Queries in Lab 4

1) Can we assume that we need to copy just one program header to the destination? We are anyway doing hoh_assert(numpages == 1, "XXX"). I saw the code of elf_numpages, and there we are counting one page for each program header. Am I correct?

Answer - No. Program headers and the number of pages are independent of each other.
So your conclusion is wrong. How many loadable program headers does the given example app has?

2) What do the following stand for: masterro, masterrw, sharedrw, startip, stackend?

Answer - masterrw: Page shared between kernel and the app - for system call, stack, etc.

(Ignore masterro and sharedrw fields - they must be zero as of now - we'll give the rationale and design philiosophy behind them during help session)

startip and stackend: I just answered on another post.

3. Do we have to specifically zero out the masterro, sharedrw and rank field? As they are already assigned to zero in the process_t structure?

Answer - No.

4. Can you please provide some documentation for bitpool, especially alloc, ... why does alloc return a 16 bit index? what is the index? Why should rank in process_t be initialized to?

5. Do we need to define ring3_preempt in order to make 4.1 work?.. my gut feeling is that we should as the kernel state is already lost and we want to take back control to the kernel.. what is the kernel state invariant that we can use to initialise the state to something that fits in our kernel? and finally, How do I change the page table I'm standing on? Won't we need some dummy entries as we do during bootstage? how do i get them?

Answer - > rank?
rank: zero
> Do we need to define ring3_preempt in order to make 4.1 work?..
No. You need to define ring3_preempt for 4.2.
> ring3_step: This function shall not return. So you don't need to save current stack pointer or local variables.
After 4.1, the control shall be with ring3 - the only way you check whether the app is in ring3 is: through qemu.log or by dumping cs register value and check the privilege level.
And whether you're executing correct instructions can be through hoh_debug (If you set IOPL flags before you jump to ring3)
> what is the kernel state invariant that we can use to initialise the state to something that fits in our kernel?
excellent question! we'll cover this in help session.
> How do I change the page table I'm standing on? Won't we need some dummy entries as we do during bootstage? how do i get them?
In 4.1, The given APIs won't even let you change "the page table you're standing on".

You'll do it in 4.4. Once you finish 4.4 - you'll know how.

6. Then what is the role of ring3_step_done?

Answer - ring3_step_done: is not needed in 4.1. It's needed only for 4.2 and 4.4.

4.2: When you implement timer, you've to turn the timer off.

4.4: to implement system calls .

7. When you implement timer, you've to turn the timer off. But we would be using one shot timer. So why would we require to turn it off ?

Answer - Assume, he yielded explicitly (using systemcall/ int 0x48).. it's better to turn the timer off.

8. Understanding ring3/app1/x86/boot.S file - What do the macros PC_RELATIVE and UNPC_RELATIVE in the boot.S file do ?

Answer - To understand what these macros are doing, here is a partial dis-assembly of the binary generated by the app code. Which makes the things very clear.

```
_start                    00000220 <_start>:
  jmp main_start              220:    eb 09          jmp    22b <main_start>
  nop                         222:    90             nop
  nop                         223:    90             nop
  jmp user_exception          224:    e9 89 00 00 00 jmp    2b2 <user_exception>
  nop                         229:    90             nop
  nop                         22a:    90             nop

main_start:                 0000022b <main_start>:
  movl %esp, %ebp             22b:    89 e5          mov    %esp,%ebp
  call 1f                     22d:    e8 00 00 00 00 call   232 <main_start+0x7>
1:  popl %ebx                 232:    5b             pop    %ebx
  UNPC_RELATIVE(1b,%ebx)      233:    8d 9b ee ff ff ff lea    -0x12(%ebx),%ebx
  subl $32, %esp              239:    83 ec 20       sub    $0x20,%esp
  leal  -4(%ebp), %esi        23c:    8d 75 fc       lea    -0x4(%ebp),%esi
  movl  %esi, -12(%ebp)       23f:    89 75 f4       mov    %esi,-0xc(%ebp)
  leal  -8(%ebp), %esi        242:    8d 75 f8       lea    -0x8(%ebp),%esi
  movl  %esi, -16(%ebp)       245:    89 75 f0       mov    %esi,-0x10(%ebp)
  movl  12(%ebp), %esi        248:    8b 75 0c       mov    0xc(%ebp),%esi
  movl  %esi, -20(%ebp)       24b:    89 75 ec       mov    %esi,-0x14(%ebp)
  movl  8(%ebp), %esi         24e:    8b 75 08       mov    0x8(%ebp),%esi
  movl  %esi, -24(%ebp)       251:    89 75 e8       mov    %esi,-0x18(%ebp)
  movl  4(%ebp), %esi         254:    8b 75 04       mov    0x4(%ebp),%esi
  movl  %esi, -28(%ebp)       257:    89 75 e4       mov    %esi,-0x1c(%ebp)
  movl  0(%ebp), %esi         25a:    8b 75 00       mov    0x0(%ebp),%esi
  movl  %esi, -32(%ebp)       25d:    89 75 e0       mov    %esi,-0x20(%ebp)
  PC_RELATIVE(core_mem_init, %esi) 260:    8d b3 f0 00 00 00 lea    0xf0(%ebx),%esi
```

As the binary can get loaded at any location, we need to know the new value of _start symbol.

UNPC_RELATIVE : This macro loads ebx register with the new value of _start symbol. Observe that, ebx is not modified anywhere further in the code and it is being used only by the macro PC_RELATIVE.

PC_RELATIVE : To calculate the addresses relative to the newly calculated _start symbol. It uses the ebx register and calculates the new values of the symbols.
Example : Consider your binary with symbols :

_start : 0
sym1 : 20
sym2 : 50

Now if your binary is loaded at location 1000 then

UNPC_RELATIVE , will store new _start value i.e. (1000 + 0) in ebx and

PC_RELATIVE will give the position independent values of sym1 and sym2 to be 1020 and 1050.

9. Questions about process_t - What is the need of the following fields :
1. startip
2. stackend
Also why is fpu_simd allocated 1024B, when it actually needs only 512 B for fxsave

Answer – startip stores the starting address of eip, incase if you want to reset the process state. It is also required for 4.3.
Stackend is used for detect stack overflow.
Also why is fpu_simd allocated 1024B, when it actually needs only 512 B for fxsave. FXSAVE saves only FPU and SIMD4 registers. Currently in this lab, FXSAVE is enough.
But for future uses, we need to support SIMD8. And for that we have to use XSAVE instruction which will require more than 512 bytes.
Please note that the data in ELF header is an user input. And it can be any random number.
If we find a bug in your code
- if it's critical: ie if it can be exploited to get arbitary user code executed in ring0 level: we'll award 0 out of 2 marks.
- if it's major: ie. if it can lead to crashing of OS, but can't be exploited in ring0 level: we'll award 0.5 out of 2 marks.
- if it's minor: then it's fine, there's no extra penalty: we'll award 1 out of 2 marks.

It's perfectly fine if you call hoh_assert and shutdown the system, if you detect invalid input.
Example of minor bug is: not checking 'align' field (Semantics of align field is taught in class) etc.
Btw If you find such bug in Alice's code, please mail me immediately. It's highly appreciated. (Note that several checks in util/elf.h are deleted on purpose).

PS: Why OS should be secure? (Aka why there's a penalty if OS isn't secure)

Imagine, there's no malicious user app, and we trust all the user app. Then: we can implement co-operative scheduling, we do NOT need ring3. etc. Then there's no need for OS. We just need a HAL

library (or library OS).

But unfortunately, you're implementing ring3. => There's malicious user app.

10. well known state - What is the "well known state" that the assignment question talks about? Currently I'm changing esp and the segment registers.

Answer - You're only expected to save the current state to process structure. in part 4.2.
Restoring to well known state is done for you. Before ring3_preempt is getting called:

1. Segment registers are already restored to kernel code/data segments.

2. kernel interrupts are getting called with esp=main_stack_end-k. And then only hardware pushes SS:ESP, EFLAGS and CS:EIP. And interrupt handlers pushes errorcode and interrupt num on the stack. So there's no need for you to restore esp as well. At max you can pop theose 5+2 registers(optional).

After your code is executed:

1. EBP is set to zero,

2. Args to core_loop are pushed onto stack

3. EIP is set to core_loop.

11. What about sti.. won't we have to?

Answer - Yes you need to do it - since there is no iret in the preempt path.

12. If we only require to save process structure in the table in part 4.2, then suppose we do nothing in this part.... still the shell should be responsive.. shouldn't it. It would execute as if a new instance of the process was called.. Then why does make qemu give an unresponsive shell?

Answer - I guess, you forgot to set the timer. Or I think your timer value is very less. I would suggest that you increase it. Also check if your timer interrupt is getting generated. If it is, then

13. Can anyone start a discussion on "How ring3/app1 works?"

Answer - How it works?? The startip points to the start label in ring3/app1/boot.S .. The control reaches there. It then copies the kernel passed arguments to where it wants them to be (currently they are in masterrw). Then it calls core_reset and core_loop. Look for them in ring3/app1/main.cc. The remaining code is for exception handling. (if core_loop ever returns.. it shouldn't)

14. Should we save the "callee saved registers" in ring3_step, or, is the compiler going to do it for us?

Answer - We don't need to save any registers in ring3_step as far as I remember.
But, as ring3_step is a function, and as we are overwriting the registers (that the kernel had just before calling this function), shouldn't we follow gcc calling conventions and save the callee saved registers?

15. Check - Can you please post a good check for 1.12. Thanks.

Answer - Check:
Generate an int3 or a page fault yourself. and see if it is getting reported correctly.

16. "state" variable - What is the new "state" variable that has appeared in the process_t struct ? What should it be initialized to ?

Answer - We need it in 4.4. It is already initialized to zero. state is needed in 4.4 only.

17. Use of errorcode at an interrupt - What is the use of the error code which is being pushed by the isr handlers above the trapframe ? I found that most of the isr are pushing 0 as errorcode. An example / use case would really help.

Answer - Yes, most of the ISRs push 0. That is to maintain a uniform trapframe. Only some ISRs actually need the error number. I think page fault does.

18. Can someone explain what exactly we have to do in ring3_step in 4.1?

Answer - You have to allow execution of user app here for 1 time slice. It can be described in 3 steps :
1. Restore all the app registers.
2. Set the timer.
3. Jump inside the ring3, to the exact position where the computation was stopped last time.
HINT : Look at xv6 code. How first process is created in xv6.

19. On exit() or invalid syscall number, we are supposed to free the resources (masterrw, to .. (what else ?)). if we free masterrw, then there would be a problem since we are writing to it at the end of the function. Even if we free the masterrw at the end, we need to know the control flow.... Where does the ring3_downcall return to ?

20. ring3_downcall is a normal C function called after ring3_step_done().

```
ring3_downcall(){

 switch(fnum){

          case 1: {  // kill the proc:

                    proc->state=PROC_DONE;

                    free the resources

                    return; // process is terminated. so return.

                  }break;       case 2: {

                     //don't return.

                   }break;
}
```

}

assuming your ring3_step will ignore() the request, if proc.state==PROC_DONE.

void ring3_step(){

  if(proc.state==PROC_DONE){

    return;
  }

//process_restore..

}

21. Here are a few questions on 4.4.

1. How do we indicate the failure of a system call ? Won't it be bad if the system call fails silently? For example, if a user performs mmio read on an invalid location.
2. I didn't understand the syscall mmu_swapva. If I am not wrong, this syscall will swap the virtual addresses of the 2 pages. Why do we need this system call ?
3. Correction/Clarification : In the code, arguments and return values are declared as uint32_t (i.e. 4 B each), but the lab page indicates that the arguments are 8 B. Also we are asked to clear first 64 bytes of masterrw but we are not using more than 20 B.

Answer - 1) If the mmio read on an invalid location happens, we'll get a page-fault and it should've been given to user-app as an upcall. I don't think we're handling this case currently.

2) I think Deepak Sir would answer this better. AFAIK, We are trying to have symmetry between user and kernel code. Allowing a process to manage which VA is mapped where (and providing an interface to change it) can be used to enhance symmetry. This is just my intuition, although. (I'm thinking of parent and child processes)

Our system call type is:

  system_call::(Arg1,Arg2,Arg3) -> (Ret1,Ret2,Ret3)

And if you want to return the status of operation, ie:

  system_call::(Arg1,Arg2,Arg3 -> OK (Ret1,Ret2,Ret3) | Error Errorcode

This can be encoded as:

  system_call :: (Arg1, Arg2,Arg3) -> (OK|Errorcode , dRet2, dRet3)

 In short:

   All we've to say that: the status will be one more return argument to the systemcall. Let's say, ret1.

We still have to make syscall_mmio[1] to zero.

But in this part: We don't have permission checks and interrupt handler router.

And it's not easy to detect failures for the system call which we asked.

For example:

mmio::read/mmio::write.

- Status should fail if the user don't have permission. And we haven't implement authorization aka capabilities.

Read about capabilities if you actually want to implement authorization - it's better than ACL which we used Linux. And ACL's are fundamentally broken.

When we add capabilities, the structure of code is going to change - and capablities are not covered in this course. And this part is supposed to be few lines of code.

In short: For this part: if there's an issue with system call.. just kill it. or just do an hoh_assert.

I think if the system call fails (i.e. user performs mmio read on invalid location), we just kill the process.

22. Understanding ELF -

I wrote this short code, complied it and did a objdump

```
#include<stdio.h>
int  main(void){
  printf("\nHello World\n");
  return 0;
}
```

```
 hello_world$ objdump -p hello_world.o

hello_world.o:     file format elf64-x86-64

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr 0x0000000000400040
align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr 0x0000000000400238
align 2**0
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000400000 paddr 0x0000000000400000
align 2**21
         filesz 0x000000000000070c memsz 0x000000000000070c flags r-x
    LOAD off    0x0000000000000e10 vaddr 0x0000000000600e10 paddr 0x0000000000600e10
align 2**21
         filesz 0x0000000000000230 memsz 0x0000000000000238 flags rw-
 DYNAMIC off    0x0000000000000e28 vaddr 0x0000000000600e28 paddr 0x0000000000600e28
```

```
align 2**3
         filesz 0x00000000000001d0 memsz 0x00000000000001d0 flags rw-
    NOTE off    0x0000000000000254 vaddr 0x0000000000400254 paddr 0x0000000000400254
align 2**2
         filesz 0x0000000000000044 memsz 0x0000000000000044 flags r--
EH_FRAME off    0x00000000000005e4 vaddr 0x00000000004005e4 paddr 0x00000000004005e4
align 2**2
         filesz 0x0000000000000034 memsz 0x0000000000000034 flags r--
   STACK off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000
align 2**4
         filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
   RELRO off    0x0000000000000e10 vaddr 0x0000000000600e10 paddr 0x0000000000600e10
align 2**0
         filesz 0x00000000000001f0 memsz 0x00000000000001f0 flags r--
```

```
hello_world$ objdump -f hello_world.o

hello_world.o:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000400440
```

I have 2 doubts:

1. What are all these prog headers : PHDR, DYNAMIC, NOTE.
2. start address 0x0000000000400440 is in Prog. Header NOTE range. Why ?

Answer - I found this manpage. Just do a Ctrl + F, unless you have a Mac :P

http://manpages.ubuntu.com/manpages/raring/man5/elf.5.html

22. System Calls

In mmio_read and mmio_write, will the "size" argument take all possible values, or, will it be only 8, 16 and 32?

Answer - only 8, 16 and 32

23. How do we check whether the second argument "addr_t" is a valid address for the uproc to access? Also, how do we do this for pmio?

Answer - No need to check. just write a simple one-liner.

24. The question mentions : mmio_read: read size bytes from the given address using mmio

Does this mean size represent the number of bytes to be read? Based on this piazza discussion, I understand that size means 8-bit or 16-bit or 32-bit. Could you please clarify and update.

Answer - it's 8 bit, 16 bit and 32 bit.

25. pmu_mappmio - What is the system call "pmu_mappmio" for? Is it to just set the iopl flags to 3?

Answer - Yes. keep it simple and make it a oneliner.

26. case PMU_MAPPMIO: {

proc.iopl = 3.

break;

}

And in ring3_step()...make sure you respect this flags.

Answer - Original functionality:

- the pmu_mappmio takes a port number as an argument.

- the system call should check if the application is allowed to map the particular port

- if allowed, it grants exactly that port using - TSS's iopb bitmask.

For this part: you grant access to all ports and we don't do any authorization/access checks.

26. I am little confused with the following statement in the 4.4 statement :

User shall pass arguments through begin of page shared between user and kernel. Memory layout:

0: reserved. must be zero.
4: Syscall num. Zero indicates No syscall request pending.
8: Syscall Arg1 / Syscall Ret1.
16: Syscall Arg2 / Syscall Ret2.
24: Syscall Arg3 / Syscall Ret3.

As i see in the code in ring3_downcall.h file, each argument is uint32_t type, meaning 4 bytes. Then shouldn't it be the case that the 2nd argument in the above statement be at 12 and 3rd argument be at 16.
I may be doing a very silly mistake. Can someone clarify ?

Answer - Yes, you are correct.
It should be :

0: reserved. must be zero.
4: Syscall num. Zero indicates No syscall request pending.
8: Syscall Arg1 / Syscall Ret1.
12: Syscall Arg2 / Syscall Ret2.
16: Syscall Arg3 / Syscall Ret3.

27. process_t missing fields

process_t - Doesn't have the following fields : PID, TSS, parent

Would they come later ?

Answer -
TSS: Why do we need TSS?

---

1. To setup the kernel_stack.:

We wanted to save the current kernel stack pointer in TSS and ask the interrupt handler/system call to run on top of the saved stack pointer.

In HoH: Unlike Linux: This location is a constant, and doesn't change per process.

So we don't need TSS per every process.

2. For IOPB:

We don't want to give control to all the IO ports for an application. just like we use MMU for MMIO, x86 uses IOPB field (8KB bitmask) in TSS for IO.

In any system only few process requires access to IO ports - for example Keyboard driver.

And it's not fair to have this extra 8KB per process.

Current plan is to statically verify the K/B driver's binary code, and not have IOPB per process.

PID

No. I don't see any reason to have unique PID per process. I don't have an alternative for PID - but it's not PID for sure.

Reasons:

1. PID is global. and thus adds an extra/unncessary synchronization among all the cores. And also makes it non-deterministic.

One alternative approach(forgot the paper):

when you create a process, give it an ID. and this ID is local to the children.

Another alternative approach:

have a PID-server process which assigns PID to processes it manages. This is the approach

chosen by GNU/HURD Operating System.
Current plan is to take the HURD approach - It'll make it easier to run Hurd on HoH.

Parent

We won't have a parent field as well.

Parent is needed only during creation. During its initial stage, children is programmed to get attached to whoever is providing the necessary resources. After that children can manage it themselves. And parents won't be there throughout child's lifetime. And the default strategy of - a parent waiting for its children to die - is not a good default for any system, IMHO.

IMHO - In My Humble Opinion.

28. Do we need to save the current context of the ring0 code before we switch to the ring3 code? Where do we save it?

Answer - We need not save the context because the function ring3_step never returns.

29. 1. Is there possibility a race condition in Ring3 Preemption.
2. process_t is it page aligned ?

```
struct process_t{
  public:
#define process_offset_fpu_simd  0
    unsigned char fpu_simd[1024] ALIGN(0x1000);
```

Is it assured that no padding bits will be put in between process_t and fpu_simd.

(What if process_t is assigned an address not aligned to 0x1000

Answer - Deepak sir said that he will ensure that there will be no race conditions. Also, he said that fpu_simd will always be 16-bit aligned.

30. free up resources in 4.4 - do we need to free up the proc structure also in the case of syscall 1. lab4

Answer - No.

31. Is the emergency stack used to implement system calls?
Can you please tell me how the tss entry of the gdt was initialized? I seem to be lost in Alice's code.

Answer - > Is the emergency stack used to accept system calls?
Yes. System calls can be called from emergency stack as well as normal user stack.
But: you cannot have an exception while you're in emergency stack - remember we're overwriting the emergency stack.
Btw, IIRC, In schedular activation paper and FCRB paper, emergency stack is called activation stack.

> Can you please tell me how the tss entry of the gdt was initialized? I seem to be lost in Alice's code.

Read The.. Code. Can you ask a specific question.

32. save current page during swap -
1. When we swap one of the page during page fault, who (kernel/user) handles that the contents of the current page is saved first. Is it already implemented or we need to implement it?

2. I believe we need to change files only in ring3/app1 folder and nothing in the labs folder for this part. Please confirm.

Answer - when you get page fault, in kernel(ring3_upcall - which is implemented by you), what did you do?

- the swap system call (mmu_swapva) is implemented by you - how did you implement it?

So, if a page fault occured in ring3, kernel doesn't even what it is - it blindly forwards that exception to ring3 app.

That means kernel has nothing to do with page fault in ring3 - ie. kernel has nothing to do with swaping.

So if a ring3 app gets page fault, and if it decides to swap, it should handle it, right?

And you're writing the entire ring3 app from scratch - nothing is implemented for you.

2. In lab5: You're writing an entire application - so yes. only files under ring3/app1 needs to be modified.

(You'll may have to fix your bugs in your previous parts (under labs/ folder), if any..)