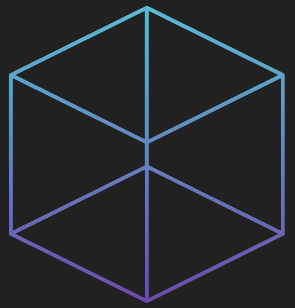


SYSTEMS
GOSSIP
MEETUP

Learning large systems using peer-to-peer gossip

Policy Against Harassment at ACM Activities

https://www.acm.org/about-acm/policy-against-harassment

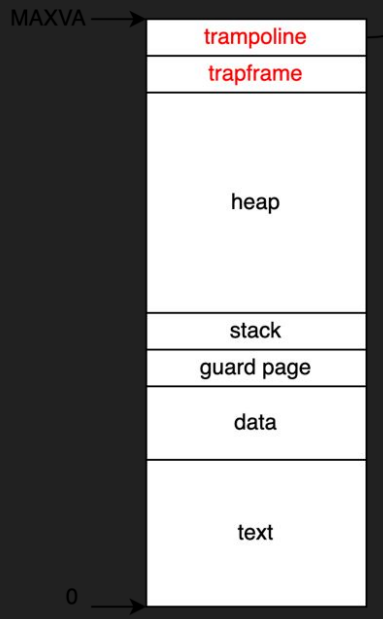
OS Meetup wants to encourage and preserve this open exchange of ideas, which requires an environment that enables all to participate without fear of personal harassment. We define harassment to include specific unacceptable factors and behaviors listed in the ACM's policy against harassment. Unacceptable behavior will not be tolerated.

Last Time

- spinlocks for multicore processing

Last Time

- spinlocks for multicore processing
- Syscall trap




```
16  uservec:
17          #
18          # trap.c sets stvec to point here, so
19          # traps from user space start here,
20          # in supervisor mode, but with a
21          # user page table.
22          #
23          # sscratch points to where the process's p->trapframe is
24          # mapped into user space, at TRAPFRAME.
25          #
26
27          # swap a0 and sscratch
28          # so that a0 is TRAPFRAME
29          csrrw a0, sscratch, a0
30
31          # save the user registers in TRAPFRAME
32          sd ra, 40(a0)
33          sd sp, 48(a0)
34          sd gp, 56(a0)
35          sd tp, 64(a0)
36          sd t0, 72(a0)
37          sd t1, 80(a0)
38          sd t2, 88(a0)
39          sd s0, 96(a0)
40          sd s1, 104(a0)
41          sd a1, 120(a0)
42          sd a2, 128(a0)
43          sd a3, 136(a0)
44          sd a4, 144(a0)
45          sd a5, 152(a0)
46          sd a6, 160(a0)
```

Last Time

- spinlocks for multicore processing
- Syscall trap
- Timer interrupt trap

```
void
timerinit()
{
  // each CPU has a separate source of timer interrupts.
  int id = r_mhartid();

  // ask the CLINT for a timer interrupt.
  int interval = 1000000; // cycles; about 1/10th second in qemu.
  *(uint64*)CLINT_MTIMECMP(id) = *(uint64*)CLINT_MTIME + interval;

  // prepare information in scratch[] for timervec.
  // scratch[0..2] : space for timervec to save registers.
  // scratch[3] : address of CLINT MTIMECMP register.
  // scratch[4] : desired interval (in cycles) between timer interrupts.
  uint64 *scratch = &timer_scratch[id][0];
  scratch[3] = CLINT_MTIMECMP(id);
  scratch[4] = interval;
  w_mscratch((uint64)scratch);

  // set the machine-mode trap handler.
  w_mtvec((uint64)timervec);

  // enable machine-mode interrupts.
  w_mstatus(r_mstatus() | MSTATUS_MIE);

  // enable machine-mode timer interrupts.
  w_mie(r_mie() | MIE_MTIE);
}
```

Today

- Threads
- Context Switching in Xv6
- Scheduler

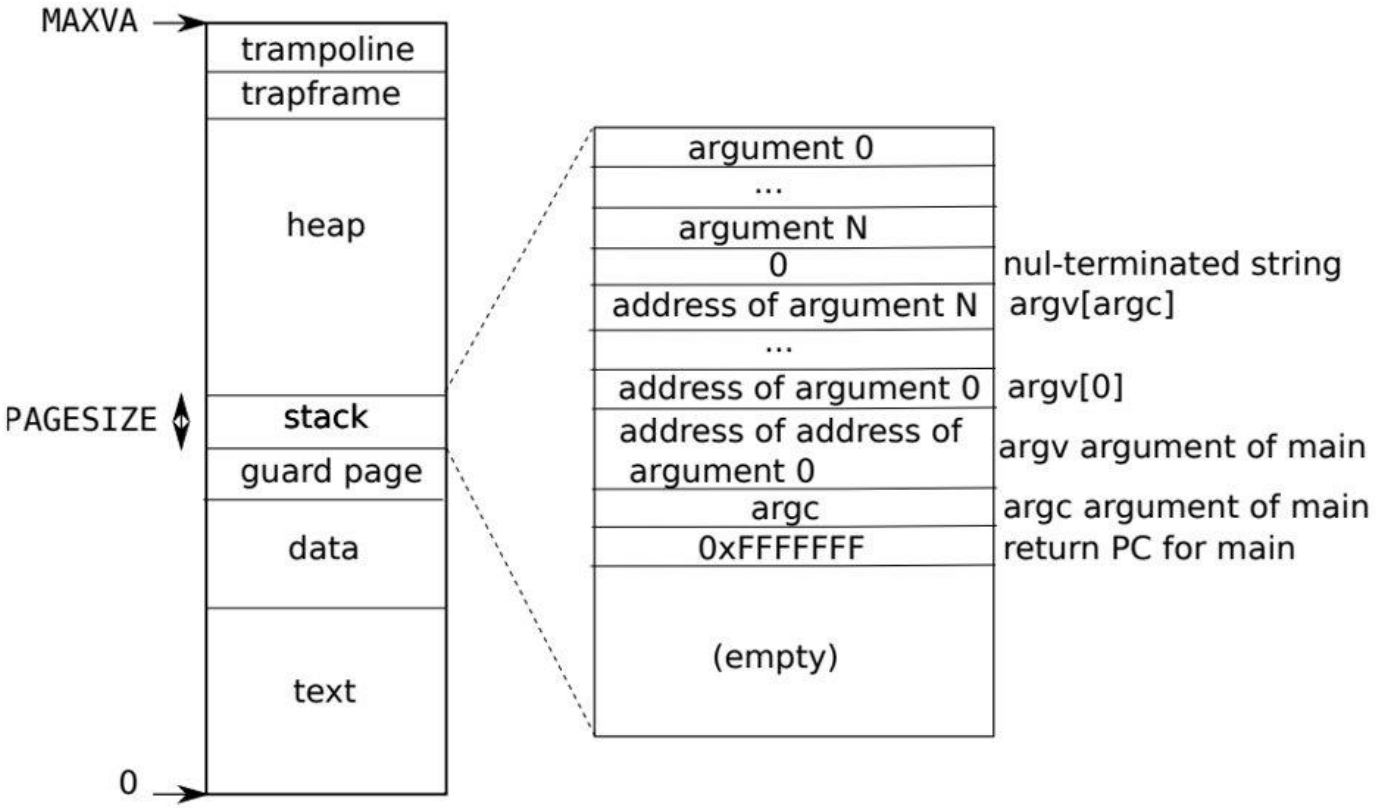


Figure 3.4: A process's user address space, with its initial stack.

Thread Switching

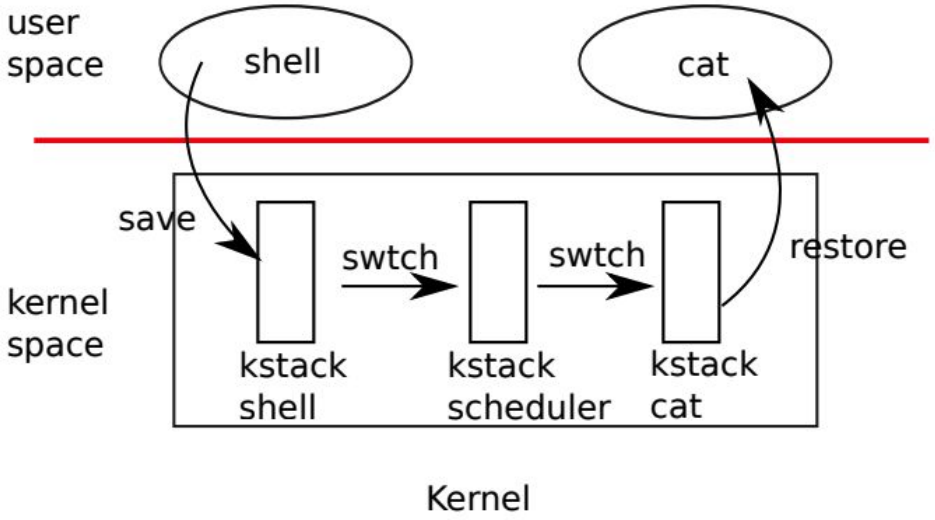


Figure 7.1: Switching from one user process to another. In this example, xv6 runs with one CPU (and thus one scheduler thread).

Threads

- In Xv6, each process has a thread of execution (we call it thread).
- If we want to support more processes than # of CPU cores, we want to transparently switch threads between processes.
- Each thread has its own thread stack, which saves local variables, parameters, return address.
- Each process has two types of stacks: user stack and kernel stack. When the process runs in user space, it executes on user stack. When the process runs in kernel space, it executes on kernel stack.

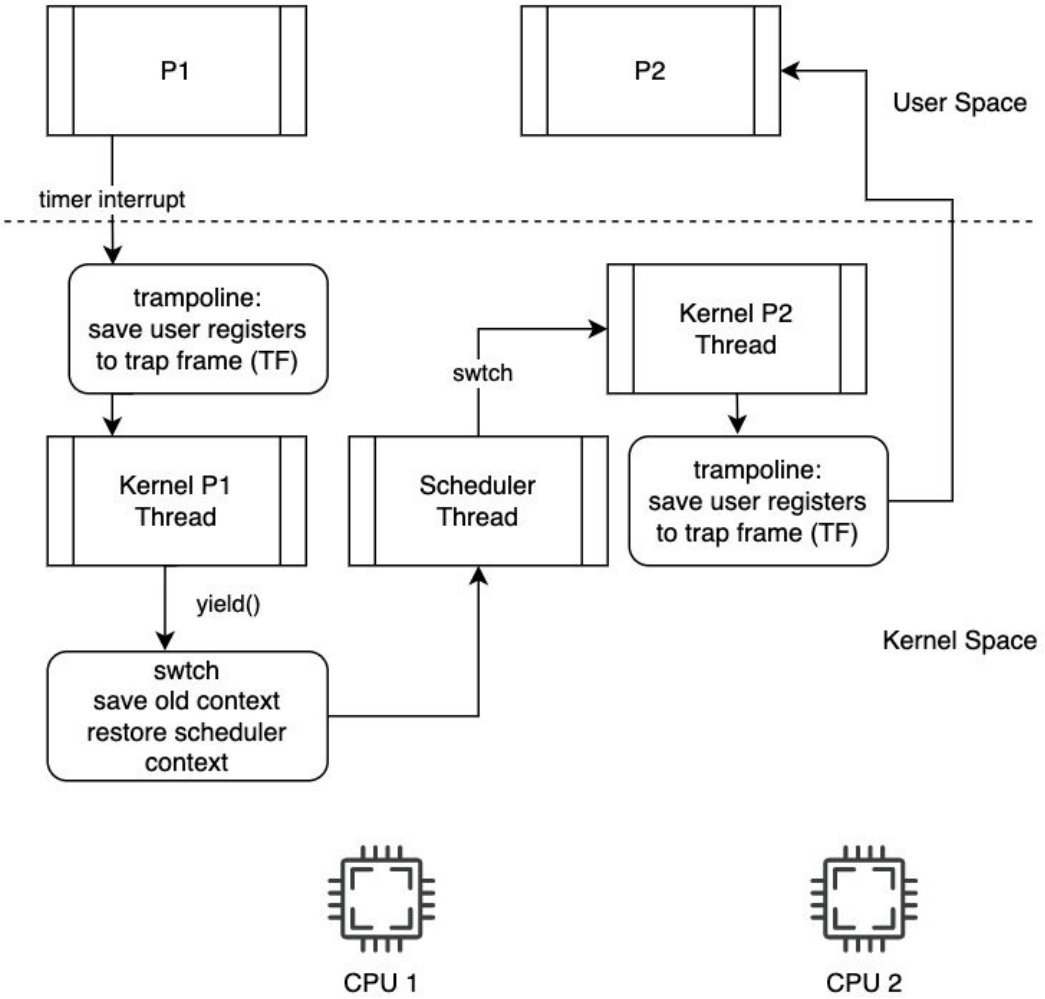
Process Lifecycle Demo

```
// Per-process state
github-classroom[bot], 2 days ago | 1 author (github-classroom[bot])
struct proc {
  struct spinlock lock;

  // p->lock must be held when using these:
  enum procstate state;        // Process state
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  int xstate;                  // Exit status to be returned to parent's wait
  int pid;                     // Process ID

  // wait_lock must be held when using this:
  struct proc *parent;         // Parent process

  // these are private to the process, so p->lock need not be held.
  uint64 kstack;               // Virtual address of kernel stack
  uint64 sz;                   // Size of process memory (bytes)
  pagetable_t pagetable;       // User page table
  struct trapframe *trapframe; // data page for trampoline.S
  struct context context;      // swtch() here to run process
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

```
enum procstate {
  UNUSED,
  USED,
  SLEEPING,
  RUNNABLE,
  RUNNING,
  ZOMBIE
};
```

-

Thread Switching



Next time

1. Lab 5 Cow is out! Due Oct 08th.
2. Lecture 12: Scheduling II