Code Modification

1. Compilation Test

```
### Makefile
(-) CS333_PROJECT ?= 0

(+) CS333_PROJECT ?= 1

(-) QEMU = qemu-system-i386

(+) QEMU = qemu-system-x86_64
```

2. System Call Tracing

```
### Makefile
(-)PRINT_SYSCALLS ?= 0

(+)PRINT_SYSCALLS ?= 1

### syscall.c
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
      (+)#ifdef PRINT_SYSCALLS
      (+)   cprintf("%s -> %d \n", syscallnames[num], curproc->tf-
>eax);
      (+)#endif
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

4. Date System Call

```
### Makefile

(-)CS333_PROJECT ?= 0

(+)CS333_PROJECT ?= 1

ifeq ($(CS333_PROJECT), 1)
CS333_CFLAGS += -DCS333_P1
(-)CS333_UPROGS += #_date
(+)CS333_UPROGS += _date
endif

### user.h
```

```
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(char*, int);
int mknod(char*, short, short);
int unlink(char*);
int fstat(int fd, struct stat*);
int link(char*, char*);
int mkdir(char*);
int chdir(char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int halt(void);
(+)#ifdef CS333_P1
(+)   int date(struct rtcdate*);
(+)#endif // CS333_P1

### usys.S

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(halt)
(+)SYSCALL(date)

### syscall.h

#define SYS_fork    1
#define SYS_exit    SYS_fork+1
```

```
#define SYS_wait    SYS_exit+1
#define SYS_pipe    SYS_wait+1
#define SYS_read    SYS_pipe+1
#define SYS_kill    SYS_read+1
#define SYS_exec    SYS_kill+1
#define SYS_fstat   SYS_exec+1
#define SYS_chdir   SYS_fstat+1
#define SYS_dup     SYS_chdir+1
#define SYS_getpid  SYS_dup+1
#define SYS_sbrk    SYS_getpid+1
#define SYS_sleep   SYS_sbrk+1
#define SYS_uptime  SYS_sleep+1
#define SYS_open    SYS_uptime+1
#define SYS_write   SYS_open+1
#define SYS_mknod   SYS_write+1
#define SYS_unlink  SYS_mknod+1
#define SYS_link    SYS_unlink+1
#define SYS_mkdir   SYS_link+1
#define SYS_close   SYS_mkdir+1
#define SYS_halt    SYS_close+1
(+)#define SYS_date    SYS_halt+1

### syscall.c

extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
#ifdef PDX_XV6
extern int sys_halt(void);
#endif // PDX_XV6
(+)#ifdef CS333_P1
(+)extern int sys_date(void);
(+)#endif //CS333_P1

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
#ifdef PDX_XV6
[SYS_halt]    sys_halt,
#endif // PDX_XV6
(+)#ifdef CS333_P1
(+)[SYS_date]    sys_date,
(+)#endif // CS333_P1
};

### sysproc.c

(+)int
(+)sys_date(void)
(+){
(+)struct rtcdate *d;
(+)   if(argptr (0 , (void*)&d, sizeof(struct rtcdate)) < 0 )
(+)   return âˆ’1;
(+)else {
(+)   cmostime(d);
(+)   return 0;
(+)   }
(+)}
```

5. Process Information

```
### proc.c
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);
  int found = 0;
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED) {
      found = 1;
      break;
    }
  if (!found) {
    release(&ptable.lock);
    return 0;
  }
  p->state = EMBRYO;
  p->pid = nextpid++;
  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  // 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->start_ticks = ticks;

  return p;
}

#elif defined(CS333_P1)
void
procdumpP1(struct proc *p, char *state_string)
{
  (+)int current = ticks - (p->start_ticks);
  (+)int val = current/1000;
  (+)int vali = current%1000;
  (-)cprintf("TODO for Project 1, delete this line and implement
procdumpP1() in proc.c to print a row\n");
  (+)cprintf("%d\t%s\t\t%d,%d\t%s\t%d\t", p->pid, p->name, val, vali,
states[p->state], p->sz);
  return;
}
#endif

### proc.h

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
  uint pid;                    // Process ID
  struct proc *parent;         // Parent process. NULL indicates no
parent
  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)
  (+)uint start_ticks;
};
```