OS Project #2 - 3 Level Paging Report

201811061 박동진

- Implementation Description

1. walkpgdir

```
78  static pte_t *
79  walkpgdir(pde_t *pgdir, const void *va, int alloc)
80  {
81    pde_t *pgdir2;
82    pde_t *pd1e;
83    pde_t *pd2e;
84    pte_t *pgtab;
85
86    // TODO: Modify for 3-level paging
87    pd1e = &pgdir[PD1X(va)];
88    if(*pd1e & PTE_P){  // pd1e have present bit
89      pgdir2 = (pde_t*)P2V(PTE_ADDR(*pd1e));
90      pd2e = &pgdir2[PD2X(va)];
91      if(*pd2e & PTE_P){  // pd2e have present bit
92        pgtab = (pte_t*)P2V(PTE_ADDR(*pd2e));
93      } else { // pd2e is not present -> allocation
94        if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
95          return 0;
96        memset(pgtab, 0, PGSIZE);
97        *pd2e = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
98      }
99    } else { // pd1e is not present -> allocation
100     if(!alloc || (pgdir2 = (pde_t*)kalloc()) == 0)
101       return 0;
102     memset(pgdir2, 0, PGSIZE);
103     *pd1e = V2P(pgdir2) | PTE_P | PTE_W | PTE_U;
104     pd2e = &pgdir2[PD2X(va)];
105     if(!alloc || (pgtab = (pte_t*)kalloc()) == 0) //
106       return 0;
107     memset(pgtab, 0, PGSIZE);
108     *pd2e = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
109   }
110   pgtab[PTNX(va) + 512] += 1; // count page access i
111   return &pgtab[PTNX(va)];
112 }
```

(87)에서 first page directory의 index에 해당하는 Page directory entry를 가리키는 주소값을 pd1e에 저장합니다. *pd1e는 실제 first page directory entry의 value를 의미합니다. PTE_P값이 존재하는지 확인(valid를 체크)하고 존재한다면, *pd1e의 PTE_ADDR를 pgdir2에 저장합니다. 동일한 방식으로 pd2e를 저장하고 PTE_P값이 존재하는지 확인합니다. 존재한다면 *pd2e의 PTE_ADDR을 pgtab에 저장합니다. (93)만일 존재하지 않는다면, walkpgdir의 parameter인 alloc값이 0이 아니라면(allocation을 하지 않을 때의 walkpgdir 호출), kalloc()을 통해 allocation을 해줍니다. 그리고 valid한 pd1e가 없다면(99), pgdir2를 allocate 해주고, 곧바로 pgtab도 allocate 합니다. (110)에서 pgtab의 남은 공간에 count를 더하여 줍니다. 이때, page table index에 512를 더하여 줍니다.

```
371 int
372 deallocuvm(pde_t *pgdir, uint oldsz, uint newsz)
373 {
374   // TODO: Modify for 3-level paging
375   pde_t *pd2e;
376   pte_t *pte;
377   uint a, pa;
378
379   if(newsz >= oldsz)
380     return oldsz;
381   a = PGROUNDUP(newsz);
382   for(; a  < oldsz; a += PGSIZE){ // traverse
383     pd2e = walkpgdir1(pgdir, (char*)a);
384     pte = walkpgdir2(pd2e, (char*)a);  // no allocation, just search
385     if(!pd2e){ // there is no present pd2e
386       if(PD1X(a) + 1 > 63){ // prevent pd1 overflow
387         break;
388       }
389       // traverse next pd1e
390       a = PG3ADDR(PD1X(a) + 1, 0, 0, 0) - PGSIZE;
391     }
392     else if(!pte){ // there is no present pte
393       if(PD2X(a) + 1 > 31){  // prevent pd2 overflow
394         if(PD1X(a) + 1 > 63){ // prevent pd1 overflow
395           break;
396         } else {
397           // traverse next pd1e
398           a = PG3ADDR(PD1X(a) + 1, 0, 0, 0) - PGSIZE;
399         }
400       }
401       else {
402         // traverse next pd2e
403         a = PG3ADDR(PD1X(a), PD2X(a) + 1, 0, 0) - PGSIZE;
404       }
405     }
406     else if((*pte & PTE_P) != 0){ // there is pte to free
407       pa = PTE_ADDR(*pte);
408       if(pa == 0)
409         panic("kfree");
410       char *v = P2V(pa);
411       kfree(v); // only free page in deallocuvm
412       *pte = 0;
413     }
414   }
415   return newsz;
416 }
```

2. deallocuvm

직접 만든 walkpgdir1 함수로 valid한 pd2e를 찾습니다. 직접 만든 walkpgdir2 함수와 pd2e를 통해 valid한 pte를 찾습니다. walkpgdir1 함수는 valid한 pd2e를 찾지 못하면, 0을 return합니다. walkpgdir2 함수도 valid한 pte를 찾지 못하면 0을 return합니다. (385) valid한 pd2e를 찾지 못하면, first page directory index에 1을 더했을 때, first page directory의 최대 엔트리 개수를 넘지 않는다면, PG3ADDR을 통해 다음 first page directory entry를 탐색합니다. (392) valid한 pte를 찾지 못하면, second page directory index에 1을 더했을 때, second page directory 의 최대 엔트리 개수를 넘고, first page directory index에 1을 더했을 때, first page dirctory의 최대 엔트리 개수를 넘지 않는다면, PG3ADDR을 통해 다음 first page directory entry를 탐색합니다. (401) second page directory index를 더 탐색해야 할때, PG3ADDR을

통해 다음 second page directory 엔트리를 탐색합니다. (406) 만일 valid한 pte를 찾았다면, kfree를 통해 page를 dealloc 해줍니다.

```
// search pgdir1
// if found, return pd2e address
// else, return 0
static pde_t* walkpgdir1(pde_t *pgdir, const void *va){
  pde_t *pgdir2;
  pde_t *pd1e;

  pd1e = &pgdir[PD1X(va)];
  if(*pd1e & PTE_P){
    pgdir2 = (pde_t*)P2V(PTE_ADDR(*pd1e));
  } else {
    return 0;
  }
  return &pgdir2[PD2X(va)];
}

// search pgdir2
// if found, return pte address
// else, return 0
static pte_t* walkpgdir2(pde_t *pd2e, const void *va){
  pte_t *pgtab;

  if(pd2e == 0) // if walkpgdir1 return 0, walkpgdir2 return 0 too.
    return 0;

  if(*pd2e & PTE_P){
    pgtab = (pte_t*)P2V(PTE_ADDR(*pd2e));
  } else {
    return 0;
  }
  return &pgtab[PTNX(va)];
}
  return &pgtab[PTNX(va)];
}
```

3. custom walkpgdir

walkpgdir을 변형한 walkpgdir1, 2 입니다. walkpgdir1은 valid한 pd1e를 찾으면 pd2e의 주소를 return합니다. valid한 pd1e를 찾지 못하면 0을 return 합니다. walkpgdir2는 walkpgdir1에서 찾은 pd2e가 valid한지 판단하여 valid하지 않으면 0, valid하면 pte의 주소를 return합니다. (356)은 walkpgdir1이 0을 return하면 같이 0을 return함으로써 deallocuvm에서의 조건문 분기를 구현하였습니다

```
void
freevm(pde_t *pgdir)
{
  // TODO: Modify for 3-level paging
  pde_t *pgdir2;
  pte_t *pgtab;
  uint i, j, k;

  if(pgdir == 0)
    panic("freevm: no pgdir");
  deallocuvm(pgdir, KERNBASE, 0); // free page, after that, free page table and
  for(i = 0; i < 64; i++){ // pd1 : 64 entries (6 bit)
    if(pgdir[i] & PTE_P){ // if its valid, it should traverse second pgdir
      pgdir2 = (pde_t*)P2V(PTE_ADDR(pgdir[i]));
      for(j = 0; j < 32; j++){ // pd2 : 32 entries (5 bit)
        if(pgdir2[j] & PTE_P){ // if its valid, it should traverse page table
          pgtab = (pte_t*)P2V(PTE_ADDR(pgdir2[j]));
          for(k = 0; k < 512; k++){ // pte : 512 entries (9 bit)
            if(pgtab[k + 512] > 1){ // only print page access count 2 or more
              cprintf("va: 0x%x, pgtab[%d]: %d\n", k*PGSIZE, k, pgtab[k + 512]);
            }
          }
          kfree((char*)pgtab); // free page table
        }
      }
      kfree((char*)pgdir2); // free second page directory
    }
  }
  kfree((char*)pgdir); // free first page director
}
```

4. freevm

우선 deallocuvm을 통해 valid page를 모두 dealloc을 해준 뒤, first page directory를 traverse하면서 valid를 확인합니다. valid하다면, second page directory를 traverse하면서 valid를 확인합니다. valid하다면, page table을 traverse하며 page access count가 2 이상인 page의 va와 count를 출력해줍니다. 출력 후, page table을 dealloc합니다. 이어서 second page directory를 dealloc합니다. 마지막으로 first page directory를 dealloc 합니다.

5. printvm : freevm과 동일하게 page directory들과 page table을 traverse하며 valid한 pte를 발견하면 page table을 다 돌면서 3 page 구조를 출력해줍니다.

6. pagefault : macro를 이용하여 3 level page table의 flags를 저장해줍니다.

7. __virt_to_phy3 : walkpgdir 함수를 이용하여 pte를 찾고 해당하는 pa를 return 합니다.

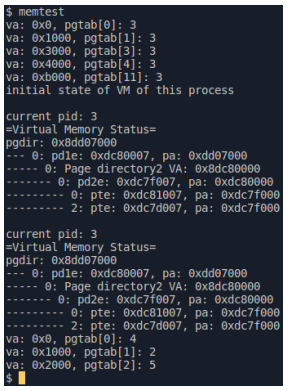
-Test analysis

1. usertests

모든 test 를 통과하여 ALL TESTS PASSED 가 출력되었습니다. 3 level paging 이 성공적으로 구현된 것을 알 수 있습니다.

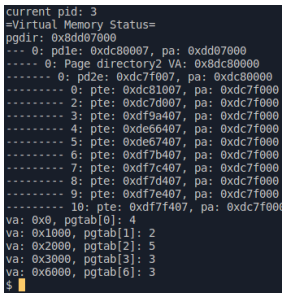
2. memtest

모든 출력이 ppt 의 memtest 의 동일하게 출력됩니다. memtest 는 malloc 을 두 번하는 test 입니다. memtest 의 code 를 한 줄씩 주석 처리하며 page access count 를 분석해보았습니다.

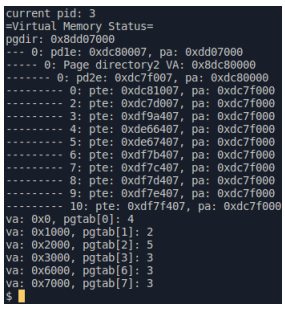


(1)
//int *a = (int*)malloc(4096*4);
//a[2] = buf[10];
//int *b = (int*)malloc(4096*4);
//b[5] = buf[11];
의 결과 입니다.

pgtab[0]이 4 번, pgtab[1]이 2 번, pgtab[2]이 5 번 access 되었습니다.

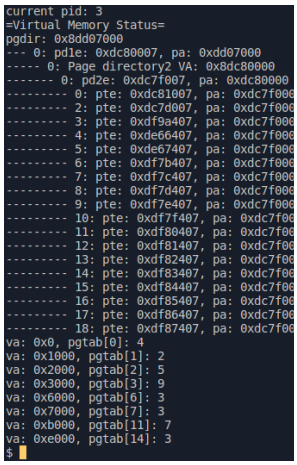


(2)
int *a = (int*)malloc(4096*4);
//a[2] = buf[10];
//int *b = (int*)malloc(4096*4);
//b[5] = buf[11];
의 결과입니다. 16kb를 malloc했을 때, 8개의 pte가 만들어진 모습입니다. 그리고 pgtab[3], [6]이 3번씩 access 되었습니다.

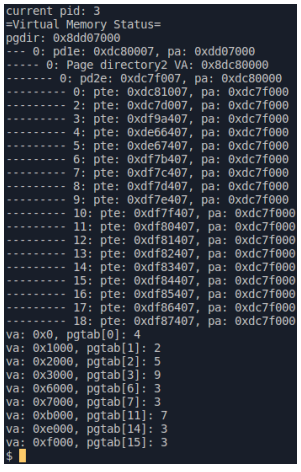


(3)
int *a = (int*)malloc(4096*4);
a[2] = buf[10];
//int *b = (int*)malloc(4096*4);
//b[5] = buf[11];
의 결과입니다.

pgtab[7]이 3 번 access 되었습니다.



(4)
int *a = (int*)malloc(4096*4);
a[2] = buf[10];
int *b = (int*)malloc(4096*4);
//b[5] = buf[11];
의 결과입니다. 16kb를 추가로 malloc했을 때, 총 16개의 pte가 만들어진 모습입니다. 그리고 pgtab[3]의 count가 6증가하였고, pgtab[11]이 7번, pgtab[14]이 3번 access 되었습니다.



(5)
int *a = (int*)malloc(4096*4);
a[2] = buf[10];
int *b = (int*)malloc(4096*4);
b[5] = buf[11];
의 결과입니다.

pgtab[15]가 3 번 access 되었습니다.

-분석

malloc(1)을 해도 pte가 8개가 만들어지는 것을 확인할 수 있었습니다. malloc(4096*7)까지는 그대로 pte가 8개가 생성되었지만, malloc(4096*8)에서는 pte가 9개 생성되고, malloc(4096*9)에서는 pte가 10개 생성되었습니다. malloc이 기본적으로 pte를 8개를 생성하고, 32kB이상의 memory를 할당할 때, pte를 추가로 생성한다는 것을 알 수 있었습니다. 그 이유는 하나의 pte가 하나의 4kB page를 가리키기 때문입니다. malloc이 기본적으로 8개의 pte를 생성함으로써 8개의 4kB page를 할당하는 것입니다. 따라서 32kB이상의 memory를 malloc할 때, pte를 하나 더 생성합니다. 그리고 4kB가 추가될 때 마다 하나의 pte를 더 할당하는 것을 알 수 있습니다. 또한, malloc으로 생성된 포인터

변수에 접근하려 할 때는, malloc으로 생성된 page의 access count가 증가하였습니다. 그리고 malloc으로 새로운 포인터 변수에 memory를 할당하면, 새로운 pte가 8개가 생성 되었습니다.

3. usertests 와 memtest 의 결과 이미지

-usertests

```
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
va: 0x2000, pgtab[2]: 3
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
fork test OK
bigdir test
bigdir ok
uio test
pid 551 usertests: trap 13 err 0 on cpu 0 eip 0x3563 addr 0xcf9c--kill proc
va: 0x3000, pgtab[3]: 3
va: 0xc000, pgtab[12]: 3
uio test done
exec test
va: 0x0, pgtab[0]: 552
va: 0x1000, pgtab[1]: 552
va: 0x2000, pgtab[2]: 552
va: 0x3000, pgtab[3]: 552
va: 0x4000, pgtab[4]: 552
va: 0x5000, pgtab[5]: 552
va: 0x6000, pgtab[6]: 551
va: 0x7000, pgtab[7]: 551
va: 0x8000, pgtab[8]: 551
va: 0x9000, pgtab[9]: 551
va: 0xa000, pgtab[10]: 551
va: 0xb000, pgtab[11]: 552
va: 0xc000, pgtab[12]: 553
va: 0xd000, pgtab[13]: 5
va: 0xe000, pgtab[14]: 4
va: 0x1ff000, pgtab[511]: 4
ALL TESTS PASSED
va: 0x0, pgtab[0]: 4
va: 0x1000, pgtab[1]: 2
va: 0x2000, pgtab[2]: 8
$
```

(page access count 출력이 너무 많아
전체 이미지를 첨부하지 못했습니다.)

-memtest

```
$ memtest
va: 0x0, pgtab[0]: 3
va: 0x1000, pgtab[1]: 3
va: 0x3000, pgtab[3]: 3
va: 0x4000, pgtab[4]: 3
va: 0xb000, pgtab[11]: 3
initial state of VM of this process

current pid: 3
=Virtual Memory Status=
pgdir: 0x8dd07000
--- 0: pd1e: 0xdc80007, pa: 0xdd07000
----- 0: Page directory2 VA: 0x8dc80000
------- 0: pd2e: 0xdc7f007, pa: 0xdc80000
--------- 0: pte: 0xdc81007, pa: 0xdc7f000
--------- 2: pte: 0xdc7d007, pa: 0xdc7f000

current pid: 3
=Virtual Memory Status=
pgdir: 0x8dd07000
--- 0: pd1e: 0xdc80007, pa: 0xdd07000
----- 0: Page directory2 VA: 0x8dc80000
------- 0: pd2e: 0xdc7f007, pa: 0xdc80000
--------- 0: pte: 0xdc81007, pa: 0xdc7f000
--------- 2: pte: 0xdc7d007, pa: 0xdc7f000
--------- 3: pte: 0xdf9a407, pa: 0xdc7f000
--------- 4: pte: 0xde66407, pa: 0xdc7f000
--------- 5: pte: 0xde67407, pa: 0xdc7f000
--------- 6: pte: 0xdf7b407, pa: 0xdc7f000
--------- 7: pte: 0xdf7c407, pa: 0xdc7f000
--------- 8: pte: 0xdf7d407, pa: 0xdc7f000
--------- 9: pte: 0xdf7e407, pa: 0xdc7f000
--------- 10: pte: 0xdf7f407, pa: 0xdc7f000
--------- 11: pte: 0xdf80407, pa: 0xdc7f000
--------- 12: pte: 0xdf81407, pa: 0xdc7f000
--------- 13: pte: 0xdf82407, pa: 0xdc7f000
--------- 14: pte: 0xdf83407, pa: 0xdc7f000
--------- 15: pte: 0xdf84407, pa: 0xdc7f000
--------- 16: pte: 0xdf85407, pa: 0xdc7f000
--------- 17: pte: 0xdf86407, pa: 0xdc7f000
--------- 18: pte: 0xdf87407, pa: 0xdc7f000
va: 0x0, pgtab[0]: 4
va: 0x1000, pgtab[1]: 2
va: 0x2000, pgtab[2]: 5
va: 0x3000, pgtab[3]: 9
va: 0x6000, pgtab[6]: 3
va: 0x7000, pgtab[7]: 3
va: 0xb000, pgtab[11]: 7
va: 0xe000, pgtab[14]: 3
va: 0xf000, pgtab[15]: 3
$
```

감사합니다.