- **Arrange the following in the correct order of execution (w.r.t. 'init')**
  - userinit() is called-> 1
  - 'initcode' struct proc is created-> 2
  - 'initcode' process is marked RUNNABLE-> 3
  - mpmain() calls scheduler()-> 4
  - scheduler() schedules initcode() process-> 5
  - initcode() returns in forkret()-> 6
  - initcode() returns from trapret()-> 7
  - initcode() calls exec("/init", ...)-> 8

init related execution sequence (Matching)

- **Map the virtual address to physical address in xv6**
  - KERNBASE-> 0
  - KERNLINK-> 0x100000
  - 80108000-> 0x108000
  - 0xFE000000-> 0xFE000000
  - -> 0x80000000

kernel memory mappings (Matching)

- **The approximate number of page frames created by kinit1 is**

  a. **(100%)** 3000

  b. **(0%)** 1000

  c. **(0%)** 2000

  d. **(0%)** 4000

  e. **(0%)** 10

  f. **(0%)** 4

  g. **(0%)** 16

#kinit1's pages (Multiple choice / One answer only)

- **Select all the correct statements about initcode**

  a. **(25%)** code of 'initcode' is loaded along with the kernel during booting

  b. **(25%)** the size of 'initcode' is 2c

  c. **(25%)** The data and stack of initcode is mapped to one single page in userinit()

  d. **(25%)** initcode essentially calls exec("/init",...)

  e. **(-33.33333%)** initcode is the 'init' process

  f. **(-33.33333%)** code of initcode is loaded in memory by the kernel during userinit()

  g. **(-33.33333%)** code of initcode is loaded at virtual address 0

correct about initcode (Multiple choice)

- **Which of the following is DONE by allocproc() ?**

  a. **(20%)** Select an UNUSED struct proc for use

  b. **(20%)** allocate PID to the process

  c. **(20%)** allocate kernel stack for the process

  d. **(20%)** setup the trapframe and context pointers appropriately

  e. **(20%)** ensure that the process starts in forkret()

  f. **(-33.33333%)** ensure that the process starts in trapret()

  g. **(-33.33333%)** setup kernel memory mappings for the process

  h. **(-33.33333%)** setup the contents of the trapframe of the process properly

not done by allocproc() (Multiple choice)

- **Which of the following is done by mappages()?**

  a. **(33.33333%)** create page table mappings for the range given by "va" and "va + size"

  b. **(33.33333%)** allocate page table if required

  c. **(33.33333%)** create page table mappings to the range given by "pa" and "pa + size"

  d. **(-50%)** allocate page directory if required

  e. **(-50%)** allocate page frame if required

not done by mappages (Multiple choice)

- **What does seginit() do?**

  a. **(100%)** Adds two additional entries to GDT corresponding to Code and Data segments, but to be used in privilege level 3

  b. **(0%)** Adds two additional entries to GDT corresponding to Code and Data segments, but to be used in privilege level 0

  c. **(0%)** Nothing significant, just repetition of earlier GDT setup but with 2-level paging setup done

  d. **(0%)** Nothing significant, just repetition of earlier GDT setup but with free frames list created now

  e. **(0%)** Nothing significant, just repetition of earlier GDT setup but with kernel page table allocated now

seginit() does? (Multiple choice / One answer only)

- **Select the statement that most correctly describes what setupkvm() does**

  a. **(100%)** creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray

  b. **(0%)** creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray and makes kpgdir point to it

  c. **(0%)** creates a 2-level page table for the use of the kernel, as specified in gdtdesc

  d. **(0%)** creates a 1-level page table for the use by the kernel, as specified in kmap[] global array

setupkvm()'s job (Multiple choice / One answer only)

- **What does userinit() do ?**

  a. **(100%)** sets up the 'initcode' process to start execution in forkret()

  b. **(0%)** sets up the 'init' process to start execution in forkret()

  c. **(0%)** sets up the 'initcode' process to start execution in trapret()

  d. **(0%)** sets up the 'initcode' process to start execution in forkret ()

  e. **(0%)** initializes the users

  f. **(0%)** initializes the process 'init' and starts executing it

userinit() does? (Multiple choice / One answer only)

- **The variable 'end' used as argument to kinit1 has the value**

  a. **(100%)** 801154a8

  b. **(0%)** 80110000

  c. **(0%)** 80000000

  d. **(0%)** 81000000

  e. **(0%)** 80102da0

  f. **(0%)** 8010a48c

value of end (Multiple choice / One answer only)

- **Does exec() code around clearptau() lead to wastage of one page frame?**

  a. **(100%)** yes

  b. **(0%)** no

wastage in exec? (Multiple choice / One answer only)

- **exec() does this: curproc->tf->eip = elf.entry, but userinit() does this: p->tf->eip = 0; Select all the statements from below, that collectively explain this**

  a. **(33.33333%)** exec() loads from ELF file and the address of first instruction to be executed is given by 'entry'

  b. **(33.33333%)** In userinit() the function inituvm() has mapped the code of 'initcode' to be starting at virtual address 0

  c. **(33.33333%)** the initcode is created using objcopy, which discards all relocation information and symbols (like entry)

  d. **(-33.33333%)** the 'entry' in initcode is anyways 0

  e. **(-33.33333%)** the code of 'initcode' is loaded at physical address 0

  f. **(-33.33333%)** elf.entry is anyways 0, so both statements mean the same

why different eip settings? (Multiple choice)

- **Why is there a call to kinit2? Why is it not merged with knit1?**

  a. **(100%)** knit2 refers to virtual addresses beyond 4MB, which are not mapped before kvalloc() is called

  b. **(0%)** Because there is a limit on the values that the argumets to knit1() can take.

  c. **(0%)** When kinit1() is called there is a need for few page frames, but later knit2() is called to serve need of more page frames

  d. **(0%)** call to seginit() makes it possible to actually use PHYSTOP in argument to kinit2()

why kinit2()? (Multiple choice / One answer only)