모

❖ SMT 프로그램 개발과정에서 시스템 자원 모니터링 툴 등을 직접 프로그램으로 구현, 기술개발 역량을 배양

20

Point of Challenge

Q & A

Thank you!

4-3. 참고자료

http://msdn.microsoft.com/en-us/library/windows/desktop/aa366589(v=vs.85).aspx (MSDN 코리아)

http://ospace.tistory.com/195 (JaPa2 - 배우면 배울 수 록 더 배울 것이 생긴다)

http://blog.daum.net/01193780006/9747158 (Daum blog)