Shweti Mahajan Vishnu Priya Chandrasekar
February 12, 2017

OS Lab 1: System Calls and Schedulers

Part 1: Add a system call info(int param) to return number of processes running and number of system calls made.

List of files modified and functions added:

1. usys.S - Add line SYSCALL(info)
2. user.h - Add line int info(int);
3. syscall.h - Add line #define SYS_info 22
4. syscall.c - Add lines extern int sys_info(void) and [SYS_info] sys_info and in syscall method proc->call_count++;
5. sysproc.c - Add snippet

```
int sys_info(void){
 int n;
 if( argint(0, &n) < 0)
   return -1;
 return info(n);
}
```

6. proc.c - Add the system call logic

```
int info(int param){
 int count = 0;
 struct proc *p;
 switch(param){
 case 1:
   for(p = ptable.proc; p<&ptable.proc[NPROC]; p++){
```

```
            if(!(p->state == UNUSED || p->state == ZOMBIE))
              count += 1;
           }
          cprintf("No of running processes: %d\n", count);
          break;
      case 2:
        cprintf("No of syscalls made by the process: %d\n", proc->call_count);
        break;
      case 3:
        cprintf("No of memory pages used: %d\n", proc->page_count);
        break;
    }
    return count;
}
```

In allocproc method add

```
p-> pages_count = 0;
before kalloc is called
p>page_count++;
p->call_count = 0;
```

In fork method, when kfree is called

```
add if(np->page_count){
      np-> page_count - -;
      }
```

In wait method add

```
if(p->page_count){
      p-> page_count - -;
}
```

7. proc.h - Add line in struct proc

```
      int call_count;
      int page_count;
```

8. def.h - Add line int info(int);
9. test.c - Test function to check the system call

```
#include   "types.h"
#include   "stat.h"
#include   "user.h"
int  main ( int  argc ,   char   * argv []) {
    info(1);
    info(2);
    info(3);
    exit();
}
```

10. Makefile - Add line _test\

Results :

```
ster -fno-stack-protector   -c -o proc.o proc.c
ld -m    elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.o
 kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o swtch.o syscall.o s
ysfile.o sysproc.o timer.o trapasm.o trap.o uart.o vectors.o vm.o  -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB) copied, 0.0311333 s, 164 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000286396 s, 2.2 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
335+1 records in
335+1 records out
171847 bytes (172 kB) copied, 0.0011408l s, 151 MB/s
qemu -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,for
mat=raw -smp 2 -m 512
Could not open option rom 'sgabios.bin': No such file or directory
xv6...
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test
No of running processes: 3
No of syscalls made by the process: 4
No of memory pages used: 1
$
```

PART 2 :

1. Lottery Scheduling :

List of files that were modified :

- usys.h
- user.h
- syscall.h
- syscall.c
- sysproc.c
- proc.c
- defs.h

Snippets of the changes made in proc.c are as follows :

```
//our code for set_tickets
int
set_tickets(int tickets)
{
        total_tickets += tickets;
        proc->tickets = tickets;
        cprintf("ticket value set to :%d\n",proc->tickets);
        return 2;
}
```

System call to set the value of tickets

The lottery scheduler code :

```
void
scheduler(void)
{
  struct proc *p;
  unsigned long winner;
  int counter;
  int sd=0;

  for(;;){
    // Enable interrupts on this processor.
    sti();
    sd++;
    //OS RNG
    //Random number generator
    //Having lottery

    winner=rnum_gen(sd)%(total_tickets+1);

    //cprintf("Winner is : %d\n",winner);
    counter = 0;
    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      //cprintf("Inside: %d", p->state);
      if(p->state != RUNNABLE)
        continue;
      //cprintf("Counter: %d, Tickets: %d", counter, p->tickets);
      counter =counter + p->tickets;
      if(counter <  winner)
        continue;
```

```
      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      //cprintf("pid : ",p->pid);
      p->count +=1;
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&cpu->scheduler, p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      proc = 0;
    }
    release(&ptable.lock);

  }
}
```

The random number generator :

```
/*
unsigned long
rndm_gen(int sd)
{
        return (sd*2754702730UL)%4294967291UL;
}
```

Changes made in allocproc() function :

```
// Set up new context to start executing at forkret,
// which returns to trapret.
sp -= 4;
*(uint*)sp = (uint)trapret;

sp -= sizeof *p->context;
p->context = (struct context*)sp;
memset(p->context, 0, sizeof *p->context);
p->context->eip = (uint)forkret;
p->tickets = 1;
p->count = 0;
p->pass = 1;
//total_tickets += 1;
return p;
}
```

The variable tickets is defined in struct proc and given a default value in allocproc(). Also, the variable count, which calculates the number of counts a process is scheduled to run by the scheduler is given a default value of 0.

Changes made in sysproc.c :

```
//Our system call
int
sys_set_tickets(void)
{
        int n;
        if(argint(0,&n)<0)
           return -1;
        //if(argint(1, &pid)<0)
        //   return -1;

        return set_tickets(n);
}
```

Results obtained on running processes with 3 tickets :

```
$ test&;test2&;test1
number of runs of sh is  2
number of runs of sh is  2
ticket value set to :20
number of runs of test1 is  8
$ ticket value set to :30
number of runs of test is  6
zombie!
ticket value set to :10
number of runs of test2 is  6
zombie!
```

2. Stride Scheduler :

List of files that were modified :

- usys.h
- user.h
- syscall.h
- syscall.c
- sysproc.c
- proc.c
- defs.h

Snippets of the changes made in proc.c are as follows :

```
void
scheduler(void)
{
  struct proc *p;
  int min_pass;
  int w_pid=0;

  for(;;){
    min_pass = 10000;
    // Enable interrupts on this processor.
    sti();
    //OS 202
    // Loop over process table looking for process to run.
    acquire(&ptable.lock);

    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
       //cprintf("Inside: %d", p->state);
      if(p->state != RUNNABLE)
        continue;
      if(p->pass < min_pass){
        min_pass = p->pass;
        w_pid = p->pid;
      }
    }
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
       //cprintf("Inside: %d", p->state);
      if(p->state != RUNNABLE)
        continue;
      if(p->pid != w_pid)
        continue;
```

```
      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      //cprintf("pid : ",p->pid);
      p->ccurt +=1;
      p->pass += (10000/p->tickets);
      proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&cpu->scheduler, p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      proc = 0;
    }
    release(&ptable.lock);

  }
}
```

Results obtained :

```
$ test&;test1&;test2&
number of runs of sh is  23
number of runs of sh is  2
ticket value set to :30
number of runs of test is  5
zombie!
ticket value set to :20
number of runs of test1 is  4
number of runs of sh is  9
zombie!
$ ticket value set to :10
number of runs of test2 is  11
zombie!

number of runs of sh is  1
```