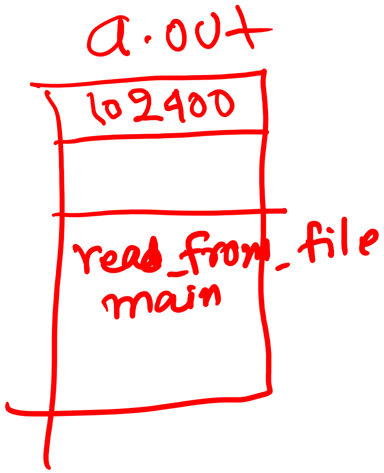
Executable and linkable format (ELF)

```
int numentries = 102400;
int buf[102400];
int main () {
  read_from_file (buf, numentries);
  sort (buf, numentries);
  print_data (buf, numentries);
  return 0;
}
```



ELF

```
int numentries = 102400;
int buf[102400];
int main () {
  read_from_file (buf, numentries);
  sort (buf, numentries);
  print_data (buf, numentries);
  return 0;
}
```

.text

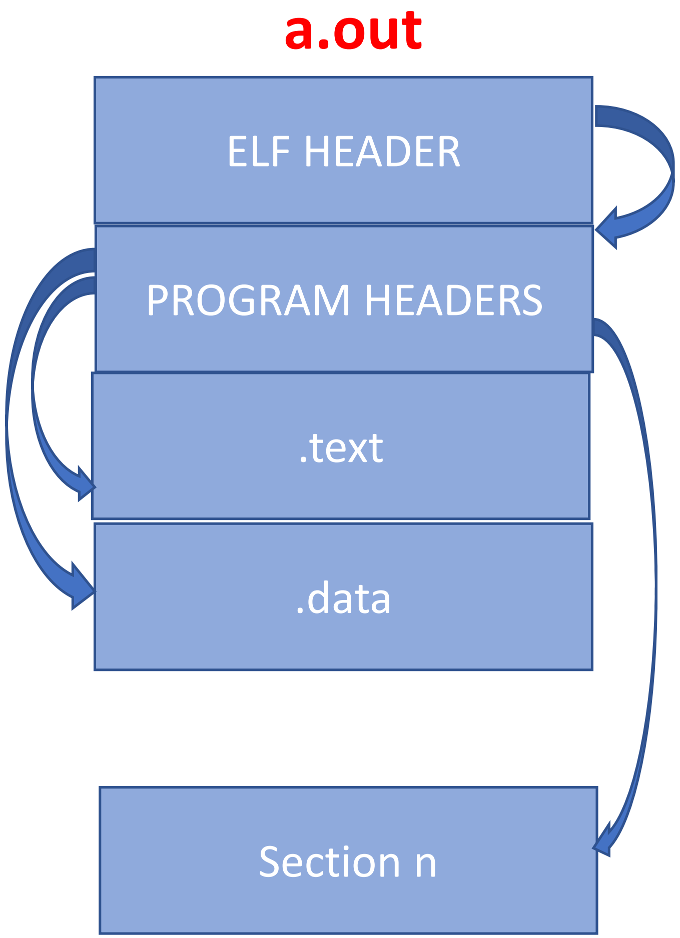
cpu instrutions of main, read_from_file, sort, print_data, etc.

.data

numentries, buf

.stack

ELF



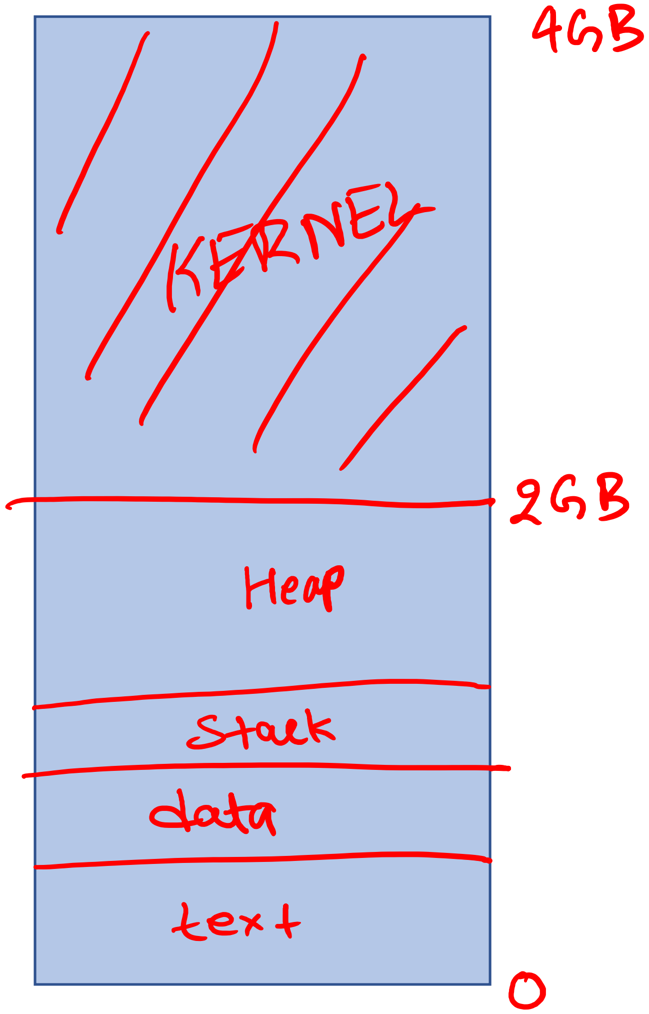
elfhdr: 1005

```
struct elfhdr {
    uint magic;     /* must equals to 0x464C457FU */
    uchar elf[12];
    ushort type;
    …
    uint phoff; /* offset of first program header */
    …
    ushort phnum; /* number of program headers */
}
```

proghdr: 1024

```
struct proghdr {
    uint type;
    uint off;
    uint vaddr;
    uint paddr;
    uint filesz;
    uint memsz;
    uint flags;
    uint align;
};
```

Process address space



- text
- data
- unmapped
- stack
- heap

exec: 6310

exec (char *path, char **argv)

- Open executable file
- Read elf header from the executable file
- Create page directory using setupkvm

setupkvm:1837

setupkvm()

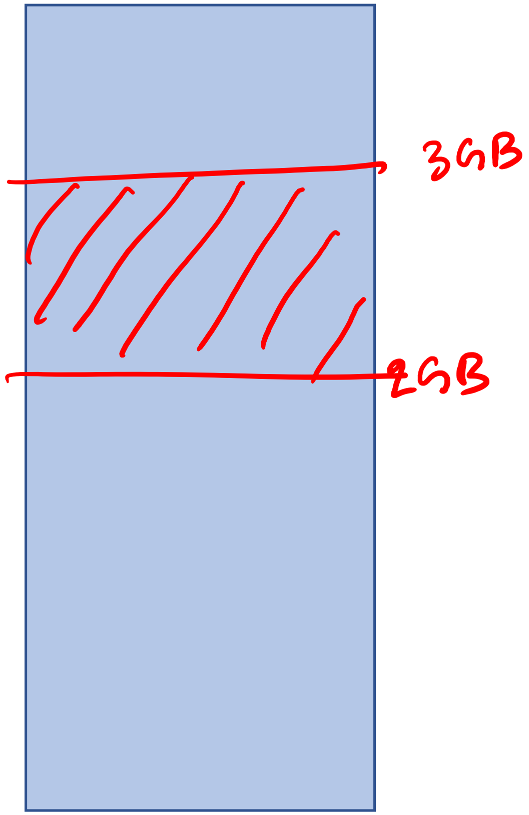
- Create page directory
- map I/O space
  - {“0”, “1MB”} -> {“KERNBASE”, “KERNBASE+1MB”}
- map kernel text
  - {“1MB”, “textsize”} -> {“KERNBASE+1MB”, “KERNBASE+1MB+textsize”}
- map rest of the RAM
  - {“1MB+textsize”, “ramsize”} -> {“KERNBASE+1MB+textsize”, “KERNBASE+ramsize”}

mappages:1779

mappages (pgdir, va, size, pa, perm)

- Map physical address range {pa, pa+size} to virtual address range {va, va+size}

Virtual address space after setupkvm



exec: 6310

exec (char *path, char **argv)

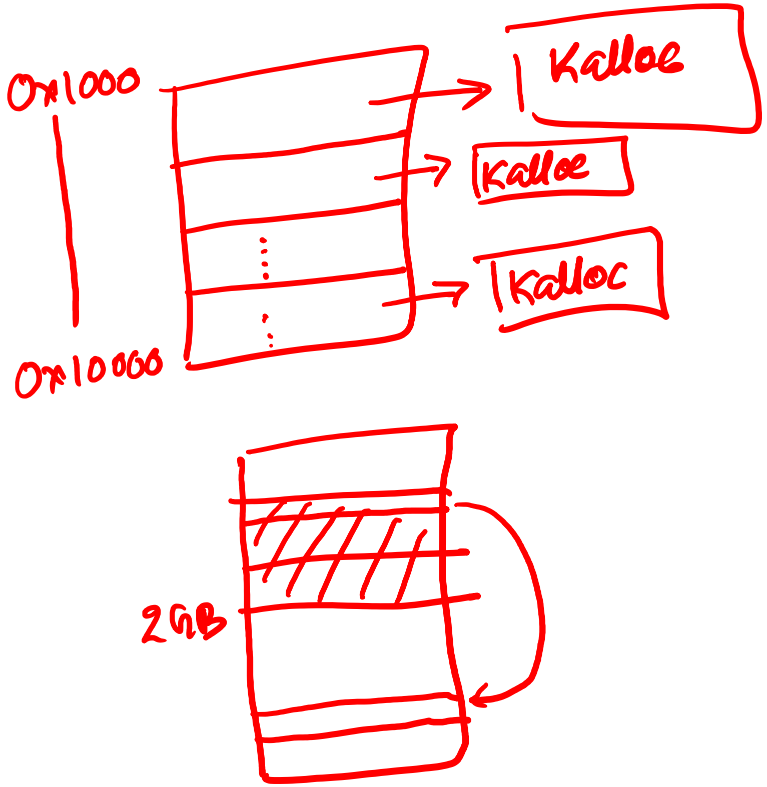
vaddr ✓

memsz ✓

- read program headers from file
- for every program header
  - fetch the virtual address (vaddr) and memory size (memsz) of the section from the program header
  - call allocuvm to map physical pages at the virtual address range {vaddr, vaddr+memsz}

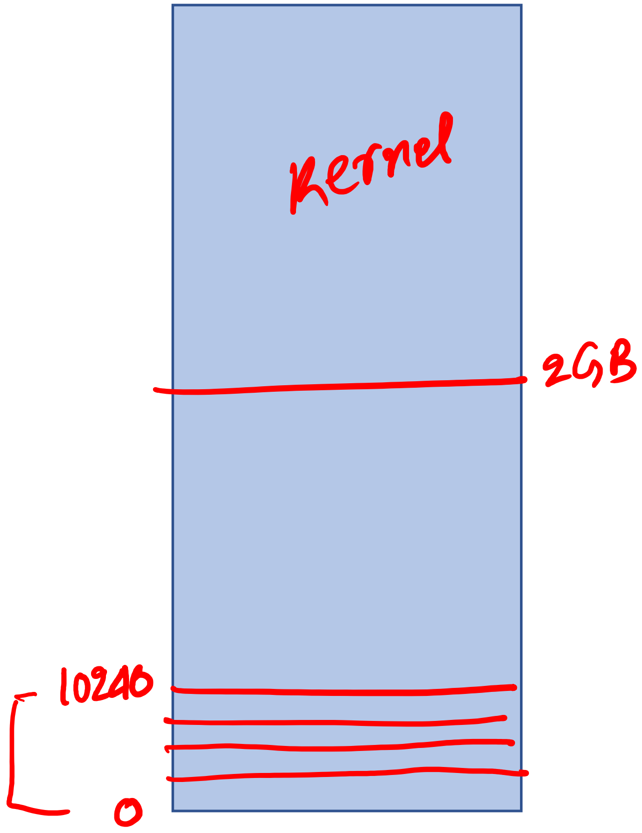
allocuvm: 1953

allocuvm (pgdir, oldsz, newsz)



- allocate physical pages for all the virtual pages between oldsz and newsz
- allocate physical pages using kalloc
- map physical pages using mappages
- return newsz on success

Virtual address space after allocuvm



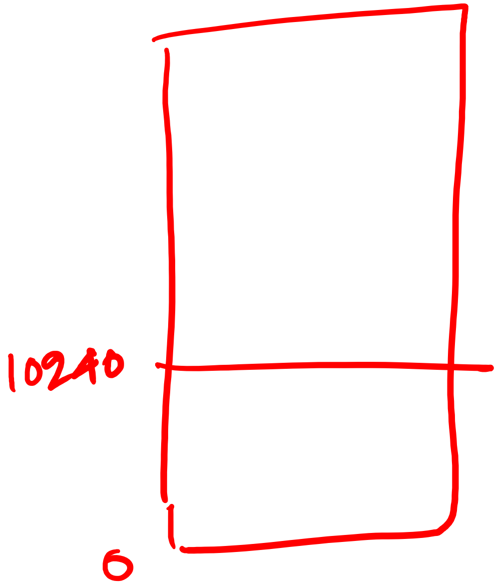
exec: 6310

exec (char *path, char **argv)

- read program headers from file
- for every program header
  - fetch the virtual address (vaddr) and memory size (memsz) of the section from the program header
  - call allocuvm to map physical pages at the virtual address range {vaddr, vaddr+memsz}
  - call loaduvm to load the content of the section from the file

loaduvm: 1918

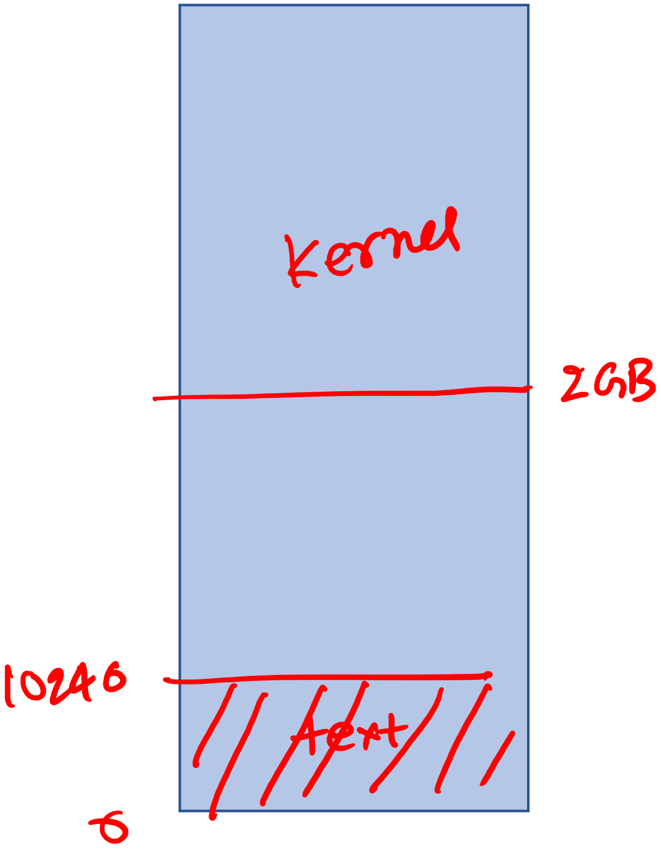
loaduvm (pgdir, addr, ip, offset, sz)



- read “sz” bytes from the input file at the given offset and store them at the virtual address “addr” mapped in the input pgdir

- Why does loaduvm walk the page directory to first get the physical address instead of directly copying into the virtual address?
  - pgdir may not be the currently active page table

Virtual address space after loaduvm

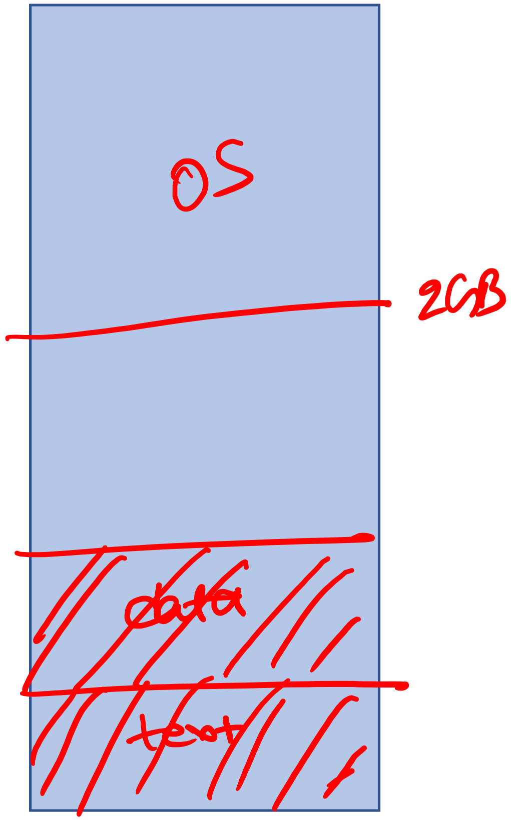


exec: 6310

exec (char *path, char **argv)

- read program headers from file
- for every program header
  - fetch the virtual address (vaddr) and memory size (memsz) of the section from the program header
  - call allocuvm to map physical pages at the virtual address range {vaddr, vaddr+memsz}
  - call loadvm to load the contents of the section from the file

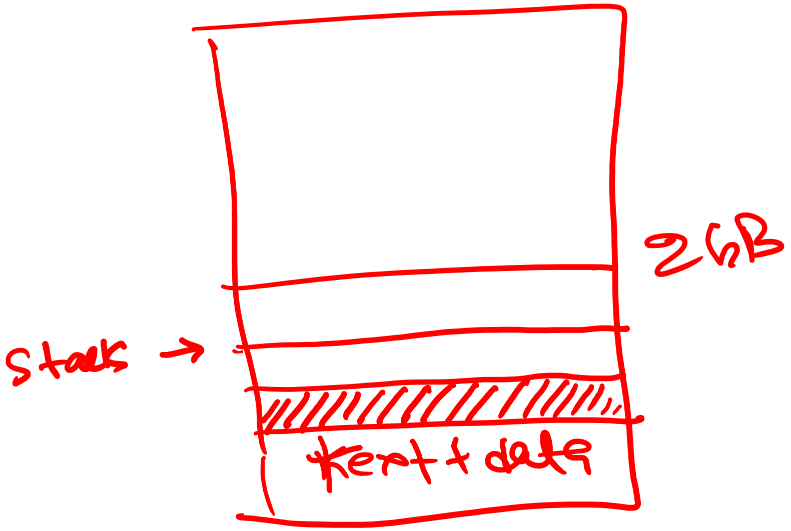
Virtual address space after for loop



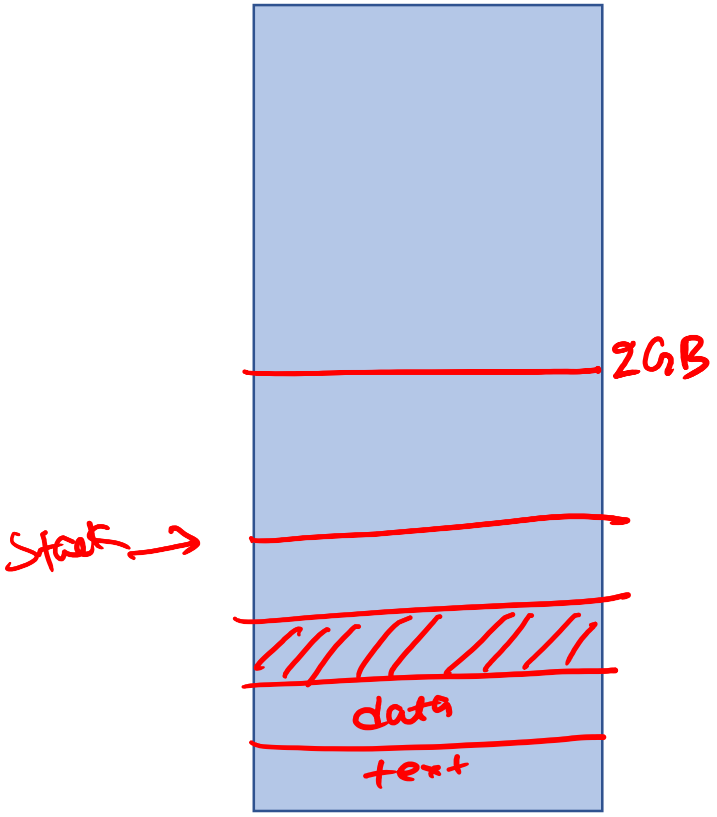
exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate two pages for stack
- make the first page inaccessible
  - by revoking the user access



Virtual address space after stack allocation



exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate two pages for stack
- make the first page inaccessible
  - by revoking the user access
- copy arguments of main to stack

Copy arguments to stack

- Our goal is to call “main (int argc, char *argv[])” on this stack

- exec is called with null terminated “char **argv”

- for each string in argv
  - allocate space on stack
  - copy the string to the allocated space

- set up the arguments, a fake return address on the stack

Copy arguments to stack

char *argv[] = {“foo”, “bar”, NULL};

argc = 2;

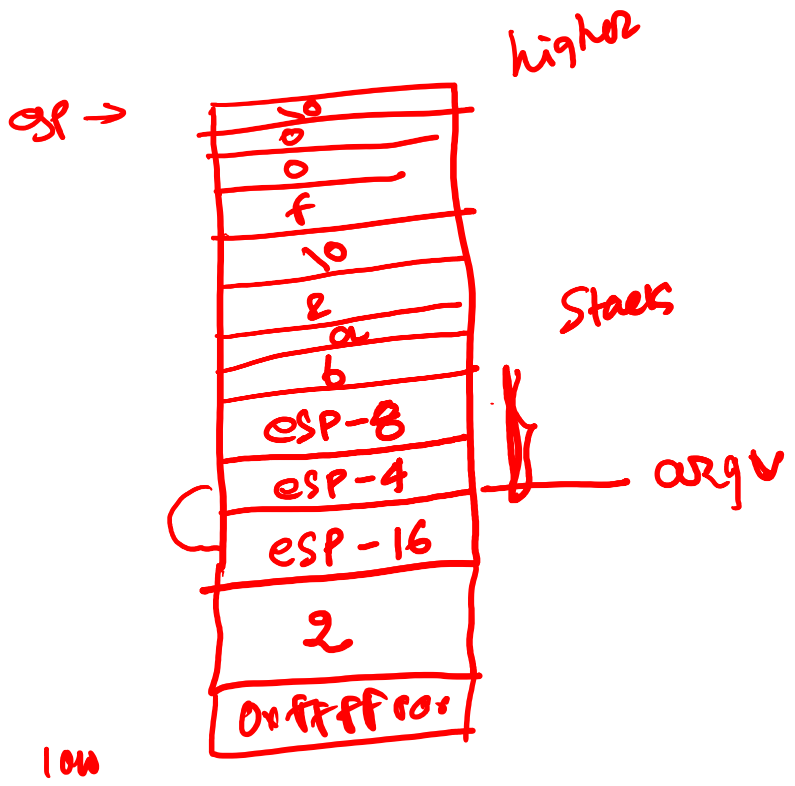
exec (argv)

arg[] = {&foo,
&bar
};

foo

bar

main(int argc, char *argv[])



exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate stack and copy arguments
- adjust process size

exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate stack and copy arguments
- adjust process size
- rewrite eip with the main of executable and esp with the new stack in the trap frame (pushed during the exec system call)

trapframe : 602

```
struct trapframe {
   uint edi;
   …
   uint eax;
   ushort gs;
   ushort padding1;
   …
   uint trapno;
   uint err;
   uint eip;
   ushort cs;
   …
};
```

alltraps: 3254

```
ss, esp, eflags, cs, eip  // pushed by hardware
error_code, vector no // pushed by vectors.S
alltraps:
  push all segment registers
  push all general purpose registers
  trap (&esp)  /* esp contains the address of
                  trapframe */
```

trap:3351

- trap (struct trapframe *tf)
- sets proc->tf to the current trapframe on system call at 3356
- exec sets “proc->tf->eip” to new executable “main” at 6396
- exec sets “proc->tf->esp” to new stack at 6397

exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate stack and copy arguments
- adjust process size
- rewrite eip and esp in trapframe
- call switchuvm to load new page table

switchuvm: 1873

- disable interrupts

- setup TSS to point to the process kernel stack

- load the new page table

- restore the original interrupt flags

exec: 6310

exec (char *path, char **argv)

- read program headers from file
- load all sections
- allocate stack and copy arguments
- rewrite eip and esp in trapframe
- adjust process size
- call switchuvm to load new page table
- call freevm to free all the user pages in the old page directory

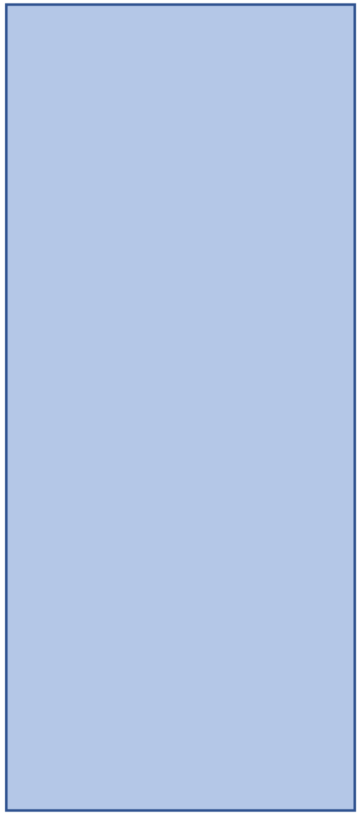
freevm: 2015

- free all user pages

- free all page table pages

- free the page directory

Virtual address space after exec

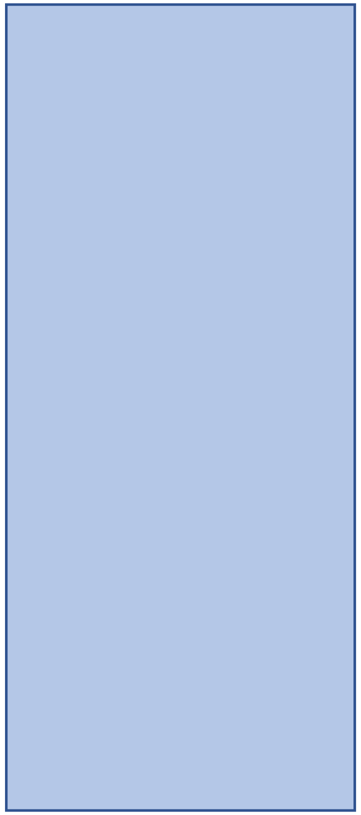


allocate space for heap

- malloc uses sbrk
- sys_sbrk (int n) : 3701
  - growproc(int n) : 2535
    - adjust the process size by n bytes
    - if n is positive
      - call allocuvm
    - if n is negative
      - call deallocuvm

- Why does “growproc” call “switchuvm”?

Process address space



Does OS has to map the entire RAM in the kernel address space?

- Linux reserves virtual addresses between 3GB – 4GB for kernel

- Windows reserves 2GB – 4GB for the kernel (similar to xv6) but can be configured to use 3GB – 4GB

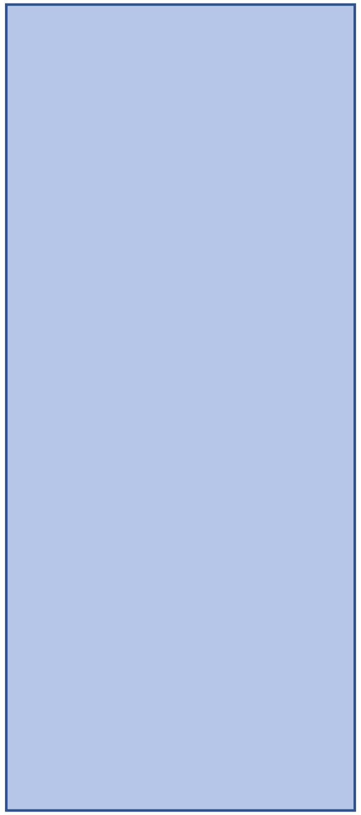
Page fault

- Page fault exception is raised by the hardware when a virtual address is dereferenced without sufficient privilege or no physical address is mapped corresponding to the virtual address

- Page fault is useful for copy on write optimization
  - The OS can make a copy, reinstate the write privilege and resume the application

- The hardware restart the execution of the partially executed instruction after returning from the exception handler

Demand paging



Demand paging

- Maintain another data structure corresponding to every page table (say frame table)
- Corresponding to every virtual page store the file identifier and offset in the frame table
- On page fault allocate a physical page, read data from the given frame offset in the frame table, and map the page in the page table