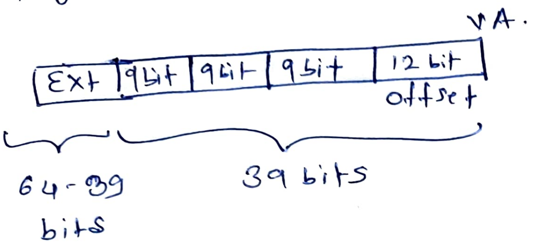
Page-tables RISC-V 3-level.



∴ 2^{27} combinations of pages + 12 bit offset to search in particular page.

every process if $\boxed{\text{allocated}}$; get 0 to MAXVA address space in ElF binary.

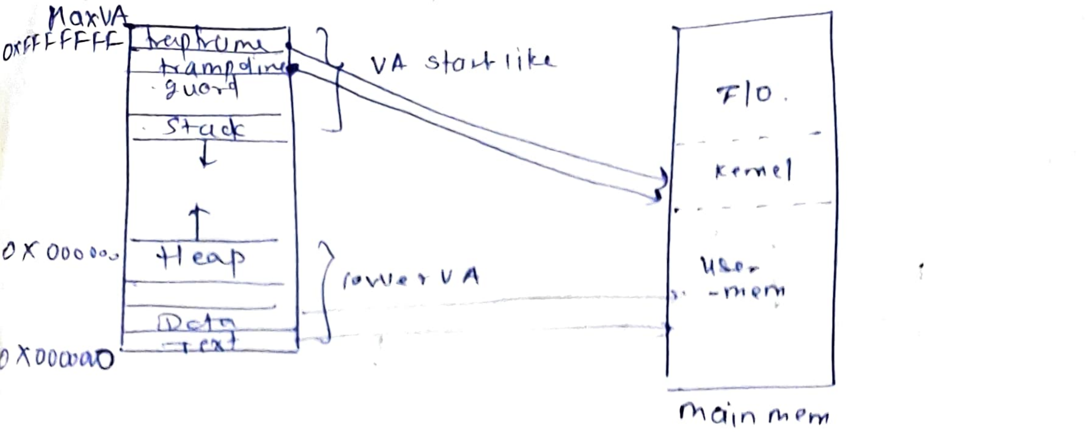
And while loading it into RAM, mappings are made.

if pagetable wonts to map, and we use single page table

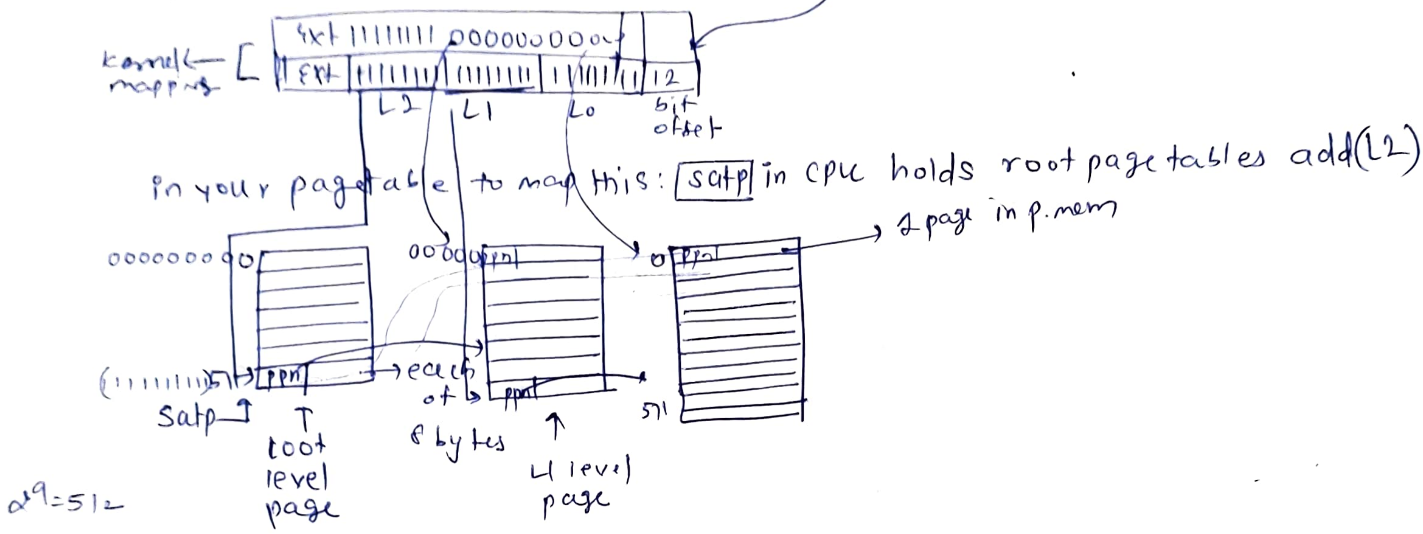
assuming each PTE will be $\boxed{\text{8bytes}}$ 2^{27} x 8bytes = ________ bytes.

this is quite huge and many smaller programs that are only 8KB require very few mappings ∴ waste of such huge mem.

then comes on demand paging & 3-level page-table.



∴ High address VA: 0xffffff c000000 → 0xfffffffff



if ppn is empty
I'll allocate ~~L1~~ page.

allocated 10 pages in total

in attack.c

Total allocated 10 pages from fo

in user/attacktest.c:

① if (fork() == 0)
// wait()
↳ the child will load in mem (secret.c)
②
③
↳ exec()
sys_exec is called
We know proc_pagetable allocates 3 pages
P10,P11,P12
↳ for child you have .text, .data for that low VA + 2 pages. + .guard .stack

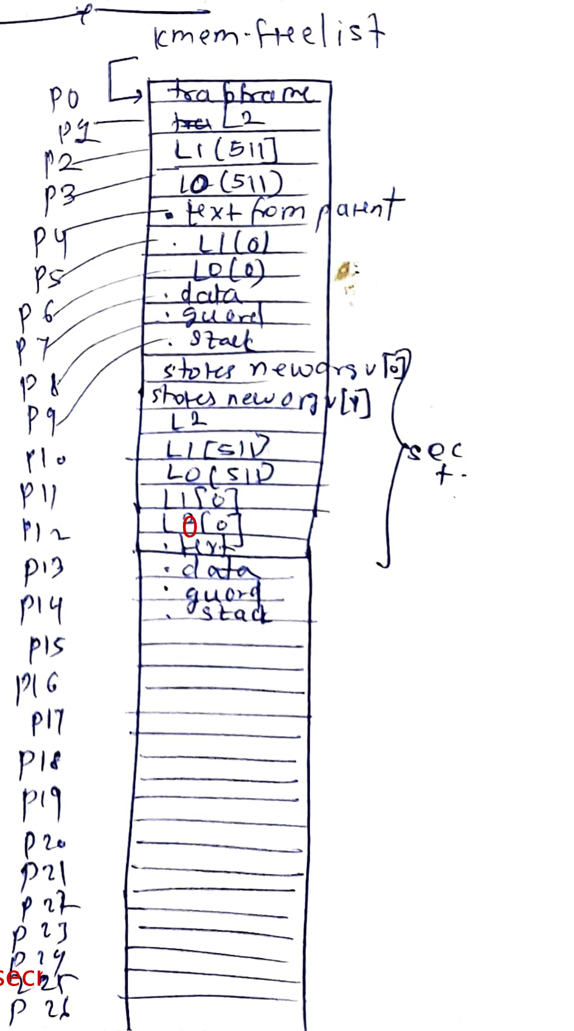
L0[0], L1[0]
.text .data
.guard .stack
P13-P18

in proc.c
fork()
{
allocproc()
↳ allocated 4 pages total

allocproc()
① allocates trapframe } 1 page total
P0
proc_pagetable()
↳ ~~trampo~~line L2 pagetable
↳ trampoline to map
3 pages total
L2,L1[511],L0[511]
P1 P2 P3

2) uvmcopy()
copies user pages of parent p (attacktest.c)
∴ (.text, .stack, .guard, .data) at lower VA
∴ you walk through page-table alloc(pagentor)it.
but lower VA mapping you need L1(0), L0(0)
P4 P5 P6 P7 P8 P9

pages are allocated to hold
newarv[0]->'secret' newargv[1]=secr



The next step before leaving exec is to free the allocated memory

This is first time of freeing the pages

sys_exec(){

↳calls free-prop-pagetable()

free-propp~~g~~table() ??

① → unmaps the TRAMPOLINE AND TRAPFRAME from pagetable without clearing them. (it is still in ~~RA~~ free-mem)

② uvmfree firstly frees the user memory pages with order low vaddr to high addr.

~~sys~~

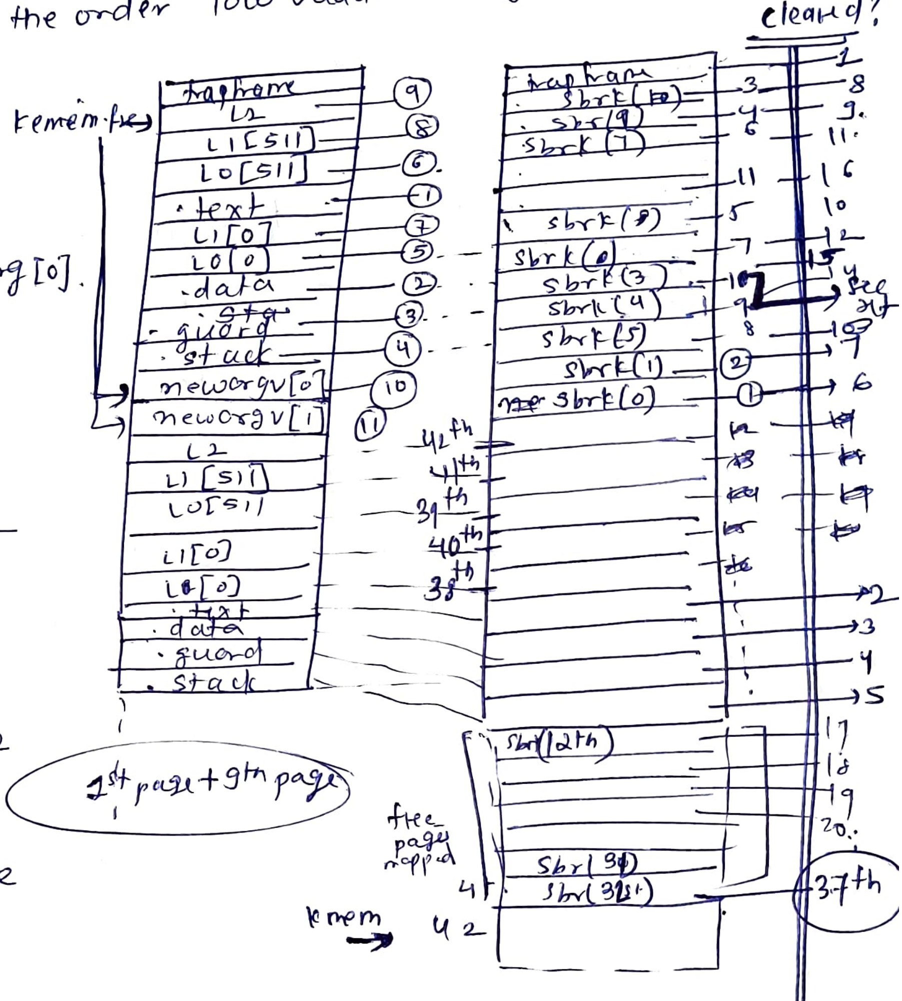
③ uvmfree secondly frees the pagetable pages by the order low vaddr to high addr. L0 to L2.

kmem.freelist → points to freshly ~~freed~~ page. freed

↳ kfree() → cleas page for kwargs → neworg[0].

then go to ~~sys-exec~~ user-space

↳ your start secret process from user space.



summary of sys_exec behaviour:

first allocates

- pages for keyword arguments
- pagetable pages
- user pages loaded from ELF file and two more pages stack & stack guard

Then free:

- the pages of old process image (excluding TRAPFRAME & TRAMPOLINE)
- the kernel pages allocated for key word argument

→ sys_exec will now return to user_space

→ The user process secret now starts.

① allocate 32 pages

↳ 42 pages have been released so far

page with secret was released at 15

42 — npage+0
41 — npage+1
40 — npage+2
39
38 ;
37
;
npage+27
15

↳ after secret completes; → it will exec → process becomes zombie

wait(0) will call free()

by secret get's cleared at 15th

the 42 pages are reordered

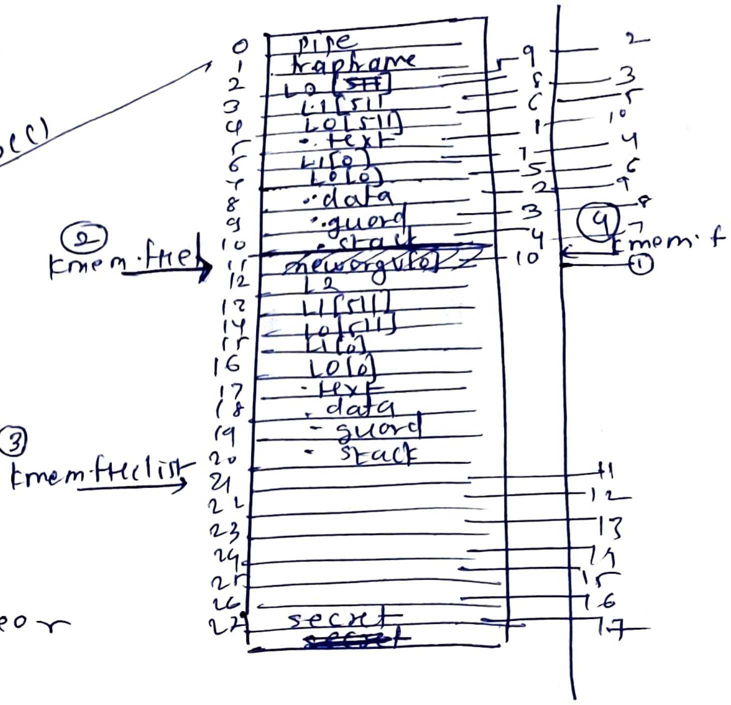
now you return back to attacktest.c

```
else{
wait(0) // ✓
if(pipe(fds) <0){        pipealloc()
//

}
if(fork() <0){      (allocates 10 pages in total)
// for attack.c
```

then exec allocates pages for ELF.

① returning from sys-exec, you clear old process image.



② clears page of keyword arguments.

sys-exec returns to userspace

to reach np+27 you have to allocate: 17 pages. 17th page will be holding secret

```
∴    int np=17;
     char *end = sbrk(np*PGSIZE);
     end = end +(-1 np) PGSIZE;
       write(2, end+32, 8);
       exit(1);
     }
```