Jared Banzuela
jbanz001@ucr.edu
862023056

Lab 1: System Calls

The modified proc.h to add an exit_status int to track exit statuses

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// 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 exit_status;
};
```

a) The exit was modified to track status in PCB

```
// Exit the current process.  Does not return.
// An exited process remains in the zombie state
// until its parent calls wait() to find out it exited.
void
exit(int status)
{
  struct proc *curproc = myproc();
  struct proc *p;
  int fd;

  curproc->exit_status = status;

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

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

sysproc changed for wait and exit

```
int
sys_exit(void)
{
  exit(0);
  return 0;  // not reached
}

int
sys_wait(void)
{
  return wait(0);
}
```

b) The wait was modified to track status in PCB

```
// Wait for a child process to exit and return its pid.
// Return -1 if this process has no children.
int
wait(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.
        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;
        if (status)
          *status = p->exit_status;
        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);
      if(status)
        *status = p->exit_status;
      return -1;
    }
    // Wait for children to exit.  (See wakeup1 call in proc_exit.)
    sleep(curproc, &ptable.lock);  //DOC: wait-sleep
  }
}
```

I was not able to return correct values for lab1.c tests

```
$ lab1 1

 This program tests the correctness of your lab#1

  Parts a & b) testing exit(int status) and wait(int* status):

This is child with PID# 4 and I will exit with status 0

 This is the parent: child with PID# 4 has exited with status 12264

This is child with PID# 5 and I will exit with status -1

 This is the parent: child with PID# 5 has exited with status 12264
$
```

I was unable to get past part a