

## **LAB #2: xv6 Threads**

### **Group Members:**

- Yash Aggarwal (yagga004)
- Nityash Gautam (ngaut006)
- Parth Bhatt (pbhat029)

### **Demonstration Link:**

[https://drive.google.com/file/d/1VILYF4fSoqz8npYC9SPanR8kn89\\_v8kU/view](https://drive.google.com/file/d/1VILYF4fSoqz8npYC9SPanR8kn89_v8kU/view)

### **PART 1: Clone() System Call**

1. Add the system call in syscall.h
2. Add the mapped function in syscall.c
3. Define the mapped function in syscall.c
4. Implement the syscall function in sysproc.c
5. Add the implementation of clone function in proc.c
6. Add a thread\_id variable in proc.h
7. Make changes for the freeproc and uvmunmap() function.
8. Implement the allocproc() function with changes
9. For clone() remember:
  - a. Stack null check
  - b. Checking of page table
  - c. Starting address of child's user stack

### **PART 2: User Level Thread Library**

1. Create the files for user thread library - thread.c and thread.h
2. Define the schema or structure for the thread.
3. Create empty functions for thread create lock\_init, acquire, release
4. Initialize lock as 0.
5. Implement the functions.
6. Add the logic for lock acquire and release sync.
7. Add code for clone
8. Test the implementation

### **CONCLUSION:**

```

user1@ubuntu: ~/Downloads/my_github
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -fmcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -DRR -c -o
user/zombie.o user/zombie.c
riscv64-linux-gnu-objdump -z max_page_size=4096 -T user/user.ld -o user/_zombie user/zombie.o user/ulib.o user/usys.o user/printf.o user/umalloc.o user/thread.o
riscv64-linux-gnu-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/zombie.sym
riscv64-linux-gnu-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/zombie_test.sym
riscv64-linux-gnu-objdump -t user/_lab1_test | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab1_test.sym
riscv64-linux-gnu-objdump -t user/_lab1_test | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab1_test.o
riscv64-linux-gnu-objdump -t user/_lab1_test.o user/lab1_test.c
riscv64-linux-gnu-objdump -t user/_lab1_test.o user/lab1_test.asm
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -MD -fmcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -DRR -c -o
user/lab2.o user/lab2.c
riscv64-linux-gnu-ld -z max_page_size=4096 -T user/user.ld -o user/_lab2 user/lab2.o user/ulib.o user/usys.o user/printf.o user/umalloc.o user/thread.o
riscv64-linux-gnu-objdump -S user/_lab2 > user/lab2.asm
riscv64-linux-gnu-objdump -t user/_lab2 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab2.sym
riscv64-linux-gnu-objdump -t user/_lab2 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab2_test.sym
riscv64-linux-gnu-objdump -t user/_lab2 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab2_test.o
riscv64-linux-gnu-objdump -t user/_lab2.o user/lab2.o
riscv64-linux-gnu-objdump -t user/_lab2.o user/lab2.asm
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_lab1_test user/_lab2 user/_lab3_test
nmeta 4G (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 1954 total 2000
ballof: first 899 blocks have been allocated
ballof: write bitmap block at sector 45
gemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,dri
rive=x0,bus=virtio-mmio-bus.0
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ lab3_test 10 3
Round 1: thread 0 is passing the token to thread 1
Round 2: thread 1 is passing the token to thread 2
Round 3: thread 2 is passing the token to thread 0
Round 4: thread 0 is passing the token to thread 1
Round 5: thread 1 is passing the token to thread 2
Round 6: thread 2 is passing the token to thread 0
Round 7: thread 0 is passing the token to thread 1
Round 8: thread 1 is passing the token to thread 2
Round 9: thread 2 is passing the token to thread 0
Round 10: thread 0 is passing the token to thread 1
Round 11: thread 1 is passing the token to thread 2
Round 12: thread 2 is passing the token to thread 0
Round 13: thread 0 is passing the token to thread 1
Round 14: thread 1 is passing the token to thread 2
Round 15: thread 2 is passing the token to thread 0
Round 16: thread 0 is passing the token to thread 1
Round 17: thread 1 is passing the token to thread 2
Round 18: thread 2 is passing the token to thread 0
Round 19: thread 0 is passing the token to thread 1
Round 20: thread 1 is passing the token to thread 2
Round 21: thread 2 is passing the token to thread 0
Frtsbee simulation has finished, 10 rounds played in total
$ 
```

## Demo for lab3\_test 10 3

```

user1@ubuntu: ~/Downloads/my_github
riscv64-linux-gnu-ld -z max_page_size=4096 -T user/user.ld -o user/_lab3 user/lab3.o user/ulib.o user/usys.o user/printf.o user/umalloc.o user/thread.o
riscv64-linux-gnu-objdump -S user/_lab3 > user/lab3.asm
riscv64-linux-gnu-objdump -t user/_lab3 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab3.sym
riscv64-linux-gnu-objdump -t user/_lab3 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab3_test.sym
riscv64-linux-gnu-objdump -t user/_lab3 | sed '1,/SYMBOL TABLE/d; s/.* /;/; /$&d' > user/lab3_test.o
riscv64-linux-gnu-objdump -t user/_lab3.o user/lab3.o
riscv64-linux-gnu-objdump -t user/_lab3.o user/lab3.asm
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_lab1_test user/_lab2 user/_lab3_test
nmeta 4G (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 1954 total 2000
ballof: first 899 blocks have been allocated
ballof: write bitmap block at sector 45
gemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,dri
rive=x0,bus=virtio-mmio-bus.0
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ lab3_test 21 20
Round 1: thread 0 is passing the token to thread 1
Round 2: thread 1 is passing the token to thread 2
Round 3: thread 2 is passing the token to thread 3
Round 4: thread 3 is passing the token to thread 4
Round 5: thread 4 is passing the token to thread 5
Round 6: thread 5 is passing the token to thread 6
Round 7: thread 6 is passing the token to thread 7
Round 8: thread 7 is passing the token to thread 8
Round 9: thread 8 is passing the token to thread 9
Round 10: thread 9 is passing the token to thread 10
Round 11: thread 10 is passing the token to thread 11
Round 12: thread 11 is passing the token to thread 12
Round 13: thread 12 is passing the token to thread 13
Round 14: thread 13 is passing the token to thread 14
Round 15: thread 14 is passing the token to thread 15
Round 16: thread 15 is passing the token to thread 16
Round 17: thread 16 is passing the token to thread 17
Round 18: thread 17 is passing the token to thread 18
Round 19: thread 18 is passing the token to thread 19
Round 20: thread 19 is passing the token to thread 0
Round 21: thread 0 is passing the token to thread 1
Frtsbee simulation has finished, 21 rounds played in total
$ 
```

## Demo for lab3\_test 21 20

## CONTRIBUTIONS

All the members discussed the implementation, strategy and changes before implementation. Also all members helped each other in case there was an issue in implementation.

Yash worked on part 1 of the lab and implemented the system call structure. Yash also created the basic schema for the user level thread library in part 2.

Nityash implemented and added the code for clone in the defined schemas.

Part 4 implemented and added the code for user level threads in part 2.

## **CODE CHANGES**

[Makefile] - File Compare

File Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

Makefile (Before)

```
68 CFLAGS += -fno-pie -no-pie
69 endif
70 $(eval $(SHELL $S(CC) -dumpspecs 2>/dev/null | grep -e '^[^f]nopie'))
71 CFLAGS += -fno-pie -nopte
72 endif
73
74 # use a variable to track which scheduler to use
75 LAB2 = STRIDE # RR, LOTTERY, STRIDE
76 CFLAGS += -D(LAB2)
77
78 LDFLAGS = -z max-page-size=4096
79
80 $(KERNEL: ${OBJJS} ${KERNEL}.ld ${VMLINUX})
81 ${LD} ${LDFLAGS} -T ${KERNEL}.ld -o ${KERNEL} ${OBJJS}
82 ${OBJDUMP} -S ${KERNEL} > ${KERNEL}.asm
83 ${OBJCOPY} -t ${KERNEL} | sed '1,/SYMBOL TABLE/d; s/ .* /; /*$$/d' > ${KERNEL}.sym
84
85 ${VMLINUX}: ${VMLINUX}.S
86 $(CC) ${CFLAGS} -march=rv64g -nostdinc -I . -I kernel -c ${VMLINUX}.S -o ${VMLINUX}.o
87 ${LD} ${LDFLAGS} -N -e start -Ttext 0 -o ${VMLINUX}.out ${VMLINUX}.o
88 ${OBJCOPY} -S -binary ${VMLINUX}.out ${VMLINUX}.o
89 ${OBJDUMP} -S ${VMLINUX}.o > ${VMLINUX}.asm
90
91 tags: ${OBJJS} _init
92         etags -r *.c
93
94 ${LIB} = ${ULIB}.a ${USYS}.o ${PRINTF}.o ${UMALLOC}.o
95
96 %.o %.${LIB}
97 ${LD} ${LDFLAGS} -T ${USER}.ld -o $@ $^
98 ${OBJDUMP} -S $@ > $*.asm
99 ${OBJCOPY} -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* /; /*$$/d' > $*.sym
100
101 ${USYS}.S : ${USYS}.pl
102     perl ${USYS}.pl > ${USYS}.S
103
104 ${USYS}.o : ${USYS}.S
105     $(CC) ${CFLAGS} -c -o ${USYS}.o ${USYS}.S
106
107 ${_FORTTEST}: ${_FORTTEST}.o ${LIB}
108     # forttest has less library code linked in - needs to be small
109     # in order to be able to max out the proc table.
110     ${LD} ${LDFLAGS} -N -e main -Ttext 0 -o ${LIB}_forttest ${_FORTTEST}.o ${ULIB}.o ${USYS}.o
LAB2 = STRIDE # RR, LOTTERY, STRIDE
LAB2 = RR, LOTTERY, STRIDE
Ready
```

Makefile (Working Tree)

```
68 CFLAGS += -fno-pie -no-pie
69 endif
70 $(eval $(SHELL $S(CC) -dumpspecs 2>/dev/null | grep -e '^[^f]nopie'))
71 CFLAGS += -fno-pie -nopte
72 endif
73
74 # use a variable to track which scheduler to use
75 LAB2 = RR # RR, LOTTERY, STRIDE
76 CFLAGS += -D(LAB2)
77
78 LDFLAGS = -z max-page-size=4096
79
80 $(KERNEL: ${OBJJS} ${KERNEL}.ld ${VMLINUX})
81 ${LD} ${LDFLAGS} -T ${KERNEL}.ld -o ${KERNEL} ${OBJJS}
82 ${OBJDUMP} -S ${KERNEL} > ${KERNEL}.asm
83 ${OBJCOPY} -t ${KERNEL} | sed '1,/SYMBOL TABLE/d; s/ .* /; /*$$/d' > ${KERNEL}.sym
84
85 ${VMLINUX}: ${VMLINUX}.S
86 $(CC) ${CFLAGS} -march=rv64g -nostdinc -I . -I kernel -c ${VMLINUX}.S -o ${VMLINUX}.o
87 ${LD} ${LDFLAGS} -N -e start -Ttext 0 -o ${VMLINUX}.out ${VMLINUX}.o
88 ${OBJCOPY} -S -binary ${VMLINUX}.out ${VMLINUX}.o
89 ${OBJDUMP} -S ${VMLINUX}.o > ${VMLINUX}.asm
90
91 tags: ${OBJJS} _init
92         etags -r *.c
93
94 ${LIB} = ${ULIB}.a ${USYS}.o ${PRINTF}.o ${UMALLOC}.o ${THREAD}.o
95
96 %.o %.${LIB}
97 ${LD} ${LDFLAGS} -T ${USER}.ld -o $@ $^
98 ${OBJDUMP} -S $@ > $*.asm
99 ${OBJCOPY} -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* /; /*$$/d' > $*.sym
100
101 ${USYS}.S : ${USYS}.pl
102     perl ${USYS}.pl > ${USYS}.S
103
104 ${USYS}.o : ${USYS}.S
105     $(CC) ${CFLAGS} -c -o ${USYS}.o ${USYS}.S
106
107 ${_FORTTEST}: ${_FORTTEST}.o ${LIB}
108     # forttest has less library code linked in - needs to be small
109     # in order to be able to max out the proc table.
110     ${LD} ${LDFLAGS} -N -e main -Ttext 0 -o ${LIB}_forttest ${_FORTTEST}.o ${ULIB}.o ${USYS}.o
LAB2 = RR, LOTTERY, STRIDE
LAB2 = RR, LOTTERY, STRIDE
Ready
```

Mar 15 08:22 •

[Makefile] - File Compare

```

File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right

Makerfile (Before)
131 SU_sh
132 SU_stressfs\
133 SU_uertests\
134 SU_grind\
135 SU_wc\
136 SU_zombie\
137 SU_lab1\
138 SU_lab2\
139 SU_lab3\
140 SU_lab4\
141
142 fs.lng: mkfs/mkfs README $(UPROGS)
143 mkfs/mkfs fs.lng README $(UPROGS)
144
145 -include kernel/*.d user/*.d
146
147 clean:
148 rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
149 /*.o /**.d /**.asm /**.sym \
150 SU/initcode.SU/initcode.out SK/kernel fs.lng \
151 mkfs/mkfs .odbinit \
152 SU/sys.S \
153 $(UPROGS)
154
155 # try to generate a unique GDB port
156 GDBPORT = $(shell expr `id -u` % 5000 + 25000)
157 # QEMU's gdb stub command line changed in 0.11
158 QEMUDBG = $(shell if $(QEMU) -help | grep -q '^gdb'; \
159 the echo "-gdb tcp::$(GDBPORT)"; \
160 else echo "-s -p $(GDBPORT); fi)
161 ifndef CPUS
162 # CPUS := 3
163 CPUS := 1
164 endif
165
166 QEMUOPTS = -machine virt-bios none -kernel SK/kernel -m 128M -smp $(CPUS) -nographic
167 QEMUOPTS += -global virtio-mmio.force-legacy=false
168 QEMUOPTS += -drive file=fs.lng,if=none,format=raw,id=x0
169 QEMUOPTS += -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus,0
170
171 qemu: SK/kernel fs.lng
172 $(QEMU) $(QEMUOPTS)
173
174 .gdbinit: .gdbinit.tpl-riscv
175 sed "s/:1234/:$(GDBPORT)/* < $^ > $@"
176

LAB2 = STRIDE # RR, LOTTERY, STRIDE
LAB2 = RR # RR, LOTTERY, STRIDE
Ready

ec input to this VM, move the mouse pointer inside or press Ctrl+G.

```

»

```

File Edit View Go To Window
Save Reload Prev. Change Next Change Take Left Take Right

Makerfile (Working Tree)
134 SU_stressfs\
135 SU_uertests\
136 SU_grind\
137 SU_wc\
138 SU_zombie\
139 SU_lab1\
140 SU_lab2\
141 SU_lab3\
142
143 fs.lng: mkfs/mkfs README $(UPROGS)
144 mkfs/mkfs fs.lng README $(UPROGS)
145
146 -include kernel/*.d user/*.d
147
148 clean:
149 rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
150 /*.o /**.d /**.asm /**.sym \
151 SU/initcode.SU/initcode.out SK/kernel fs.lng \
152 mkfs/mkfs .odbinit \
153 SU/sys.S \
154 $(UPROGS)
155
156 # try to generate a unique GDB port
157 GDBPORT = $(shell expr `id -u` % 5000 + 25000)
158 # QEMU's gdb stub command line changed in 0.11
159 QEMUDBG = $(shell if $(QEMU) -help | grep -q '^gdb'; \
160 the echo "-gdb tcp::$(GDBPORT)"; \
161 else echo "-s -p $(GDBPORT); fi)
162 ifndef CPUS
163 CPUS := 3
164 endif
165
166 QEMUOPTS = -machine virt-bios none -kernel SK/kernel -m 128M -smp $(CPUS) -nographic
167 QEMUOPTS += -global virtio-mmio.force-legacy=false
168 QEMUOPTS += -drive file=fs.lng,if=none,format=raw,id=x0
169 QEMUOPTS += -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus,0
170
171 qemu: SK/kernel fs.lng
172 $(QEMU) $(QEMUOPTS)
173
174 .gdbinit: .gdbinit.tpl-riscv
175 sed "s/:1234/:$(GDBPORT)/* < $^ > $@"
176

75:1 +1-3
ENG 7:52 PM

```

[kernel/proc.c] - File Compare

File Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/proc.c (Before)

```

1 #include "types.h"
2 #include "paran.h"
3 #include "memlayout.h"
4 #include "riscv.h"
5 #include "spinlock.h"
6 #include "proc.h"
7 #include "defs.h"
8
9 struct cpu cpus[NCPU];
10
11 struct proc proc[NPROC];
12
13 struct proc *intproc;
14
15 int nextpid = 1;
16 struct spinlock pid_lock;
17
18 extern void forkret(void);
19 static void freeproc(struct proc *p);
20
21 extern char trampoline[]; // trampoline.S
22
23 // helps ensure that wakeups of wait()ing
24 // parents are not lost, helps obey the
25 // memory model when using p->parent.
26 // must be acquired before any p->lock.
27 struct spinlock wait_lock;
28
29 // pseudo random number generator
30 // https://stackoverflow.com/a/7603688
31 unsigned short lfsr = 0xACE1u;
32 unsigned short bit;
33 unsigned short rand()
34 {
35     bit = ((lfsr > 8) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1;
36     return lfsr = (lfsr >> 1) | (bit << 15);
37 }
38
39 // Allocate a page for each process's kernel stack.
40 // Map it high in memory, followed by an invalid
41 // guard page.
42 void
43 proc_mapstacks(pagetable_t kpgtbl)
44 {
45     int nexttid = 1; // lab3
46
47     Ready
48
49     // to this VM, move the mouse pointer inside or press Ctrl+G.
50 }
```

kernel/proc.c (Working Tree)

```

1 #include "types.h"
2 #include "paran.h"
3 #include "memlayout.h"
4 #include "riscv.h"
5 #include "spinlock.h"
6 #include "proc.h"
7 #include "defs.h"
8
9 struct cpu cpus[NCPU];
10
11 struct proc proc[NPROC];
12
13 struct proc *intproc;
14
15 int nextpid = 1;
16 int nexttid = 1; // lab3
17
18 struct spinlock pid_lock;
19 struct spinlock tid_lock; // lab3
20
21 extern void forkret(void);
22 static void freeproc(struct proc *p);
23
24 extern char trampoline[]; // trampoline.S
25
26 // helps ensure that wakeups of wait()ing
27 // parents are not lost, helps obey the
28 // memory model when using p->parent.
29 // must be acquired before any p->lock.
30 struct spinlock wait_lock;
31
32 // pseudo random number generator
33 // https://stackoverflow.com/a/7603688
34 unsigned short lfsr = 0xACE1u;
35 unsigned short bit;
36 unsigned short rand()
37 {
38     bit = ((lfsr > 8) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1;
39     return lfsr = (lfsr >> 1) | (bit << 15);
40 }
41
42 // Allocate a page for each process's kernel stack.
43 // Map it high in memory, followed by an invalid
44 // guard page.
45 void
46 proc_mapstacks(pagetable_t kpgtbl)
47 {
48     int nexttid = 1; // lab3
49
50     Ready
51
52     // to this VM, move the mouse pointer inside or press Ctrl+G.
53 }
```

Ready

16:1 +7-2

[kernel/proc.c] - File Compare

File Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/proc.c (Before)

```

42 void
43 proc_mapstacks(pagetable_t kpgtbl)
44 {
45     struct proc *p;
46
47     for(p = proc; p < &proc[NPROC]; p++) {
48         char *pa = kalloc();
49         if(pa == 0)
50             panic("kalloc");
51         uint64 va = KSTACK((int)(p - proc));
52         kvmmap(kpgtbl, va, (uint64)pa, PGSIZE, PTE_R | PTE_W);
53     }
54
55     // initialize the proc table.
56     void
57     procinit(void)
58     {
59         struct proc *p;
60
61         initlock(&pid_lock, "nextpid");
62         initlock(&wait_lock, "wait_lock");
63         for(p = proc; p < &proc[NPROC]; p++) {
64             initlock(&p->lock, "proc");
65             p->state = UNUSED;
66             p->kstack = KSTACK((int)(p - proc));
67         }
68     }
69 }
70
71 // Must be called with interrupts disabled,
72 // to prevent race with process being moved
73 // to a different CPU.
74 int
75 cpuid()
76 {
77     int id = r_tp();
78     return id;
79 }
80
81 // Return this CPU's cpu struct.
82 // Interrupts must be disabled.
83 struct cpu*
84 mycpu(void)
85 {
86     struct proc *p;
87
88     initlock(&tid_lock, "nexttid");
89 }
```

kernel/proc.c (Working Tree)

```

40 proc_mapstacks(pagetable_t kpgtbl)
41 {
42     struct proc *p;
43
44     for(p = proc; p < &proc[NPROC]; p++) {
45         char *pa = kalloc();
46         if(pa == 0)
47             panic("kalloc");
48         uint64 va = KSTACK((int)(p - proc));
49         kvmmap(kpgtbl, va, (uint64)pa, PGSIZE, PTE_R | PTE_W);
50     }
51
52     // initialize the proc table.
53     void
54     procinit(void)
55     {
56         struct proc *p;
57
58         initlock(&pid_lock, "nextpid");
59         initlock(&wait_lock, "wait_lock");
60         initlock(&tid_lock, "nexttid");
61     }
62
63     // Must be called with interrupts disabled,
64     // to prevent race with process being moved
65     // to a different CPU.
66     int
67     cpuid()
68     {
69         int id = r_tp();
70         return id;
71     }
72
73     // Return this CPU's cpu struct.
74     // Interrupts must be disabled.
75     struct cpu*
76     mycpu(void)
77     {
78         struct proc *p;
79
80         initlock(&tid_lock, "nexttid");
81     }
82 }
```

Ready

64:1 +7-2

[kernel/proc.c - File Compare]

**kernel/proc.c (Before)**

```

72 myproc(void)
94 {
95     push_off();
96     struct cop *c = mycpu();
97     struct proc *p = c->proc;
98     pop_off();
99     return p;
100 }
101
102 int
103 allocpid()
104 {
105     int pid;
106     ...
107     acquire(&pid_lock);
108     pid = nextpid;
109     nextpid = nextpid + 1;
110     release(&pid_lock);
111 }
112
113 return pid;
114
115 // Look in the process table for an UNUSED proc.
116 // If found, initialize state required to run in the kernel,
117 // and return with p->lock held.
118 // If there are no free procs, or a memory allocation fails, return 0.
119 static struct proc*
120 allocproc(void)
121 {
122     struct proc *p;
123
124     for(p = proc; p < &proc[NPROC]; p++) {
125         acquire(&p->lock);
126         if(p->state == UNUSED) {
127             goto found;
128         } else {
129             release(&p->lock);
130         }
131     }
132     return 0;
133 }
134
135 found:
136     p->pid = allocpid();
137     p->state = USED;
138
139 // Lab3, similar to allocpid()

```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

**kernel/proc.c (Working Tree)**

```

104 J
105
106 int
107 allocpid()
108 {
109     int pid;
110     ...
111     acquire(&pid_lock);
112     pid = nextpid;
113     nextpid = nextpid + 1;
114     release(&pid_lock);
115
116     return pid;
117 }
118
119 // Lab3, similar to allocpid()
120 int
121 alloctid()
122 {
123     int tid;
124
125     acquire(&tid_lock);
126     tid = nexttid;
127     nexttid = nexttid + 1;
128     release(&tid_lock);
129
130     return tid;
131 }
132
133 // Look in the process table for an UNUSED proc.
134 // If found, initialize state required to run in the kernel,
135 // and return with p->lock held.
136 // If there are no free procs, or a memory allocation fails, return 0.
137 static struct proc*
138 allocproc(void)
139 {
140     struct proc *p;
141
142     for(p = proc; p < &proc[NPROC]; p++) {
143         acquire(&p->lock);
144         if(p->state == UNUSED) {
145             goto found;
146         } else {
147             release(&p->lock);
148         }
149     }
150     return 0;
151 }
152
153 found:
154     p->pid = allocpid();
155     p->state = USED;
156     p->thread_id = 0; // lab3
157
158 // Allocate a trapframe page.
159 if(p->trapframe = (struct trapframe *)kalloc()) == 0{
160     freeproc(p);
161     release(&p->lock);
162     return 0;
163 }
164
165 // An empty user page table.
166 p->pagetable = proc_pagetable(p);
167 if(p->pagetable == 0){
168     freeproc(p);
169     release(&p->lock);
170     return 0;
171 }
172
173 // Set up new context to start executing at forkret,
174 // which returns to user space.
175 memset(&p->context, 0, sizeof(p->context));
176 p->context.ra = (uint64)forkret;
177 p->context.sp = p->kstack + PGSIZE;
178
179 p->thread_id = 0; // lab3

```

Ready

[kernel/proc.c - File Compare]

**kernel/proc.c (Before)**

```

112 // Look in the process table for an UNUSED proc.
113 // If found, initialize state required to run in the kernel,
114 // and return with p->lock held.
115 // If there are no free procs, or a memory allocation fails, return 0.
116 static struct proc*
117 allocproc(void)
118 {
119     struct proc *p;
120
121     for(p = proc; p < &proc[NPROC]; p++) {
122         acquire(&p->lock);
123         if(p->state == UNUSED) {
124             goto found;
125         } else {
126             release(&p->lock);
127         }
128     }
129     return 0;
130 }
131
132 found:
133     p->pid = allocpid();
134     p->state = USED;
135
136 // Allocate a trapframe page.
137 if(p->trapframe = (struct trapframe *)kalloc()) == 0{
138     freeproc(p);
139     release(&p->lock);
140     return 0;
141 }
142
143 // An empty user page table.
144 p->pagetable = proc_pagetable(p);
145 if(p->pagetable == 0){
146     freeproc(p);
147     release(&p->lock);
148     return 0;
149 }
150
151 // Set up new context to start executing at forkret,
152 // which returns to user space.
153 memset(&p->context, 0, sizeof(p->context));
154 p->context.ra = (uint64)forkret;
155 p->context.sp = p->kstack + PGSIZE;
156
157 p->thread_id = 0; // lab3

```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

**kernel/proc.c (Working Tree)**

```

132 // If found, initialize state required to run in the kernel,
133 // and return with p->lock held.
134 // If there are no free procs, or a memory allocation fails, return 0.
135 static struct proc*
136 allocproc(void)
137 {
138     struct proc *p;
139
140     for(p = proc; p < &proc[NPROC]; p++) {
141         acquire(&p->lock);
142         if(p->state == UNUSED) {
143             goto found;
144         } else {
145             release(&p->lock);
146         }
147     }
148     return 0;
149 }
150
151 found:
152     p->pid = allocpid();
153     p->state = USED;
154     p->thread_id = 0; // lab3
155
156 // Allocate a trapframe page.
157 if(p->trapframe = (struct trapframe *)kalloc()) == 0{
158     freeproc(p);
159     release(&p->lock);
160     return 0;
161 }
162
163 // An empty user page table.
164 p->pagetable = proc_pagetable(p);
165 if(p->pagetable == 0){
166     freeproc(p);
167     release(&p->lock);
168     return 0;
169 }
170
171 // Set up new context to start executing at forkret,
172 // which returns to user space.
173 memset(&p->context, 0, sizeof(p->context));
174 p->context.ra = (uint64)forkret;
175 p->context.sp = p->kstack + PGSIZE;
176
177 p->thread_id = 0; // lab3

```

Ready

Ubuntu 20.04 • Activities SmartGit • Mar 15 08:24 • [kernel/proc.c] - File Compare

**File** Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/proc.c (Before)

```

140 p->pagecursor = proc_pagecursor(p);
141 if(p->pagetable == 0){
142     freeproc(p);
143     release(&p->lock);
144     return 0;
145 }
153 // Set up new context to start executing at forkret,
154 // which returns to user space.
155 memset(&p->context, 0, sizeof(p->context));
156 p->context.ra = (uint64)forkret;
157 p->context.sp = p->kstack + PGSIZE;
158
159 // initialize the default value of ticks, tickets, strides and pass
160 p->ticks = 0;
161 p->tickets = 0;
162 p->strides = 1;
163 p->pass = p->strides;
164
165 return p;
166 }
167
168 // free a proc structure and the data hanging from it,
169 // including user pages.
170 // p->lock must be held.
171 static void
172 freeproc(struct proc *p)
173 {
174     if(p->trapframe)
175         kfree((void*)p->trapframe);
176     p->trapframe = 0;
177     if(p->pagetable)
178         proc_freepagetable(p->pagetable, p->sz);
179     p->pagetable = 0;
180     p->sz = 0;
181     p->pid = 0;
182     p->parent = 0;
183     p->name[0] = 0;
184     p->chan = 0;
185     p->killed = 0;
186     p->xstate = 0;
187     p->state = UNUSED;
188     // reset sys_calls count to 0
189     p->sys_call_count = 0;
190 }
191
192 // lab3, similar to allocproc()
193 Ready

```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

**File** Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/proc.c (Working Tree)

```

179 p->ticks = 0;
180 p->tickets = 1;
181 p->pass = p->strides;
182
183 return p;
184
185 // lab3, similar to allocproc()
186 static struct proc*
187 allocproc_thread(void)
188 {
189     struct proc *p;
190
191     for(;; p < &proc[NPROC]; p++)
192         acquire(&p->lock);
193         if(p->state == UNUSED) {
194             goto found;
195         } else {
196             release(&p->lock);
197         }
198     }
199
200 return 0;
201
202 found:
203     p->pid = allocpid();
204     p->state = USED;
205     p->thread_id = alloctid(); // lab3
206
207 // Allocate a trapframe page.
208 if((p->trapframe = (struct trapframe *)kalloc()) == 0){
209     freeproc(p);
210     release(&p->lock);
211     return 0;
212 }
213
214 // lab3 comment out this part, share pagetable with parent
215 // An empty user page table.
216 // p->pagetable = proc_pagetable(p);
217 // if(p->pagetable == 0){
218 //     freeproc(p);
219 //     release(&p->lock);
220 //     return 0;
221 // }
222
223
224 // Set up new context to start executing at forkret,
225 // which returns to user space.
226 memset(&p->context, 0, sizeof(p->context));
227 p->context.ra = (uint64)forkret;
228 p->context.sp = p->kstack + PGSIZE;
229
230 return p;
231 }
232
233 // free a proc structure and the data hanging from it,
234 // including user pages.
235 // p->lock must be held.
236 static void
237 freeproc(struct proc *p)
238 {
239     if(p->trapframe)
240         kfree((void*)p->trapframe);
241     p->trapframe = 0;
242     if(p->pagetable)
243         proc_freepagetable(p->pagetable, p->sz);
244     p->pagetable = 0;
245     p->sz = 0;
246 }
247
248 // lab3, similar to allocproc()
249 Ready

```

To input to this VM, move the mouse pointer inside or press Ctrl+G.

Mar 15 08:25 • [kernel/proc.c - File Compare]

```
File Edit View Go To Window
Save Reload Prev Change Next Change Take Left Take Right
kernel/proc.c (Before)
```

```

221
298 // Create a new process, copying the parent.
299 // Sets up child kernel stack to return as if from fork() system call.
300 in
301 fork(void)
302 {
303     int i, pid;
304     struct proc *np;
305     struct proc *p = myproc();
306
307     // Allocate process.
308     if((np = allocproc()) == 0){
309         return -1;
310     }
311
312     // Copy user memory from parent to child.
313     if(unvcopy(p->pagetable, np->pagetable, p->sz) < 0){
314         freeproc(np);
315         release(&np->lock);
316         return -1;
317     }
318     np->sz = p->sz;
319
320     // copy saved user registers.
321     *(np->trapframe) = *(p->trapframe);
322
323     // Cause fork to return 0 in the child.
324     np->trapframe->a0 = 0;
325
326     // increment reference counts on open file descriptors.
327     for(i = 0; i < NOFILE; i++)
328         if(p->ofile[i])
329             np->ofile[i] = filedup(p->ofile[i]);
330     np->cwd = dup(p->cwd);
331
332     safestrncpy(np->name, p->name, sizeof(p->name));
333
334     pid = np->pid;
335
336     release(&np->lock);
337
338     acquire(&wait_lock);
339     np->parent = p;
340     release(&wait_lock);

```

```
kernel/proc.c (Working Tree)
323 // Create a new process, copying the parent.
324 // Sets up child kernel stack to return as if from fork() system call.
325 in
326 fork(void)
327 {
328     int i, pid;
329     struct proc *np;
330     struct proc *p = myproc();
331
332     // Allocate process.
333     if((np = allocproc()) == 0){
334         return -1;
335     }
336
337     // Copy user memory from parent to child.
338     if(unvcopy(p->pagetable, np->pagetable, p->sz) < 0){
339         freeproc(np);
340         release(&np->lock);
341         return -1;
342     }
343     np->sz = p->sz;
344     p->thread_id = 0; // lab3
345
346     // copy saved user registers.
347     *(np->trapframe) = *(p->trapframe);
348
349     // Cause fork to return 0 in the child.
350     np->trapframe->a0 = 0;
351
352     // increment reference counts on open file descriptors.
353     for(i = 0; i < NOFILE; i++)
354         if(p->ofile[i])
355             np->ofile[i] = filedup(p->ofile[i]);
356     np->cwd = dup(p->cwd);
357
358     safestrncpy(np->name, p->name, sizeof(p->name));
359
360     pid = np->pid;
361
362     release(&np->lock);
363
364     acquire(&wait_lock);
365     np->parent = p;
366     release(&wait_lock);

```

Show Applications / lab3

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

Q Search 319:1 +7~2

Mar 15 08:25 • [kernel/proc.c - File Compare]

```
File Edit View Go To Window
Save Reload Prev Change Next Change Take Left Take Right
kernel/proc.c (Before)
```

```

417 acquire(&wait_lock);
418
419 for(;;){
420     // Scan through table looking for exited children.
421     havekids = 0;
422     for(pp = proc; pp < &proc[NPROC]; pp++){
423         if(pp->parent == p){
424             // make sure the child isn't still in exit() or switch().
425             acquire(&pp->lock);
426
427             havekids = 1;
428             if(pp->state == ZOMBIE){
429                 // Found one.
430                 pid = pp->pid;
431                 if(addr != 0 && copyout(pp->pagetable, (char *)addr, &pp->xstate, sizeof(pp->xstate)) < 0) {
432                     release(&pp->lock);
433                     release(&wait_lock);
434                     return -1;
435                 }
436             }
437         }
438     }
439     freeproc(pp);
440     release(&pp->lock);
441     release(&wait_lock);
442     return pid;
443 }
444
445 // No point waiting if we don't have any children.
446 if(havekids || killed(p)){
447     release(&wait_lock);
448     return -1;
449 }
450
451 // Wait for a child to exit.
452 sleep(p, &wait_lock); //DOC: wait-sleep
453 }
454
455 // Per-CPU process scheduler.
456 // Each CPU calls scheduler() after setting itself up.
457
458 // lab3, if it's parent, free all, otherwise, free only thread's resource

```

```
kernel/proc.c (Working Tree)
497     pid = pp->pid;
498     if(addr != 0 && copyout(pp->pagetable, addr, (char *)addr, &pp->xstate, sizeof(pp->xstate)) < 0) {
499         release(&pp->lock);
500         release(&wait_lock);
501         return -1;
502     }
503
504     // lab3, if it's parent, free all, otherwise, free only thread's resource
505     if(pp->thread_id == 0)
506     {
507         // printf("[DEBUG] wait() 1 - pid=%d, tid=%d..%n, pp->pid, pp->thread_id);
508         freeproc(pp);
509     }
510     else
511     {
512         // freeproc for thread
513         // printf("[DEBUG] wait() 2 - pid=%d, tid=%d, pp->pid, pp->thread_id);
514         if(pp->trapframe)
515             kfree((void*)pp->trapframe);
516         pp->trapframe = 0;
517
518         // basically same code as freeproc, except here
519         if(pp->pagetable != 0) {
520             vunmap(pp->pagetable, TRAPFRAME - PGSIZE * (pp->thread_id), 1, 0);
521             pp->thread_id = 0; // lab3: and here
522             pp->pagetable = 0;
523             pp->sz = 0;
524             pp->pid = 0;
525             pp->parent = 0;
526             pp->name[0] = 0;
527             pp->chan = 0;
528             pp->killed = 0;
529             pp->xstate = 0;
530             pp->state = UNUSED;
531         }
532         release(&pp->lock);
533         release(&wait_lock);
534         return pid;
535     }
536     release(&pp->lock);
537 }
538

```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

ENG 7:55 PM 31/3/2023

[kernel/proc.c - File Compare]

```

File Edit View Go To Window
Save Reload Prev.Change Next Change Take Left Take Right
kernel/proc.c (Before)
865 }
866 // print stats as mentioned in lab document
867 int sched_statistics(void)
868 {
869
870 #if defined(LOTTERY)
871 printf("\n -- Lottery Scheduler -- \n\n");
872 #elif defined(STRIDE)
873 printf("\n -- Stride Scheduler -- \n\n");
874 #else
875 printf("\n -- RR Scheduler -- \n\n");
876 #endif
877
878 struct proc *p;
879
880 for (p = proc; p < &proc[NPROC]; p++)
881 {
882 if (p->state != UNUSED)
883 {
884 acquire(&p->lock);
885 printf("%d(%s): tickets: %d, ticks: %d\n", p->pid, p->name, p->tickets, p->ticks);
886 release(&p->lock);
887 }
888 }
889
890 // always return 0 as per lab
891 return 0;
892 }
893
894 // assign tickets to process
895 int sched_tickets(int tickets)
896 {
897 struct proc *p = myproc();
898 p->tickets += tickets;
899 // calculate the value of strides
900 p->strides = LARGE_CONSTANT_K / tickets;
901 p->pass = p->strides;
902 // calculate the value of pass
903 p->pass = p->strides;
904
905 // always return 0 as per lab
906 return 0;
907 }

clone(void *stack)
Ready

```

input to this VM, move the mouse pointer inside or press Ctrl+G.

1016:19 +7-2

[kernel/proc.c - File Compare]

```

File Edit View Go To Window
Save Reload Prev.Change Next Change Take Left Take Right
kernel/proc.c (Before)
865 }
866 // print stats as mentioned in lab document
867 int sched_statistics(void)
868 {
869
870 #if defined(LOTTERY)
871 printf("\n -- Lottery Scheduler -- \n\n");
872 #elif defined(STRIDE)
873 printf("\n -- Stride Scheduler -- \n\n");
874 #else
875 printf("\n -- RR Scheduler -- \n\n");
876 #endif
877
878 struct proc *p;
879
880 for (p = proc; p < &proc[NPROC]; p++)
881 {
882 if (p->state == UNUSED)
883 {
884 acquire(&p->lock);
885 printf("%d(%s): tickets: %d, ticks: %d\n", p->pid, p->name, p->tickets, p->ticks);
886 release(&p->lock);
887 }
888 }
889
890 // always return 0 as per lab
891 return 0;
892 }
893
894 // assign tickets to process
895 int sched_tickets(int tickets)
896 {
897 struct proc *p = myproc();
898 p->tickets += tickets;
899 // calculate the value of strides
900 p->strides = LARGE_CONSTANT_K / tickets;
901 p->pass = p->strides;
902 // calculate the value of pass
903 p->pass = p->strides;
904
905 // always return 0 as per lab
906 return 0;
907 }

clone(void *stack)
Ready

```

input to this VM, move the mouse pointer inside or press Ctrl+G.

1016:19 +7-2

The screenshot shows a dual-monitor setup with two terminal windows side-by-side. Both windows are titled '[kernel/proc.c] - File Compare' and '[kernel/proc.h] - File Compare'. The left monitor displays the 'kernel/proc.c' file, while the right monitor displays the 'kernel/proc.h' file. Each window has a 'Before' state on the left and a 'Working Tree' state on the right, showing the differences between the two versions of the kernel source code. The code is written in C and includes comments and various kernel-specific macros and structures. The terminal windows are part of a desktop environment with a dark theme, and the overall interface is clean and organized.

May 15 08:26

### [kernel/syscall.c] - File Compare

**kernel/syscall.c (Before)**

```

1 //> poor implement of syscalls.h
105 extern uint64 sys_sysinfo(void);
106 // add mapped syscall function here.
107 extern uint64 sys_procinfo(void);
108 // add mapped syscall function here.
109 extern uint64 sys_sched_statistics(void);
110 // add mapped syscall function here.
111 extern uint64 sys_sched_tickets(void);
112
113 // ---- system level variables ----
114
115 // store the total system calls made by the system
116 extern uint64 global_sys_calls_counter;
117
118 // An array mapping syscall numbers from syscall.h
119 // to the function that handles the system call.
120 static uint64 [syscalls] (void) = {
121 [SYS_fork] sys_fork,
122 [SYS_exit] sys_exit,
123 [SYS_wait] sys_wait,
124 [SYS_pipe] sys_pipe,
125 [SYS_read] sys_read,
126 [SYS_kill] sys_kill,
127 [SYS_exec] sys_exec,
128 [SYS_fstat] sys_fstat,
129 [SYS_chdir] sys_chdir,
130 [SYS_dup] sys_dup,
131 [SYS_getpid] sys_getpid,
132 [SYS_sbrk] sys_sbrk,
133 [SYS_setSIGSIG] sys_setSIGSIG,
134 [SYS_getSIGSIG] sys_getSIGSIG,
135 [SYS_open] sys_open,
136 [SYS_write] sys_write,
137 [SYS_mknod] sys_mknod,
138 [SYS_unlink] sys_unlink,
139 [SYS_link] sys_link,
140 [SYS_chattr] sys_chattr,
141 [SYS_close] sys_close,
142 [SYS_sysinfo] sys_sysinfo, // map the defined syscall to a user function.
143 [SYS_procinfo] sys_procinfo, // map the defined syscall to a user function.
144 [SYS_sched_statistics] sys_sched_statistics, // map the defined syscall to a user function.
145 [SYS_sched_tickets] sys_sched_tickets, // map the defined syscall to a user function.
146 };
147
// add mapped syscall function here.

```

**kernel/syscall.c (Working Tree)**

```

1 //> poor implement of syscalls.h
107 extern uint64 sys_procinfo(void);
108 // add mapped syscall function here.
109 extern uint64 sys_sched_statistics(void);
110 // add mapped syscall function here.
111 extern uint64 sys_sched_tickets(void);
112 // add mapped syscall function here.
113 extern uint64 sys_clone(void);
114
115 // ---- system level variables ----
116
117 // store the total system calls made by the system
118 extern uint64 global_sys_calls_counter;
119
120 // An array mapping syscall numbers from syscall.h
121 // to the function that handles the system call.
122 static uint64 [syscalls] (void) = {
123 [SYS_fork] sys_fork,
124 [SYS_exit] sys_exit,
125 [SYS_wait] sys_wait,
126 [SYS_pipe] sys_pipe,
127 [SYS_read] sys_read,
128 [SYS_kill] sys_kill,
129 [SYS_exec] sys_exec,
130 [SYS_fstat] sys_fstat,
131 [SYS_chdir] sys_chdir,
132 [SYS_dup] sys_dup,
133 [SYS_getpid] sys_getpid,
134 [SYS_sbrk] sys_sbrk,
135 [SYS_setSIGSIG] sys_setSIGSIG,
136 [SYS_getSIGSIG] sys_getSIGSIG,
137 [SYS_open] sys_open,
138 [SYS_write] sys_write,
139 [SYS_mknod] sys_mknod,
140 [SYS_unlink] sys_unlink,
141 [SYS_link] sys_link,
142 [SYS_chattr] sys_chattr,
143 [SYS_close] sys_close,
144 [SYS_sysinfo] sys_sysinfo, // map the defined syscall to a user function.
145 [SYS_procinfo] sys_procinfo, // map the defined syscall to a user function.
146 [SYS_sched_statistics] sys_sched_statistics, // map the defined syscall to a user function.
147 [SYS_sched_tickets] sys_sched_tickets, // map the defined syscall to a user function.
148 [SYS_clone] sys_clone, // map the defined syscall to a user function.
149 };
150

```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

May 15 08:26

### [kernel/syscall.h] - File Compare

**kernel/syscall.h (Before)**

```

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_write 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_setSIGSIG 13
15 #define SYS_getSIGSIG 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_chattr 20
22 #define SYS_close 21
23 // define the syscall here, assign it a number and define the corresponding function in syscall.c
24 #define SYS_sysinfo 22
25 // define the syscall here, assign it a number and define the corresponding function in syscall.c
26 #define SYS_procinfo 23
27 // define the syscall here, assign it a number and define the corresponding function in syscall.c
28 #define SYS_sched_statistics 24
29 // define the syscall here, assign it a number and define the corresponding function in syscall.c
30 #define SYS_sched_tickets 25

```

**kernel/syscall.h (Working Tree)**

```

1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_write 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_setSIGSIG 13
15 #define SYS_getSIGSIG 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_chattr 20
22 #define SYS_close 21
23 // define the syscall here, assign it a number and define the corresponding function in syscall.c
24 #define SYS_sysinfo 22
25 // define the syscall here, assign it a number and define the corresponding function in syscall.c
26 #define SYS_procinfo 23
27 // define the syscall here, assign it a number and define the corresponding function in syscall.c
28 #define SYS_sched_statistics 24
29 // define the syscall here, assign it a number and define the corresponding function in syscall.c
30 #define SYS_sched_tickets 25
31 // define the syscall here, assign it a number and define the corresponding function in syscall.c
32 #define SYS_clone 26
33

```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

[kernel/sysproc.c] - File Compare

File Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/sysproc.c (Before)

```

106     return get_sys_sysinfo(n, global_sys_calls_counter);
107 }
108 // body of defined syscall function.
109 // call a function in process to execute0.
110 // get first argument passed to the system call
111 // pass the argument to process level function
112 uint64
113 sys_procinfo(void)
114 {
115     uint64 pinfo_pointer; // user pointer to struct pinfo
116     argaddr(0, &pinfo_pointer);
117     return get_sys_procinfo(pinfo_pointer);
118 }
119 }

120 // print the process statistics
121 uint64
122 sys_sched_statistics(void)
123 {
124     return sched_statistics();
125 }
126 }

127 // set tickets to a process. The max set value can only be 10000 as per lab..
128 // The number of tickets can not be negative.
129 uint64
130 sys_sched_tickets(void)
131 {
132     int tickets;
133     uint64 atickets;
134     if (tickets > MAX_TICKETS)
135     {
136         tickets = MAX_TICKETS;
137     }
138     else if (tickets <= 0)
139     {
140         tickets = 0;
141     }
142     else
143     {
144         // do nothing
145     }
146     return sched_tickets(tickets);
147 }

148 }
```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

kernel/sysproc.c (Working Tree)

```

114 sys_procinfo(void)
115 {
116     uint64 pinfo_pointer; // user pointer to struct pinfo
117     argaddr(0, &pinfo_pointer);
118     return get_sys_procinfo(pinfo_pointer);
119 }
120

121 // print the process statistics
122 uint64
123 sys_sched_statistics(void)
124 {
125     return sched_statistics();
126 }

127 // set tickets to a process. The max set value can only be 10000 as per lab..
128 // The number of tickets can not be negative.
129 uint64
130 sys_sched_tickets(void)
131 {
132     int tickets;
133     uint64 atickets;
134     if (tickets > MAX_TICKETS)
135     {
136         tickets = MAX_TICKETS;
137     }
138     else if (tickets <= 0)
139     {
140         tickets = 0;
141     }
142     else
143     {
144         // do nothing
145     }
146     return sched_tickets(tickets);
147 }

148 // Implement the clone system call
149 uint64
150 sys_clone(void)
151 {
152     uint64 addr;
153     argaddr(0, &addr);
154     return clone((void *)addr);
155 }
```

148:1 +1

[kernel/trap.c] - File Compare

File Edit View Go To Window

Save Reload Prev.Change Next Change Take Left Take Right

kernel/trap.c (Before)

```

108 p->trapframe->kernel_hartid = r_tp; // hartid for cpuid()
109 // set up the registers that trampoline.S's sret will use
110 // to get to user space.
111 ...
112 ...
113 // set S Previous Privilege mode to User.
114 unsigned long x = r_sstatus();
115 x &= ~STATUS_SPP; // clear SPP to 0 for user mode
116 x |= SSTATUS_SPIE; // enable Interrupts in user mode
117 w_sstatus(x);
118

119 // set S Exception Program Counter to the saved user pc.
120 w_sepc(p->trapframe->epc);
121
122 // tell trampoline.S the user page table to switch to.
123 uint64 satp = MAKE_SATP(p->pageable);
124
125 // jump to userret in trampoline.S at the top of memory, which-
126 // switches to the user page table, restores user registers,
127 // and switches to user mode with sret.
128 uint64 trampoline_userret = TRAPOLINE + (userret - trampoline);
129 ((void *)(uint64))trampoline_userret(satp);
130 }

131 // interrupts and exceptions from kernel code go here via kernelvec,
132 // on whatever the current kernel stack is.
133 void
134 kerneltrap()
135 {
136     int which_dev = 0;
137     uint64 sepc = r_sepc();
138     uint64 sstatus = r_sstatus();
139     uint64 scause = r_scause();
140     ...
141
142     if((rstatus & SSTATUS_SPP) == 0)
143     {
144         panic("kerneltrap: not from supervisor mode");
145     }
146     if(intr_get() != 0)
147     {
148         panic("kerneltrap: interrupts enabled");
149     }
150
151     if(which_dev == devintr()) == 0)
152     {
153         printf("cause %08x\n", scause);
154         printf("sepc=%08x stval=%08x", r_sepc(), r_stval());
155         panic("kerneltrap");
156     }
157
158 // ((void *)(uint64))trampoline_userret(satp); //original code
159 }
```

Ready

Input to this VM, move the mouse pointer inside or press Ctrl+G.

kernel/trap.c (Working Tree)

```

108 p->trapframe->kernel_hartid = r_tp; // hartid for cpuid()
109
110 // set up the registers that trampoline.S's sret will use
111 // to get to user space.
112 ...
113
114 // set S Previous Privilege mode to User.
115 unsigned long x = r_sstatus();
116 x &= ~STATUS_SPP; // clear SPP to 0 for user mode
117 x |= SSTATUS_SPIE; // enable interrupts in user mode
118 w_sstatus(x);
119
120 // set S Exception Program Counter to the saved user pc.
121 w_sepc(p->trapframe->epc);
122
123 // tell trampoline.S the user page table to switch to.
124 uint64 satp = MAKE_SATP(p->pageable);
125
126 // jump to userret in trampoline.S at the top of memory, which-
127 // switches to the user page table, restores user registers,
128 // and switches to user mode with sret.
129 uint64 trampoline_userret = TRAPOLINE + (userret - trampoline);
130 ((void *)(uint64))trampoline_userret(satp); //original code
131
132 // changes for lab trap.c
133 ((void *)(uint64))trampoline_userret(TRAPFRAME - PGSIZE * p->thread_id, satp);
134
135 // interrupts and exceptions from kernel code go here via kernelvec,
136 // on whatever the current kernel stack is.
137 void
138 kerneltrap()
139 {
140     int which_dev = 0;
141     uint64 sepc = r_sepc();
142     uint64 sstatus = r_sstatus();
143     uint64 scause = r_scause();
144 ...
145     if((rstatus & SSTATUS_SPP) == 0)
146     {
147         panic("kerneltrap: not from supervisor mode");
148     }
149     if(intr_get() != 0)
150     {
151         panic("kerneltrap: interrupts enabled");
152     }
153     if(which_dev == devintr()) == 0{
154 }
```

129:1 ~1

[user/user.h] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

user/user.h (Before)

```
12 int close(int);
13 int dup(int);
14 int exec(const char*, char**);
15 int open(const char*, int);
16 int mknod(const char*, short, short);
17 int unlink(const char*);
18 int fstat(int fd, struct stat*);
19 int link(const char*, const char*);
20 int chmod(const char*);
21 int chdir(const char*);
22 int dup(int);
23 int getpid(void);
24 char* sbrk(int);
25 int sleep(int);
26 int uptime(void);
27 ...
28 // sysproc.c
29 // define prototype of sysinfo syscall
30 int sysinfo(int);
31 // define prototype of procinfo syscall
32 int procinfo(struct pinfo*);
33 // define prototype of sched_statistics syscall
34 int sched_statistics(void);
35 // define prototype of sched_tickets syscall
36 int sched_tickets(int);
37 ...
38 // ulib.c
39 int stat(const char*, struct stat*);
40 char* strcpy(char*, const char*);
41 void* malloc(void*, uint);
42 char* strchr(const char*, char c);
43 int strcmp(const char*, const char*);
44 void fprintf(int, const char*, ...);
45 void printf(const char*, ...);
46 char* gets(char*, int max);
47 uint strlen(const char*);
48 void* realloc(void*, uint, uint);
49 void* malloc(uint);
50 void free(void*);
51 int atoi(const char*);
52 int memcmp(const void*, const void*, uint);
53 void *memcpy(void*, const void*, uint);
54 ...
55 int clone(void *stack);
```

Ready

put to this VM, move the mouse pointer inside or press Ctrl+G.

54:1 -1

[user/usys.pl] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

user/usys.pl (Before)

```
6
7 print "#include <kernel/syscall.h>\n";
8
9 sub entry {
10     my $name = shift;
11     print "my $name = shift;\n";
12     print "$name:\n";
13     print "    l1 a7, SYS_$name\n";
14     print "    ecall\n";
15     print "    retw\n";
16 }
17
18 entry("fork");
19 entry("exit");
20 entry("wait");
21 entry("pipe");
22 entry("read");
23 entry("write");
24 entry("close");
25 entry("null");
26 entry("exec");
27 entry("open");
28 entry("mknod");
29 entry("unlink");
30 entry("fstat");
31 entry("link");
32 entry("dup");
33 entry("chdir");
34 entry("dup");
35 entry("getpid");
36 entry("sbrk");
37 entry("sleep");
38 entry("uptime");
39
40 # add an entry for sysinfo syscall here
41 entry("sysinfo");
42 # add an entry for procinfo syscall here
43 entry("procinfo");
44 # add an entry for sched_ticks syscall here
45 entry("sched_ticks");
46 # add an entry for sched_statistics syscall here
47 entry("sched_statistics");
48 ...
49 # add an entry for clone syscall here
```

Ready

put to this VM, move the mouse pointer inside or press Ctrl+G.

5:1 ENG 7:58 PM

[user/user.h] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

user/user.h (Working Tree)

```
12 int close(int);
13 int kill();
14 int exec(const char*, char**);
15 int open(const char*, int);
16 int mknod(const char*, short, short);
17 int unlink(const char*);
18 int fstat(int fd, struct stat*);
19 int link(const char*, const char*);
20 int chmod(const char*);
21 int chdir(const char*);
22 int dup(int);
23 int getpid(void);
24 char* sbrk(int);
25 int sleep(int);
26 int uptime(void);
27 ...
28 // sysproc.c
29 // define prototype of sysinfo syscall
30 int sysinfo(int);
31 // define prototype of procinfo syscall
32 int procinfo(struct pinfo*);
33 // define prototype of sched_statistics syscall
34 int sched_statistics(void);
35 // define prototype of sched_ticks syscall
36 int sched_ticks(int);
37 ...
38 // ulib.c
39 int stat(const char*, struct stat*);
40 char* strcpy(char*, const char*);
41 void* malloc(void*, uint);
42 char* strchr(const char*, char c);
43 int strcmp(const char*, const char*);
44 void fprintf(int, const char*, ...);
45 void printf(const char*, ...);
46 char* gets(char*, int max);
47 uint strlen(const char*);
48 void* realloc(void*, uint, uint);
49 void* malloc(uint);
50 void free(void*);
51 int atoi(const char*);
52 int memcmp(const void*, const void*, uint);
53 void *memcpy(void*, const void*, uint);
54 int clone(void *stack);
```

Ready

put to this VM, move the mouse pointer inside or press Ctrl+G.

54:1 -1

[user/usys.pl] - File Compare

File Edit View Go To Window

Save Reload Prev Change Next Change Take Left Take Right

user/usys.pl (Working Tree)

```
8
9 sub entry {
10     my $name = shift;
11     print ".global $name\n";
12     print "$name:\n";
13     print "    l1 a7, sys_$name\n";
14     print "    ecall\n";
15     print "    ret\n";
16 }
17
18 entry("fork");
19 entry("exit");
20 entry("wait");
21 entry("pipe");
22 entry("read");
23 entry("write");
24 entry("close");
25 entry("kill");
26 entry("exec");
27 entry("open");
28 entry("mknod");
29 entry("unlink");
30 entry("fstat");
31 entry("link");
32 entry("mkdir");
33 entry("chdir");
34 entry("dup");
35 entry("dup2");
36 entry("dup3");
37 entry("sbrk");
38 entry("sleep");
39 entry("uptime");
40
41 # add an entry for sysinfo syscall here
42 entry("sysinfo");
43 # add an entry for procinfo syscall here
44 entry("procinfo");
45 # add an entry for sched_ticks syscall here
46 entry("sched_ticks");
47 # add an entry for sched_statistics syscall here
48 entry("sched_statistics");
49 # add an entry for clone syscall here
50 entry("clone");
```

Ready

put to this VM, move the mouse pointer inside or press Ctrl+G.

5:1 ENG 7:58 PM

SmartGit

File Edit View Go To Window

Save Reload Prev. Change Next Change Take Left Take Right

[user/thread.h] - File Compare

user/thread.h (Working Tree)

```
1 // thread.h
2 typedef struct {
3     uint locked;
4 } lock_t;
5
6 int thread_create(void *(start_routine)(void*), void *arg);
7 void lock_init(lock_t* lock);
8 void lock_acquire(lock_t* lock);
9 void lock_release(lock_t* lock);
10
```

SmartGit

File Edit View Go To Window

Save Reload Prev. Change Next Change Take Left Take Right

[user/thread.c] - File Compare

user/thread.c (Working Tree)

```
#include "kernel/types.h"
1 #include "kernel/stat.h"
2 #include "kernel/param.h"
3 #include "kernel/riscv.h"
4 #include "user/user.h"
5 #include "user/thread.h"
6
7 int thread_create(void *(start_routine)(void*), void *arg) {
8     // printf("[DEBUG] thread_create() - start...\n");
9     void *stack;
10    int pid;
11
12    lock_t lk;
13    lock_init(&lk);
14    lock_acquire(&lk);
15    // Set up new threads stack
16    stack = malloc( PGSIZE * sizeof(void*));
17
18    if (stack == 0) return -1;
19    pid = clone((void*)stack);
20
21    if (pid == 0) {
22        // pid == 0, run 4 time, but didn't call start_routine()
23        // otherwise, it run 1 time, but call
24        if (pid == 0){
25            // printf("[DEBUG] thread_create() - after clone, before start_routine...\n");
26            start_routine(arg);
27            // printf("[DEBUG] thread_create() - after start_routine...\n");
28            free(stack);
29            // printf("[DEBUG] thread_create() - after free...\n");
30            exit(0);
31        }
32        lock_release(&lk);
33
34        // printf("[DEBUG] thread_create() - ok...\n");
35        return pid;
36    }
37
38
39
40    void lock_init(lock_t* lock) {
41        lock->locked = 0;
42    }
43 }
```

Ubuntu 20.04 • Mar 15 08:29 • [user/thread.c] - File Compare

File Edit View Go To Window Save Reload Prev.Change Next Change Take Left Take Right user/thread.c (Before)

```
user/thread.c (Working Tree)
1
14 lock_init(&lk);
15 lk_acqut(&lk);
16 // Set up new thread stack
17 stack = malloc( PGSIZE * sizeof(void*));
18
19 if (stack == 0) return -1;
20 pid = clone((void*)stack,
21
22 // pid == 0, run 4 time, but didn't call start_routine()
23 // otherwise, it run 1 time, but call
24 if (pid == 0){
25     // printf("[DEBUG] thread_create() - after clone, before start_routine...\n");
26     start_routine();
27     // printf("[DEBUG] thread_create() - after start_routine...\n");
28     free(stack);
29     // printf("[DEBUG] thread_create() - after free...\n");
30     exit(0);
31 }
32 lock_release(&lk);
33
34 // printf("[DEBUG] thread_create() - ok...\n");
35
36 return pid;
37
38
39
40 void lock_init(lock_t* lock) {
41     lock->locked = 0;
42 }
43
44 void lock_acquire(lock_t* lock) {
45     // same as spinlock.c line 32, 39
46     while(!_sync_lock_test_and_set(&lock->locked, 1) != 0);
47     __sync_synchronize();
48 }
49
50 void lock_release(lock_t* lock) {
51     // same as spinlock.c line 68, 69
52     __sync_synchronize();
53     __sync_lock_release(&lock->locked);
54 }
55 }
```

#include "kernel/types.h"  
Ready

Activities SmartGit • Mar 15 08:30 • [user/lab3\_test.c] - File Compare

File Edit View Go To Window Save Reload Prev.Change Next Change Take Left Take Right user/lab3\_test.c (Before)

```
user/lab3_test.c (Working Tree)
1
1 #include "kernel/types.h"
2 #include "kernel/stat.h"
3 #include "user/user.h"
4 #include "user/thread.h"
5 lock_t lock;
6 int n_threads, n_passes, cur_turn, cur_pass;
7 void* thread_fn(void* arg)
8 {
9     int thread_id = (uint64)arg;
10    int done = 0;
11    while(!done) {
12        lock_acquire(&lock);
13        if (cur_pass >= n_passes) done = 1;
14        else if (cur_turn == thread_id) {
15            cur_turn = (cur_turn + 1) % n_threads;
16            printf("Round %d: thread %d is passing the token to thread %d\n",
17                  ++cur_pass, thread_id, cur_turn);
18        }
19        lock_release(&lock);
20        sleep(0);
21    }
22    return 0;
23 }
24 int main(int argc, char *argv[])
25 {
26     if (argc < 3) {
27         printf("Usage: %s [N_PASSES] [N_THREADS]\n", argv[0]); exit(-1);
28     }
29     n_passes = atoi(argv[1]);
30     n_threads = atoi(argv[2]);
31     cur_turn = 0;
32     cur_pass = 0;
33     lock_init(&lock);
34     for (int i = 0; i < n_threads; i++) {
35         // printf("[DEBUG] lab3] i=%d\n", i);
36         thread_create(thread_fn, (void*)(uint64)i);
37     }
38     for (int i = 0; i < n_threads; i++) {
39         walt(0);
40     }
41     printf("Frisbee simulation has finished, %d rounds played in total\n", n_passes);
42     exit(0);
43 }
```

#include "kernel/types.h"  
Ready