システムソフトウェア

学籍番号: 20B30790　藤井　一喜

課題１の方針及び実装

- 方針

  `kmem.freelist` はフリーメモリとなっているページの先頭を指しているため、以下の方針で空き容量を算出することができる。

  1. `kmem.freelist` より線形リストの先頭ポインタを取得する
  2. `r = kmem.freelist`のように代入し、`r = r->next`により順に探索する
  3. `r`が NULL になった時点で、これ以上空き容量がないことがわかるため、これまでの空き容量の合計を返す

  線形リストを単純に線形探索しているため、`O(n)`の計算量となる。

- 実装

```
// Return sum of free memory in kernel.
uint64 sys_freemem(void)
{
    uint64 free_memory_size = 0;
    struct run *r;

    acquire(&kmem.lock);
    for(r = kmem.freelist; r; r = r->next){
        free_memory_size += PGSIZE;
    }
    release(&kmem.lock);

    return free_memory_size;
}
```

変更ファイル

```
xv6-riscv/
├── Makefile            <-- freememtest を実行できるようにするため
├── kernel
│   ├── kalloc.c        <-- sys_freemem() を実装
│   ├── syscall.c       <-- sys_freemem() を認識させるため
│   └── syscall.h       <-- sys_freemem() に対応する数字を define するため
└── user
    ├── freememtest.c   <-- sys_freemem() をテストするためのコード
    ├── user.h          <-- freememtest で freemem を呼び出せるようにするため
    └── usys.pl         <-- freememtest で freemem を呼び出せるようにするため
```

- Makefile

  `$U/_freememtest\`を追加した

- `kernel/kalloc.c`

  `sys_freemem()`を実装した。（すでに前述）

- `kernel/syscall.c`

  > `extern uint64 sys_freemem(void);`をシステムコールを扱う関数のプロトタイプとして追加
  >
  > また、`[SYS_freemem] sys_freemem,`を追加し、システムコールの番号と実際の関数を対応させた。

- `kernel/syscall.h`

  `#define SYS_freemem 22`を追加し、システムコールの番号を定義した。(前述の`syscall.c`と対応)

- `user/freememtest.c`

  > `freemem()`の動作をチェックするためのコードを追加

- `user/user.h`

  > `int freemem(void);`を追加し、`freememtest.c`で`freemem()`を呼び出せるようにした。

- `user/usys.pl`

  > `freemem`をシステムコールとして認識させるためのコードを追加

テスト結果

- 結果

```
$ freememtest
133382144
133378048
133378048
133373952
```

```
133378048
133382144
```

- 結果の説明

  `sbrk(1);`により 1B 確保しようとするが、1 ページは 4KB(=4096B)なので、1 ページ分の空き容量が減る。(-4096)

  その後`sbrk(4095);`により、新たに 4095B 分確保しようとするが、先ほど確保した 1 ページ分の空き容量があるため、空き容量は減らない。(-0)

  さらに`sbrk(1);`により、1B 分確保しようとするが、今回はすでに確保されている領域に空きが存在しないため、新たに 1 ページ分だけ確保する必要がある。そのため、空き容量は減る。(-4096)

  `sbrk(-4096);`により、1 ページ分の空き容量が返却される。(+4096) これは、先ほどの`sbrk(1);`と`sbrk(4095);`により確保した領域が解放されたためである。依然として最初に確保した`sbrk(1);`の領域は解放されていないため、すべての領域は解放されない。

  `sbrk(-1);`により、前述した領域が解放されるので、空き容量が増える。(+4096)

環境

Docker コンテナを使用

得られた知見

xv6 におけるメモリ管理の仕組みをもう一度理解することができた。

またテストプログラムを通して、ページ単位でのメモリ管理の仕組みを理解することができた。