1. 코드 내용 요약 (코드의 길이가 길어 간략히 설명)

..... 필요한 모듈 import

..... 필요함수 선언

1. 2^bit == size 확인 함수 // mustbepowerof2()
2. num2 = num1 * 상수 확인 함수 // mustbemultipleof()
3. 크기 변환 함수 // convert()

..... 메인함수

1. 옵션 부여 부분
   - optionparser사용 // 디버깅시 옵션 부여 안하면 디폴트 값으로 동작

2. 부여된 옵션에 따른 변수 값들 출력
   - 출력 변수 : seed, address space size, phys mem size, page size, verbose, addresses

3. 입력된 pagesize, physical memory size, address space size들 변환
   - convert함수 사용

4. address space size와 physicial memory size가 page size의 배수인지 확인
   - mustbemultipleof 함수 사용

5. virtual memory의 page table과 physical memory의 page table 생성

6. virtual address의 비트부분 선언
   - virtual page num bit의 크기 및 offset 크기 결정

7. 페이지 테이블 출력 부분

8. virtual 페이지 테이블 출력 부분

2. Option Parse 분석

1) option -A : address 를 나타냄

액세스할 쉼표로 구분된 페이지 집합으로 -1의 값을 가질 경우 랜덤으로 생성

2) option -s : seed값을 나타냄

위의 option -A가 -1 일 경우 addresses을 랜덤으로 생성해야할 때 사용할 seed값

3) option -a : address space size를 나타냄

주소공간의 크기를 나타낸다. ex) 32일 경우 주소비트는 5비트

4) option -p : physical memory size를 나타냄

실제 저장공간의 총 크기를 나타낸다.

5) option -P : page size를 나타냄

page의 크기를 나타냄 이때 page의 크기는 physical memory size의 약수 이어야함

6) option -n : number of virtual addresses to generate

7) option -u : 사용할 가상주소공간의 크기의 비율

Default = 50%

8) option -v : verbose mode

VPN 을 출력하고 싶으면 verbose 모드를 사용한다.

9) option -c : compute answers for me

3. 과제 내용

1. Now let's try some different random seeds, and some different (and sometimes quite crazy) address-space parameters, for variety:

-P 8 -a 32 -p 1024 -v -s 1

-P 8k -a 32k -p 1m -v -s 2

-P 1m -a 256m -p 512m -v -s 3

이때 seed 옵션 값이 바뀌는 이유는 옵션에 따라 시드값을 다르게 주어서 사용되는 주소값들을 다르게 하기 위해서이다 또한 v 옵션은 VPN을 출력하기 위해 모든 옵션에 부여한 옵션 값이다.

그러므로 문제 상황에서 우리가 조작할 옵션은 P, a, p 총 3가지로써

Pagesize, address space size, physical memory size,값들의 변화에 따라 출력되는 주소값들을 분석하면 된다.

Option 1.

실행결과

```
Page Table (from entry 0 down to the max size)
  [       0]
0x00000000
  [       1]
0x80000061
  [       2]
0x00000000
  [       3]
0x00000000

Virtual Address Trace
  VA 0x0000000e (decimal:        14) --> PA or invalid address?
  VA 0x00000014 (decimal:        20) --> PA or invalid address?
  VA 0x00000019 (decimal:        25) --> PA or invalid address?
  VA 0x00000003 (decimal:         3) --> PA or invalid address?
  VA 0x00000000 (decimal:         0) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).

>>> # 201600282 엄기산
>>> #-P 8 -a 32 -p 1024 -v -s 1
```

유도과정

Address space = 32 -> 주소비트는 5비트 // page 수 = 4 -> VPN 비트는 2비트

1) 0x0000000e

- 이진수 변환 = 0 1110 -> VPN = 01 , offset = 110
- page table의 index 01의 맨 앞 비트가 1 이므로 valid (값 : 0x61 = 0110 0001)
- table에 있는 값을 offset 크기만큼 왼쪽으로 shift (11 0000 1000 = 0x308)
- offset과 or연산 실행
  (11 0000 1000 OR 00 0000 0110 = 11 0000 1110 = 0x30e)
- 결과 = **0x30e**

2) 0x00000014

- 이진수 변환 = 1 0100 -> VPN = 10 , offset = 100
- page table의 index 01의 맨 앞 비트가 0 이므로 **not valid**

3) 0x00000019

- 이진수 변환 = 1 1001 -> VPN = 11 , offset = 001
- page table의 index 11의 맨 앞 비트가 0 이므로 **not valid**

4) 0x00000003

- 이진수 변환 = 0 0011 -> VPN = 00 , offset = 011
- page table의 index 00의 맨 앞 비트가 0 이므로 **not valid**

5) 0x00000000

- 이진수 변환 = 0 0000 -> VPN = 00 , offset = 000
- page table의 index 00의 맨 앞 비트가 0 이므로 **not valid**

Option 2.

실행결과

```
Page Table (from entry 0 down to the max size)
  [       0]
0x80000079
  [       1]
0x00000000
  [       2]
0x00000000
  [       3]
0x8000005e

Virtual Address Trace
  VA 0x000055b9 (decimal:    21945) --> PA or invalid address?
  VA 0x00002771 (decimal:    10097) --> PA or invalid address?
  VA 0x00004d8f (decimal:    19855) --> PA or invalid address?
  VA 0x00004dab (decimal:    19883) --> PA or invalid address?
  VA 0x00004a64 (decimal:    19044) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).

>>> #-P 8k -a 32k -p 1m -v -s 2
>>> #201600282 엄기산
```

유도과정

Address space = 32k -> 주소비트는 15비트 // page 수 = 4 -> VPN 비트는 2비트

1) 0x000055b9

- 이진수 변환 = 101 0101 1011 1001 -> VPN = 10 offeet = 1 0101 1011 1001
- page table의 index 10의 맨 앞 비트가 0 이므로 **not valid**

2) 0x00002771

- 이진수 변환 = 010 0111 0111 0001 -> VPN = 01 offeet = 0 0111 0111 0001
- page table의 index 01의 맨 앞 비트가 0 이므로 **not valid**

3) 0x00004d8f

- 이진수 변환 = 100 1101 1000 1111 -> VPN = 10 offset = 0 1101 1000 1111
- page table의 index 10의 맨 앞 비트가 0 이므로 **not valid**

4) 0x00004dab

- 이진수 변환 = 100 1101 1010 1011 -> VPN = 10 offset = 0 1101 1010 1011
- page table의 index 10의 맨 앞 비트가 0 이므로 **not valid**

5) 0x00004a64

- 이진수 변환 = 100 1010 0110 0100 -> VPN = 10 offset = 0 1010 0110 0100
- page table의 index 10의 맨 앞 비트가 0 이므로 **not valid**

Option 3.

실행결과

```
0x00000000
  [     253]
0x00000000
  [     254]
0x80000159
  [     255]
0x00000000

Virtual Address Trace
  VA 0x0308b24d (decimal: 50901581) --> PA or invalid address?
  VA 0x042351e6 (decimal: 69423590) --> PA or invalid address?
  VA 0x02feb67b (decimal: 50247291) --> PA or invalid address?
  VA 0x0b46977d (decimal: 189175677) --> PA or invalid address?
  VA 0x0dbcceb4 (decimal: 230477492) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).

>>> # -P 1m -a 256m -p 512m -v -s 3
>>> #201600282 엄기산
>>>
```

유도과정

Address space = 256m -> 주소비트는 28비트

page 수 = 256 -> VPN 비트는 8비트

1) 0x0308b24d

- 이진수 변환 = 0011 0000 1000 1011 0010 0010 1101
  -> VPN = 0011 0000 offset = 1000 1011 0010 0010 1101
- page table의 index 0011 0000 (= 48) 의 맨 앞 비트가 1 이므로 valid
  -> 0x800001f6 = 1 1111 0110
- table에 있는 값을 offset 크기만큼 왼쪽으로 shift (총 20회)
  -> 1 1111 0110 0000 0000 0000 0000 0000
- 구한 값과 offset OR 연산

  1 1111 0110 0000 0000 0000 0000 0000
  OR 0 0000 0000 1000 1011 0010 0010 1101
  = 1 1111 0110 1000 1011 0010 0010 1101 -> **0x1f68b24d**

2) 0x042351e6

- 이진수 변환 = 0100 0010 0011 0101 0001 1110 0110
  -> VPN = 0100 0010 offset = 0011 0101 0001 1110 0110
- page table의 index 0100 0010 (= 66) 의 맨 앞 비트가 0 이므로 **not valid**

3) 0x02feb67b

- 이진수 변환 = 0010 1111 1110 1011 0110 0111 1011
  -> VPN = 0010 1111 offset = 1110 1011 0110 0111 1011

- page table의 index 0010 1111 (= 47) 의 맨 앞 비트가 1 이므로 valid
  -> 0x800000a9 = 0 1010 1001
- table에 있는 값을 offset 크기만큼 왼쪽으로 shift (총 20회)
  -> 0 1010 1001 0000 0000 0000 0000 0000
- 구한 값과 offset OR 연산
  0 1010 1001 0000 0000 0000 0000 0000
  OR 0 0000 0000 1110 1011 0110 0111 1011
  = 0 1010 1001 1110 1011 0110 0111 1011 -> **0x0a9eb67b**

4) 0x0b46977d

- 이진수 변환 = 1011 0100 0110 1001 0111 0111 1101
  -> VPN = 1011 0100 offset = 0110 1001 0111 0111 1101
- page table의 index 1011 0100 (= 180) 의 맨 앞 비트가 0 이므로 **not valid**

5) 0x0dbcceb4

- 이진수 변환 = 1101 1011 1100 1100 1110 1011 0100
  -> VPN = 1101 1011 offset = 1100 1100 1110 1011 0100
- page table의 index 1101 1011 (= 219) 의 맨 앞 비트가 1 이므로 valid
  -> 0x800001f2 = 1 1111 0010
- table에 있는 값을 offset 크기만큼 왼쪽으로 shift (총 20회)
  -> 1 1111 0010 0000 0000 0000 0000 0000
- 구한 값과 offset OR 연산
  1 1111 0010 0000 0000 0000 0000 0000
  OR 0 0000 0000 1100 1100 1110 1011 0100
  = 1 1111 0010 1100 1100 1110 1011 0100 -> **0x1f2cceb4**

Q2. Which of these parameter combinations are unrealistic? Why?

- 1번과 3번의 옵션이 현실성이 없다.

	Option 1	Option 2	Option 3
Page 갯수	4	4	256
Physical memory	1024	1m	512m
총Page 크기	32	32k	256m
저장 가능 프로세스 수	32	32	2

- 1번의 경우에는 page table의 크기가 너무 작아 프로세스 하나가 사용할수 있는 memory의 크기가 매우 작아진다. 또한 3번의 경우에는 page table의 크기가 너무 커서 2개의 프로세스까지만 page table을 만들 수 있다.