**1.vm.c (xv6 design & implementation. (xv6 source code))**

**Ans:**

**Vm.c(allocuvm):**

int allocuvm(pde\_t \*pgdir, uint oldsz, uint newsz)

{

char \*mem;

uint a;

if(newsz >= KERNBASE) return 0; if(newsz < oldsz) return oldsz;

a = PGROUNDUP(oldsz);

for(; a < newsz; a += PGSIZE){

mem = kalloc();

if(mem == 0){

cprintf("allocuvm out of memory\n"); deallocuvm(pgdir, newsz, oldsz); return 0;

}

memset(mem, 0, PGSIZE);

if(mappages(pgdir, (char\*)a, PGSIZE, V2P(mem), PTE\_W|PTE\_U) < 0){ cprintf("allocuvm out of memory (2)\n");

deallocuvm(pgdir, newsz, oldsz);

kfree(mem);

return 0;

}

}

return newsz;

}

**Vm.c(deallocuvm):**

int deallocuvm(pde\_t \*pgdir, uint oldsz, uint newsz)

{

pte\_t \*pte; uint a, pa;

if(newsz >= oldsz) return oldsz;

a = PGROUNDUP(newsz);

for(; a < oldsz; a += PGSIZE){

pte = walkpgdir(pgdir, (char\*)a, 0);

if(!pte)

a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;

else if((\*pte & PTE\_P) != 0){

pa = PTE\_ADDR(\*pte);

if(pa == 0) panic("kfree");

char \*v = P2V(pa);

kfree(v);

\*pte = 0;

}

}

return newsz;

}

**Vm.c(seginit):**

void seginit(void)

{

struct cpu \*c;

// Map "logical" addresses to virtual addresses using identity map.

// Cannot share a CODE descriptor for both kernel and user // because it would have to have DPL\_USR, but the CPU forbids // an interrupt from CPL=0 to DPL=3.

c = &cpus[cpunum()];

c->gdt[SEG\_KCODE] = SEG(STA\_X|STA\_R, 0, 0xffffffff, 0); c->gdt[SEG\_KDATA] = SEG(STA\_W, 0, 0xffffffff, 0);

c->gdt[SEG\_UCODE] = SEG(STA\_X|STA\_R, 0, 0xffffffff, DPL\_USER);

c->gdt[SEG\_UDATA] = SEG(STA\_W, 0, 0xffffffff,DPL\_USER);

// Map cpu, and curproc c->gdt[SEG\_KCPU] = SEG(STA\_W, &c->cpu, 8, 0); lgdt(c->gdt,sizeof(c->gdt));

loadgs(SEG\_KCPU << 3);

// Initialize cpu-local storage.

cpu = c; proc = 0;

}

**2.ps, back trace (xv6 customization)** Ans:

**Ps.c**

**Follow the steps:**

Step1: Ps.c

#include "types.h"

#include "stat.h"

#include "user.h"

#include "uproc.h"

#define STDOUT 1

#define NPROC 3 // maximum number of processes void getprocdata()

{ int i = 0;

struct uproc up;

printf(STDOUT, "Name\tpid\tppid\tsize\t\twaiting\_on\_channel\tkilled\n");

printf(STDOUT, "-----------------------------------------------------------------------------\n");

for (i=0; i<NPROC; i++) {

getprocinfo(i, &up);

printf(STDOUT, "%s\t%d\t%d\t%d\t\t%d\t\t%d\n", up.name, up.pid, up.ppid, up.sz, up.wait, up.killed); }

exit(); }

int main(int argc, char \*argv[])

{

getprocdata(); exit();

}

Step 2: Syscall.c

extern int sys\_getprocinfo(void);

[SYS\_getprocinfo] sys\_getprocinfo,

Step 3: Syscall.h

#define SYS\_getprocinfo 22

Step 4: Sysproc.c int sys\_getprocinfo()

{

int proc\_num,

size = sizeof(struct uproc);

struct uproc \*up;

if ((argint(0, &proc\_num) < 0) || (argptr(1, (char \*\*)&up, size) < 0)) return -1;

return getprocinfo(proc\_num, up);

}

Step 5: User.h struct stat;

struct rtcdate; struct uproc;

//Add this in System Calls block int getprocinfo(int, struct uproc\*);

Step 6: usys.S

SYSCALL(getprocinfo)

Step 7: Proc.h

//Add at the end struct uproc{ char name[16]; int pid;

int ppid;

uint sz;

int state;

int wait;

int killed;

};

Step 8: Proc.c

// Add below code in proc.c at the end

int getprocinfo(int proc\_num, struct uproc \*up)

{ struct proc \*p;

if (proc\_num >= NPROC)

return -1;

p = &ptable.proc[proc\_num]; memset(up, 0, sizeof(struct uproc)); memmove(up->name, p->name, 16); up->pid = p->pid;

up->state = p->state;

if ((up->state != UNUSED)) {

up->ppid = p->parent->pid;

up->sz = p->sz;

}

if (up->pid == 1) {

up->sz = p->sz;

up->ppid = 0;

}

if (p->chan)

up->wait = 1;

else

up->wait = 0;

up->killed = p->killed; return 0;

}

Step 9: defs.h //Add uproc here.. struct buf;

struct context; struct file;

struct inode; struct pipe;

struct proc;

struct uproc;

//pagebreak 16 //proc.c (In defs.h) int getprocinfo(int, struct uproc\*);

Step 10: uproc.h #include "types.h" struct uproc{ char name[16]; int pid; int ppid; uint sz; int state; int wait; int killed;

};

Step 11: Makefile:

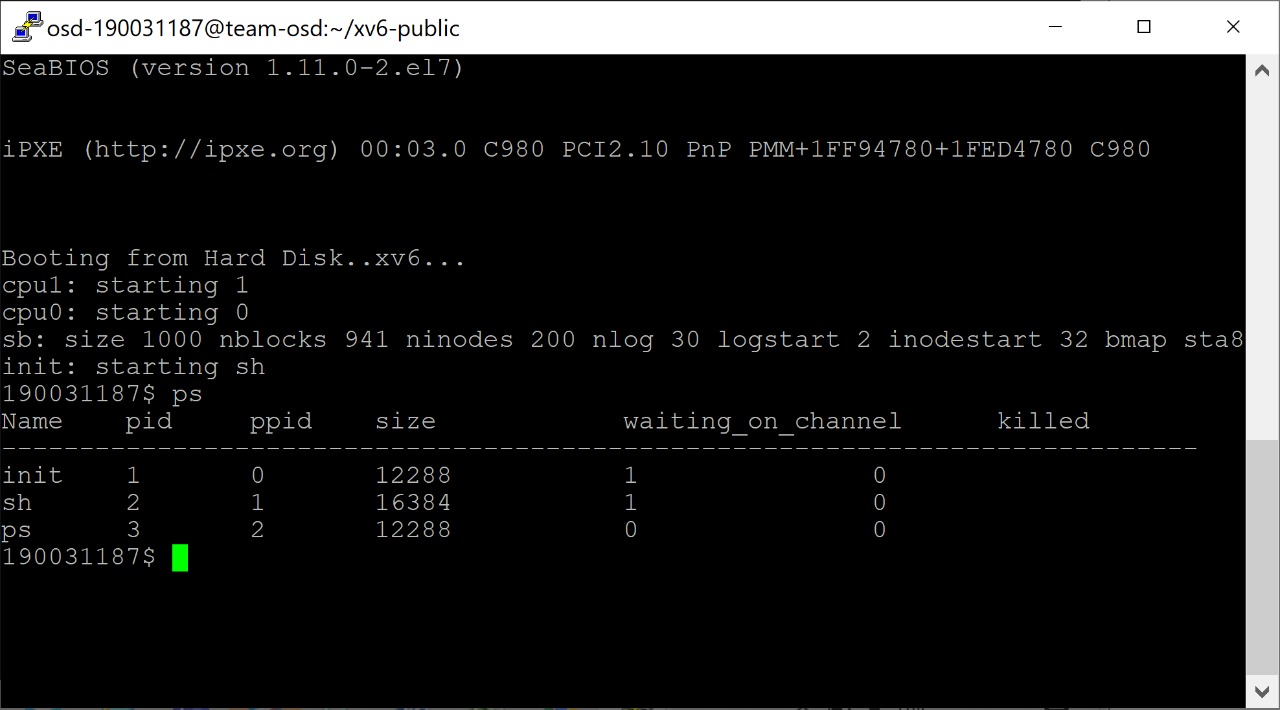
$ nano Makefile

//In UPROGS\ section, Add

\_ps\

//In Extras Section, Add ps.c

**OUTPUT**



**Backtree.c**

**Follow the steps:**

Step-1: Syscall.h

$ nano syscall.h

#define SYS\_backtrace 23

Step 2: Syscall.c

$ nano syscall.c extern int sys\_backtrace(void); [SYS\_backtrace] sys\_backtrace,

Step 3: Sysproc.c $ nano sysproc.c

int sys\_backtrace(void)

{

//struct proc \*curproc = myproc(); uint ebp, ret\_addr, next\_addr; int count = 0; struct trapframe \*tf = myproc()->tf; cprintf("eax : 0x%x\n", tf->eax); cprintf("ebx : 0x%x\n", tf->ebx); cprintf("ecx : 0x%x\n", tf->ecx); cprintf("edx : 0x%x\n", tf->edx); cprintf("edi : 0x%x\n", tf->edi); cprintf("esi : 0x%x\n", tf->esi); cprintf("esp : 0x%x\n", tf->esp); cprintf("ebp : 0x%x\n", tf->ebp); cprintf("eip : 0x%x\n", tf->eip); ebp = tf->ebp; next\_addr = tf->eip; while(next\_addr && next\_addr != (uint)-1) { ret\_addr = \*(uint \*) (ebp + 4); cprintf("#%d 0x%x\n", count++, ret\_addr); ebp = \*(uint \*)ebp; next\_addr = ret\_addr;

}

return 0;

}

Step 4: Usys.S

$ nano usys.s

SYSCALL(backtrace) Step 5: User.h

$ nano user.h int backtrace(void);

Step 6: defs.h $ nano defs.h

int backtrace(void);

Step 7: bt.c

$ nano bt.c

#include "types.h"

#include "stat.h" #include "user.h" int baz() \_\_attribute\_\_((noinline)); int baz() { int a; a = backtrace(); return a + uptime();

}

int bar() \_\_attribute\_\_((noinline)); int bar() { int b; b = baz(); return b + uptime();

}

int foo() \_\_attribute\_\_((noinline)); int foo() { int c; c = bar(); return c + uptime();

} int main(int argc, char \*argv[])

{ foo(); exit();

}

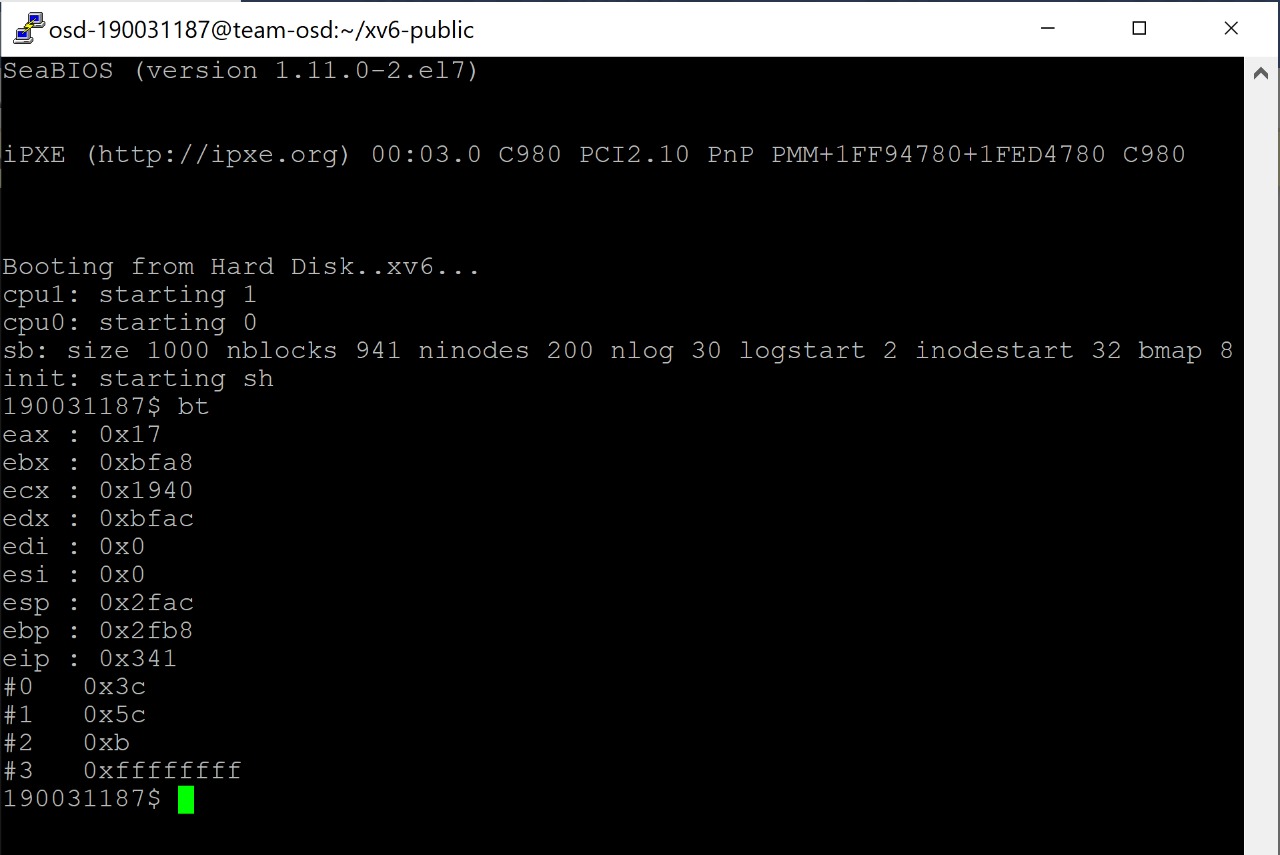
Step 8: Makefile

$ nano Makefile

UPROGS: \_bt\

EXTRAS: bt

**OUTPUT**

.c