判断题

1-10 vvvvx vxvxx

内联汇编函数（from lcgg's report）

1. 内联函数可在编译中直接转换成对应的汇编代码，从而而读取的ebp值为调用用read_ebp时的ebp。具体而言，读取到的ebp即为print_stackframe函数堆栈中的ebp。此时该ebp保存了上一个栈的ebp，ebp+4为print_stackframe的返回地址，ebp+8,12,16,20分别为需要输出的4个参数（实际上参数没有那么多）。若不使用用内联函数，读取到的ebp即为调用用read_ebp()函数时的ebp的值，而不是调用print_stackframe时的ebp的值。
2. 非内联函数read_eip读取的是调用read_eip时的当前eip的值。其原理为，调用read_eip后，由于非内联函数，需要进行压栈，因此在当前堆栈中压入入return address和old ebp，并将ebp指向old ebp。而read_eip将ebp+4赋值给函数临时变量eip，即把调用read_eip()函数时的return address赋值给了这个临时变量eip。由于return address恰好为调用read_eip返回后的下一条指令的地址，即恰好为调用read_eip时候的eip寄存器的值。这个做法利用了堆栈切换得到eip寄存器器的值，是十分巧妙的。

连续内存分配算法

操作	返回值	free list
Z = alloc 1025KB	失败	[(0, 1024)]
A = alloc 256KB	(0, 256)	[(256, 768)]
B = alloc 128KB	(256, 128)	[(384, 640)]
C = alloc 256KB	(384, 256)	[(640, 384)]
D = alloc 128KB	(640, 128)	[(768, 256)]
free B	/	[(256, 128), (768, 256)]
free D	/	[(256, 128), (640, 384)]
E = alloc 512KB	失败	[(256, 128), (640, 384)]
F = alloc 257KB	(640, 257)	[(256, 128), (897, 127)]
G = alloc 64KB	(897, 64)	[(256, 128), (961, 63)]
H = alloc 64KB	(256, 64)	[(320, 64), (961, 63)]
free F	/	[(320, 64), (640, 257), (961, 63)]
I = alloc 300KB	失败	同上

没有足够的剩余连续空间 384KB
外碎片
紧凑 分区对换

页表

```
Virtual Address 0a90:
  --> pde index:0x2  pde contents:(valid 1, pfn 0x1b)
    --> pte index:0x14  pte contents:(valid 0, pfn 0x7f)
      --> Fault (page table entry not valid)
Virtual Address 7913:
  --> pde index:0x1e  pde contents:(valid 1, pfn 0x5a)
    --> pte index:0x8  pte contents:(valid 1, pfn 0x3e)
      --> Translates to Physical Address 0x7d3 --> Value: 0f
Virtual Address 7215:
  --> pde index:0x1c  pde contents:(valid 1, pfn 0x32)
    --> pte index:0x10  pte contents:(valid 1, pfn 0x16)
      --> Translates to Physical Address 0x2d5 --> Value: 0b
Virtual Address 2cff:
  --> pde index:0xb  pde contents:(valid 1, pfn 0x00)
    --> pte index:0x7  pte contents:(valid 1, pfn 0x7b)
      --> Translates to Physical Address 0xf7f --> Value: 15
```

自映射

1. 0xfffff000
2. 0xffc00000[la >> 10 & (~0x3)]

置换算法

答案

1. 局部置换是在当前进程占用的物理页面范围内置换；全局置换是基于不同进程的物理页面需求是不同的，在所有可换出的物理页面范围内置换；

局部置换算法：LRU，FIFO，CLOCK，OPT

全局置换算法：工作集置换算法、缺页率置换算法

2. CLOCK：4次 LRU：3次 (如果一开始内存布局是abcd，和课件一样)
3. Belady现象：分配的物理页面增加，缺页次数也增加。OPT、LRU、带计数的LFU不存在Belady现象，其他存在。

进程切换

1. 切换进程地址空间:`lcr3(next->cr3);`
2. 恢复下一个进程的EIP: `pushl 0(%eax)`
3. 切换进程内核栈:`load_esp0(next->kstack + KSTACKSIZE);`
4. 恢复从context数据结构中恢复edi，24是edi字段的偏移量；eax中是`next->context`基址；

进程创建

01432 N
01342 Y
03142 Y
01234 N
03412 Y