CA3 - Amay Dixit

Step 1:

in proc.h

```
// Per-process state
struct proc {
 uint sz;                     // Size of process memory (bytes)
 pde_t* pgdir;                // Page table
 char *kstack;                // Bottom of kernel stack for this process
 enum procstate state;        // Process state
 int pid;                     // Process ID
 struct proc *parent;         // Parent process
 struct trapframe *tf;        // Trap frame for current syscall
 struct context *context;     // swtch() here to run process
 void *chan;                  // If non-zero, sleeping on chan
 int killed;                  // If non-zero, have been killed
 struct file *ofile[NOFILE];  // Open files
 struct inode *cwd;           // Current directory
 char name[16];               // Process name (debugging)
 int userflag;
 int sched_count; // added this
 int run_ticks; // added this
};
```

Step 2:

In proc.c

```
 p->context->eip = (uint)forkret;
 p->sched_count = 0; // added this
 p->run_ticks = 0; // added this

 return p;
}
```

Step 3: scheduler function in proc.c

```
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state != RUNNABLE)
      continue;
    // Switch to chosen process.  It is the process's job
    // to release ptable.lock and then reacquire it
    // before jumping back to us.
    c->proc = p;
    switchuvm(p);
```

```
    p->state = RUNNING;

    p->sched_count++; // added this

    swtch(&(c->scheduler), p->context);
    switchkvm();
```

Step 4: trap functoin in trap.c

```
case T_IRQ0 + IRQ_TIMER:

  // added step 4 here
  if (myproc())
    if (myproc()->state == RUNNING) myproc()->run_ticks++;


  if(cpuid() == 0){
    acquire(&tickslock);
    ticks++;
    wakeup(&ticks);
    release(&tickslock);
  }
  lapiceoi();
  break;
```

Step 5:

In syscall.h

```
#define SYS_getstats 25
```

In syscall.c

```
extern int sys_getstats(void);
```

```
[SYS_getstats] sys_getstats,
```

Step 6:

In sysproc.c

```
int
sys_getstats(void) {
 int *user_stats_ptr;

 if (argptr(0, (void**)&user_stats_ptr, 2*sizeof(int)) < 0)
   return -1;

 struct proc *p = myproc();
```

```
 int kernel_stats[2];
  kernel_stats[0] = p->sched_count;
 kernel_stats[1] = p->run_ticks;

 if (copyout(p->pgdir, (uint)user_stats_ptr, (char*)kernel_stats,
sizeof(kernel_stats)) < 0)
   return -1;

 return 0;
}
```

Step 7 & 8:

In user.h

```
struct procstats {
   int pid;
   int run_ticks;
   int sched_count;
   char name[16];
};
int getstats ( int * stats_array ) ;
```

In usys.S

```
SYSCALL(getstats)
```

Step 9:

In statstest.h

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main(void)
{
   int stats[2];
   int i;
   for (i = 0; i < 2; i++)
   {
       if (<call the system call> == 0){
           printf(1, " Scheduled %d times , ran for %d ticks\n", stats[0],
stats[1]);
       }
       else{
           printf(2, " getstats failed\n");
       }
```

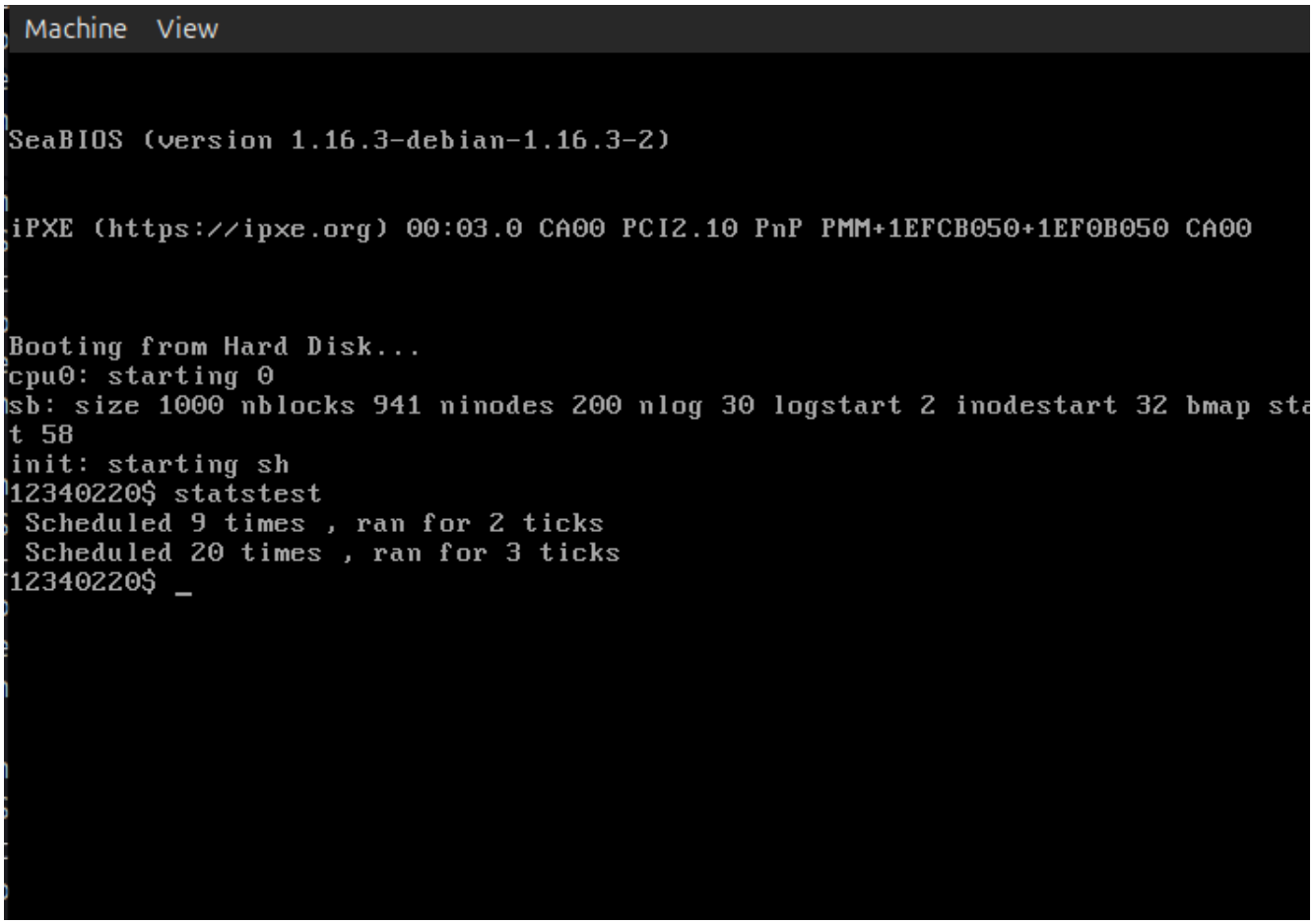
```
        sleep(10);
    }
    exit();
}
```

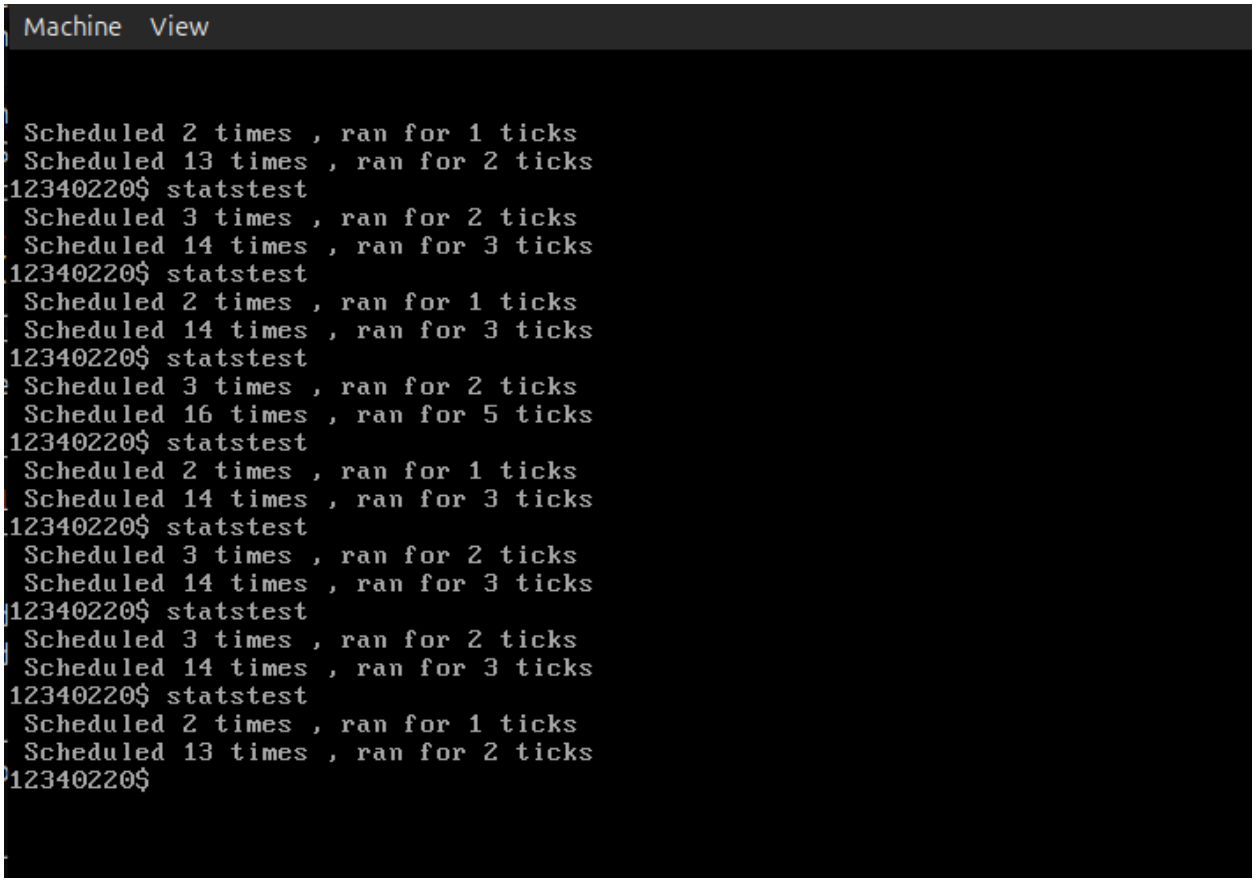
Step 10:

In Makefile

```
_statstest\
```

Final Output:




Machine View
Scheduled 2 times , ran for 1 ticks
Scheduled 13 times , ran for 2 ticks
12340220$ statstest
Scheduled 3 times , ran for 2 ticks
Scheduled 14 times , ran for 3 ticks
12340220$ statstest
Scheduled 2 times , ran for 1 ticks
Scheduled 14 times , ran for 3 ticks
12340220$ statstest
Scheduled 3 times , ran for 2 ticks
Scheduled 16 times , ran for 5 ticks
12340220$ statstest
Scheduled 2 times , ran for 1 ticks
Scheduled 14 times , ran for 3 ticks
12340220$ statstest
Scheduled 3 times , ran for 2 ticks
Scheduled 14 times , ran for 3 ticks
12340220$ statstest
Scheduled 3 times , ran for 2 ticks
Scheduled 14 times , ran for 3 ticks
12340220$ statstest
Scheduled 2 times , ran for 1 ticks
Scheduled 13 times , ran for 2 ticks
12340220$