LAB 2: FUN WITH SYSTEM CALLS

a) Change the exit system call signature to void exit (int status):

Changes made in the following files:

Open ▾ user.h ~/xv6-public

```
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(int) __attribute__((noreturn));
```

Open ▾ defs.h ~/xv6-public

```
103
104 //PAGEBREAK: 16
105 // proc.c
106 int             cpuid(void);
107 void            exit(int);
```

Open ▾ *sysproc.c ~/xv6-public

```
16 int
17 sys_exit(void)
18 {
19         int exit_status;
20         argint(0, &exit_status);
21         exit(exit_status);
22         return 0;  // not reached
23 }
```

Open ▾ proc.h ~/xv6-public Save

```
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this
   process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52   int exit_status;             // exit status for proc
53 };
```

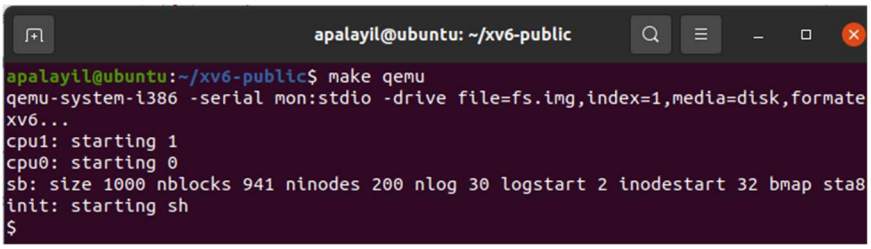
```
pen                                    *proc.c
                                       ~/xv6-public
void
exit(int status)
{
  struct proc *curproc = myproc();
  struct proc *p;
  int fd;

  if(curproc == initproc)
  {
    curproc->exit_status = 1; //error give 1 exit status
    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);

  // 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");
}
```

The following changes were made for the exit () function:

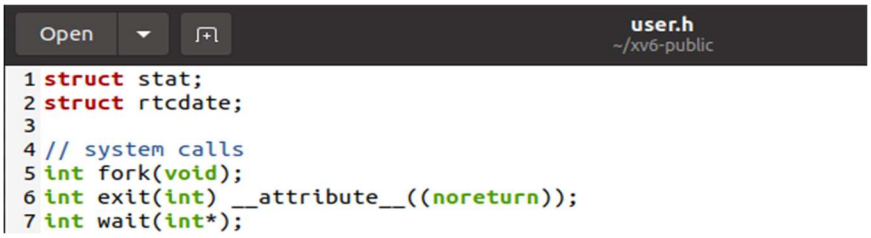
- In cat.c lines 15, 20, and 37 were changed to exit (-1); Lines 31 and 42 were changed to exit (0)
- In echo.c line 12 was changed to exit (0)
- In forktest.c lines 33, 39 and 45 were changed to exit (-1); Lines 28 and 55 were changed to exit (0)
- In grep.c lines 46, 52, and 58 were changed to exit (-1); Line 63 was changed to exit (0)
- In init.c lines 27 and 32 were changed to exit (-1)
- In kill.c lines 12 and 16 were changed to exit (0)
- In ln.c lines 10 was changed to exit (-1); Line 14 was changed to exit (0)
- In ls.c lines 80 was changed to exit (27); Line 84 was changed to exit (1)
- In mkdir.c lines 22 was changed to exit (0); Line 12 was changed to exit (-1)
- In rm.c lines 22 was changed to exit (0); Line 12 was changed to exit (-1)
- In sh.c lines 77, 87, 171, and 178 were changed to exit (0); Line 68 and 130 was changed to exit (1)
- In stressfs.c line 48 was changed to exit (0)

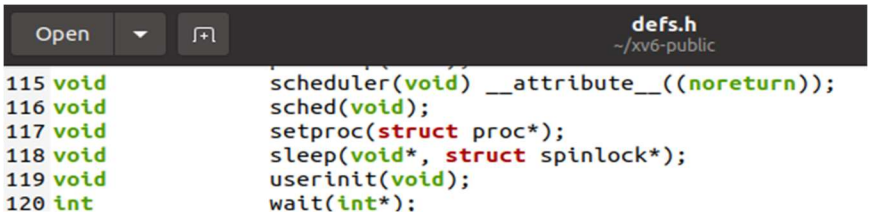
- In trap.c lines 41, 45, 101, and 111 were changed to exit (0)
- In usertests.c lines 24, 28, 32, 36, 52, 57, 61, 65, 92, 97, 110, 127, 150, 155, 159, 169, 176, 182, 197, 204, 213, 222, 227, 239, 273, 278, 283, 288, 299, 313, 324, 346, 352, 450, 511, 533, 540, 547, 566, 574, 596, 607, 614, 634, 637, 665, 673, 677, 686, 690, 694, 714, 718, 724, 730, 736, 740, 790, 810, 814, 824, 832, 868, 904, 915, 927, 944, 950, 962, 968, 976, 981, 987, 992, 1001, 1005, 1009, 1013, 1019, 1023, 1095, 1099, 1107, 1111, 1130, 1158, 1164, 1172, 1179, 1185, 1189, 1196, 1213, 1217, 1222, 1228, 1250, 1254, 1266, 1278, 1293, 1324, 1354, 1358, 1402, 1408, 1431, 1439, 1445, 1458, 1468, 1473, 1481, 1486, 1493, 1502, 1507, 1516, 1540, 1585, 1603, 1633, 1639, 1718, 1721, and 1733 were changed to exit(1);Lines 67, 103, 105, 133, 232, 327, 421, 454, 487, 550, 618, 746, 752, 757, 795, 847, 886, 957, 996, 1029, 1034, 1038, 1042, 1046, 1050, 1054, 1058, 1062, 1066, 1070, 1074, 1078, 1082, 1086, 1090, 1103, 1137, 1234, 1238, 1258, 1262, 1270, 1274, 1298, 1303, 1308, 1312, 1316, 1320, 1330, 1335, 1391, 1396, 1448, 1575, 1630, 1755, and 1802 were changed to exit(0)
- In wc.c lines 30 and 48 were changed to exit (1); Lines 42 and 53 were changed to exit (0)
- In zombie.c line 13 was changed to exit (0);

The xv6 complies once make qemu or make qemu-nox is entered.



b) Update the wait system call signature to it wait (int *status):





```
Open                                              *sysproc.c
                                                  ~/xv6-public
25   return 0;  // not reached
26 }
27
28 int
29 sys_wait(void)
30 {
31         char *pointer; //arg ptr requires char ptr
32         argptr(0,&pointer,32);
33         return wait((int *)pointer);
34 }
```

```
Open                                              *proc.c
                                                  ~/xv6-public
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.
        //for exit 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;
        if (status != 0) {
                *status = p->exit_status; //Exit Status
                }
        p->exit_status = 0;
        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
  }
```

The following changes were made for the wait () function:

- In sh.c lines 96, 120, 121, and 169 were changed to wait (&stat)
- In stressfs.c line 46 was changed to wait (&stat)
- In forktest.c lines 37 and 43 was changed to wait (&stat)

```
apalayil@ubuntu: ~/xv6-public
apalayil@ubuntu:~/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,formate
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting sh
$
```

c) Add a waitpid system call: int waitpid (int pid, int *status, int options):

user.h ~/xv6-public

```
24 int sleep(int);
25 int uptime(void);
26 int waitpid(int,int*,int);
```

defs.h ~/xv6-public

```
int  wait(int*);
void wakeup(void*);
void yield(void);
int  waitpid(int,int*,int);
```

*sysproc.c ~/xv6-public-apalayil

*mytests.c | *sysproc.c

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

*proc.c ~/xv6-public

```
                p->parent = curproc;
                sleep(curproc, &ptable.lock); //does not go away - goes to
        }
}*/
int
waitpid(int pid, int* status, int options)
{
  struct proc *p, *curproc = myproc();
  int found_pid;
  acquire(&ptable.lock);
  //Loops continuously till the process with given pid is terminated
  for(;;) {
    found_pid = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
      //processes will check if ZOMBIE state, else it will inherit it.
      if(p->pid != pid) continue;
      found_pid = 1;
      if(p->state == ZOMBIE) {
        //Found the process with the given pid that has exited.
        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;
        if(status) *status = p->exitstatus;
        p->exitstatus = 0;
        release(&ptable.lock);
        return pid;
      } else if(options == 1) {
        release(&ptable.lock);
        return 0;
      }
    }
    // or the current process is killed.
    if(!found_pid || curproc->killed) {
      release(&ptable.lock);
      return -1;
    }
    // Wait for the process with the given pid to exit.
    sleep(curproc, &ptable.lock);
  }
}
```

d) Write an example program to illustrate that your waitpid works:

Open *mytests.c ~/xv6-public-apalayil Save

```c

#include "types.h"
#include "stat.h"
#include "user.h"
#define NULL 0

void exitTest(int status) {
          int pid;
          int return_pid, fork_count = 3;
          int exit_status;
          int i;
          // use this part to test exit(int status) and wait(int* status)
          for(i = 0; i < fork_count; i++)
          {
            pid = fork();
            if(pid == 0){ // only the child executed this code
            if(i == 0){
                printf(1, "\nThis is child with PID# %d ,exit with status %d\n", getpid(), 0);
                exit(0);
              }
            else{
                printf(1, "\nThis is child with PID# %d ,exit with status %d\n" ,getpid(), -1);
                exit(-1);
              }
            }
            else if (pid > 0){ // only the parent executes this code
              return_pid = wait(&exit_status);
              printf(1, "\nThis is the parent: child with PID# %d has exited with status %d\n", return_pid, exit_status);
            }
            else{ // something went wrong with fork system call
              printf(2, "\n Error using fork\n");
              exit(-1);
            }
          }
}

void waitpidTest(void) {
          int exit_status;
          int i;
          int ret_fork;
          // use this part to test wait(int pid, int* status, int options)
          int cpid;
        // fork 6 children
        for(i=0; i<6;i = i +1) {
                ret_fork = fork();
                if(ret_fork  == 0) { // for some reason ref pid[_] in any way breaks the loop
                        printf(1, "Child Process: %d\n",i);
                        int exit_stat = 100 + i;
                        exit(exit_stat); //exit right away
                }
        }
        cpid = waitpid(6,&exit_status,0);//note the pids will range for 4 to 10
        printf(1,"Child with pid: %d, exited with status: %d\n",cpid, exit_status);
        cpid = waitpid(8,&exit_status,0);//note the pids will range for 4 to 10
        printf(1,"Child with pid: %d, exited with status: %d\n",cpid, exit_status);
        //clean up the rest of the processes
        int has_children = 1;
        while(has_children != -1) {
                has_children = wait(NULL);
        }
}

int main() {
        exitTest(NULL);
        waitpidTest();
        exit(0);
}
```

Compiling xv6-public and running the test files:

Thus the results show when the parent process waits for the child process to finish printing after which it exits.