Parth Mangrola
861286610
Section 21
Lab 1

defs.h

- Modified exit and wait system call
- Added waitpid system call

```
int             cpuid(void);
void            exit(int status);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(int *status);
int             waitpid(int pid, int *status, int options);
void            wakeup(void*);
void            yield(void);
```

proc.h
- Line 52: Added status to proc

syscall.c
- Line 106 & 130: Added waitpid function

syscall.h
- Line 23: Added waitpid define

Makefile
- Modified makefile to run test

proc.c

- Line 265: Modified exit to store status
- Line 293: Modified wait to return status
- Line 321-371: Added waitpid function

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



  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;
   }
  }

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

  acquire(&ptable.lock);

  // Parent might be sleeping in wait().
  wakeup1(curproc->parent);
        // test
  // 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);
    }
  }
  curproc->status = status; //lab1

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

```
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.

        if(p->status != 0){   //lab1 if not null return exit 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
  }
}
```

sysproc.c

- Added sys_waitpid
- Modified sys_exit and sys_wait

```
int
sys_exit(void)
{
  int status;
  if(  argptr(0,(void*) &status, sizeof(status)) < 0){
          return -1;
        }

  exit(status);
  return 0;  // not reached
}

int
sys_wait(void)
{
  int *status;
  if(argptr(0,(void*)&status, sizeof(status))<0){
          return -1;
  }
  return wait(status);
}

int sys_waitpid(void){
  int pid, *status, options;
  if( argint(0, &pid) < 0){
          return -1;
  }
  if( argptr(1,(void*)&status, sizeof(status))< 0){
          return -1;
  }
  if(  argint(2, &options) < 0){
          return -1;
  }
  return waitpid(pid, status, 0);
}
```

```
 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# 6 and I will exit with status 0

 This is the parent: child with PID# 6 has exited with status 42

This is child with PID# 7 and I will exit with status -1

Error using fork
```

```
This program tests the correctness of your lab#1

 Part c) testing waitpid(int pid, int* status, int options):

The is child with PID# 14 and I will exit with status 18

The is child with PID# 16 and I will exit with status 20
The is child with PID# 15 and I will exit with status 19

The is child with PID# 17 and I will exit with status 21

The is child with PID# 18 and I will exit with status 22

This is the parent: Now waiting for child with PID# 17

This is the partent: Child# 17 has exited with status 21

This is the parent: Now waiting for child with PID# 15

This is the partent: Child# 15 has exited with status 19

This is the parent: Now waiting for child with PID# 16

This is the partent: Child# 16 has exited with status 20

This is the parent: Now waiting for child with PID# 14

This is the partent: Child# 14 has exited with status 18

This is the parent: Now waiting for child with PID# 18

This is the partent: Child# 18 has exited with status 22
```