

```
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(wait1)
SYSCALL(exit1)
SYSCALL(waitpid)
"usys.S" 34L, 508C
```

In usys.S I added 3 new system calls so I can use my new exit, wait, and waitpid.

```

void
exit1(int status)
{
    struct proc *curproc = myproc();
    struct proc *p;
    int fd;

    if(curproc == initproc)
        panic("init exiting");

    curproc->status = status;
    // Close all open files.
    for(fd = 0; fd < NOFILE; fd++){
        if(curproc->ofile[fd]){
            fileclose(curproc->ofile[fd]);
            curproc->ofile[fd] = 0;
        }
    }

    begin_op();
    iput(curproc->cwd);
    end_op();
    curproc->cwd = 0;

    acquire(&ptable.lock);

    // Parent might be sleeping in wait().
    wakeup1(curproc->parent);

    // Pass abandoned children to init.
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p->parent == curproc){
            p->parent = initproc;
            if(p->state == ZOMBIE)
                wakeup1(initproc);
        }
    }

    // Jump into the scheduler, never to return.
    curproc->state = ZOMBIE;
    sched();
    panic("zombie exit");
}

```

In proc.c I added my new exit function that took in the status pointer and set the current status to the one passed into exit.

```

int
wait1(int* status)
{
    struct proc *p;
    int havekids, pid;
    struct proc *curproc = myproc();

    acquire(&ptable.lock);
    for(;;){
        // Scan through table looking for exited children.
        havekids = 0;
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->parent != curproc)
                continue;
            havekids = 1;
            if(p->state == ZOMBIE){
                // Found one.
                if (status){
                    *status = p->status;
                }
                pid = p->pid;
                kfree(p->kstack);
                p->kstack = 0;
                freevm(p->pgdir);
                p->pid = 0;
                p->parent = 0;
                p->name[0] = 0;
                p->killed = 0;
                p->state = UNUSED;
                release(&ptable.lock);
                return pid;
            }
        }

        // No point waiting if we don't have any children.
        if(!havekids || curproc->killed){
            release(&ptable.lock);
            return -1;
        }

        // Wait for children to exit. (See wakeup1 call in proc_exit.)
        sleep(curproc, &ptable.lock); //DOC: wait-sleep
    }
}

```

In my new wait I added a statement that made the proc status equal to the new status that was passed in. I also returned the pid.

```

int
waitpid(int pid, int* status, int options)
{
    struct proc *p;
    int havekids, npid;
    struct proc *curproc = myproc();

    acquire(&ptable.lock);
    for(;;){
        // Scan through table looking for exited children.
        havekids = 0;
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p->pid == pid){
                if(p->parent != curproc)
                    continue;
                havekids=1;
                if(p->state == ZOMBIE){
                    // Found one.

                    if (status){
                        *status = p->status;
                    }

                    npid = p->pid;
                    kfree(p->kstack);
                    p->kstack = 0;
                    freevm(p->pgdir);
                    p->pid = 0;
                    p->parent = 0;
                    p->name[0] = 0;
                    p->killed = 0;
                    p->state = UNUSED;
                    release(&ptable.lock);
                    return npid;
                }
            }
            else if (options == WNOHANG){
                if(curproc->status >= 0 ){
                    release(&ptable.lock);
                    return curproc->status;
                }
                else{
                    return -1;
                }
            }
        }
    }
    // No point waiting if we don't have any children.
    if(!havekids || curproc->killed){
        release(&ptable.lock);
        return -1;
    }


    // Wait for children to exit.  (See wakeup1 call in proc_exit.)
    sleep(curproc, &ptable.lock); //DOC: wait-sleep

```

In waitpid we copy pasted our wait pid and then we checked the current pid with the status given. For the extra credit we checked if the status was greater than negative one then we released the lock and returned the current status.

```
cbott001@sledge:~/xv6
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup     10
#define SYS_getpid  11
#define SYS_sbrk    12
#define SYS_sleep   13
#define SYS_uptime  14
#define SYS_open    15
#define SYS_write   16
#define SYS_mknod   17
#define SYS_unlink  18
#define SYS_link    19
#define SYS_mkdir   20
#define SYS_close   21
#define SYS_exit1   22
#define SYS_wait1   23
#define SYS_waitpid 24
~
~
~
~
~
```

Just added the new exit, wait and the waitpid

 cbott001@sledge:~/xv6

```
extern int sys_exit(void);
extern int sys_exit1(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_wait1(void);
extern int sys_waitpid(void);
extern int sys_write(void);
extern int sys_uptime(void);

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,
[SYS_exit1]     sys_exit1,
[SYS_wait1]     sys_wait1,
[SYS_waitpid]   sys_waitpid,
};

void
syscall(void)
```

Just added the new system calls



cbott001@sledge:~/xv6

```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int exit1(int);
int wait(void);
int wait1(int*);
int waitpid(int, int*, int);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
```

Still making the new system calls