Kinjal Mugatwala (kmuga001)
CS 153, Discussion section: 022

Lab 1 Report

Summary of Lab 1:

The overall purpose of this lab is to be able to use a status value as a checking measure to see if the process is exiting or terminating correctly. In this lab, I am updating the exit system call and wait system call and creating a waitpid system call. The exit system call actually terminates the process and stores the process's status into the status variable. The wait system call forces the current process to wait until all of its children processes have exited and the status parameter will be used for storing the exiting process. The waitpid system call forces a process to wait for a particular child to exit.

Output for "Lab1 1" Test Bench:

```
$ lab1

[ This program tests the correctness of your lab#1

type "lab1 1" to test exit and wait, "lab1 2" to test waitpid and "lab1 3" to test the extra
$ 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# 5 and I will exit with status 0

 This is the parent: child with PID# 5 has exited with status 16360

This is child with PID# 6 and I will exit with status -1

 This is the parent: child with PID# 6 has exited with status 16360
```

Output for "Lab1 2" Test Bench:

```
$ lab1 2

 This program tests the correctness of your lab#1

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

 The is child with PID# 21 and I will exit with status 25

 The is child with PID# 20 and I will exit with status 24

 The is child with PID# 22 and I will exit with status 26

 The is child with PID# 24 and I will exit with status 28


 This is the parent: Now waiting for child with PID# 23
 The is child with PID# 23 and I will exit with status 27

 This is the partent: Child# 23 has exited with status 0

 This is the parent: Now waiting for child with PID# 21

 This is the partent: Child# 21 has exited with status 0

 This is the parent: Now waiting for child with PID# 22

 This is the partent: Child# 22 has exited with status 0

 This is the parent: Now waiting for child with PID# 20

 This is the partent: Child# 20 has exited with status 0

 This is the parent: Now waiting for child with PID# 24

 This is the partent: Child# 24 has exited with status 0
```

Part A - Exit System Call Update:

In order to store the status value, I needed to add int status to the proc structure in proc.h.

```
// 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 status;                  // for the exit status system call (added myself)
};
```

To update the exit system call, I changed the function definition to void exit(int status) and added the int status parameter in proc.c, user.h, defs.h, and sysproc.c in order to check that the system is correctly exiting. Also, because I added in a parameter to the exit system call, I needed to update any .c file that uses the exit() system call and I did that by adding a 0 with the use of grep.

proc.c:

```
void
exit(int status)
{
```

user.h

```
int exit(int status) __attribute__((noreturn));
```

defs.h

```
void            exit(int status);
```

sysproc.c

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

Part B - Wait System Call Update:

The wait system call is updated by changing the function definition to int wait(int *status) and so, we need to accomodate for the new *status parameter.

proc.c:

- Added in the int *status parameter
- Added an if statement about the status. If the status is not equal to 0, we want to assign the status to the current process' status because according to the status, a child has not exited yet.

```
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(status != 0){
          *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
  }
}
```

user.h

```
int wait(int *status);
```

defs.h

```
int             wait(int *status);
```

sysproc.c

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

Part C - waitpid system call:

The waitpid function definition is int waitpid(int pid, int *status, int options). The waitpid system call is implemented like the wait() system call, but in the waitpid system call, we want to track the status of a specific process with the specified pid value. In this function, I am waiting for a specific process to exit based on the pid value. I am returning the pid value of the child process that exited/terminated.

proc.c:

- Added an if statement about the status. If the status is not equal to 0, we want to assign the status to the current process' status because according to the status, a child has not exited yet.
- Added another if statement to check the pid value (the process id value). If the current pid is not the pid from the parameters, we want to "continue" and move through the for loop until we get the pid value we want.

```
int waitpid(int pid, int *status, int options)
{
  //need to implement how we are trying to check if the value is the pid we want
  struct proc *p;
  int havekids;
  struct proc *curproc = myproc();

  acquire(&ptable.lock);
  for(;;){
    havekids = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->pid != pid)
        continue;
      if(p->parent != curproc)
        continue;
      havekids = 1;
      if(p->state == ZOMBIE){
        if(status != 0){
         *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;
        //if(status){
        // *status = p->status;
        //}
        return pid;
      }
    }

    if(!havekids || curproc->killed){
      release(&ptable.lock);
      return -1;
    }

    sleep(curproc, &ptable.lock);

  }
}
```

user.h

```
int waitpid(int pid, int *status, int options);
```

defs.h

```
int             waitpid(int pid, int *status, int options);
```

sysproc.c

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

syscall.h

```
#define SYS_waitpid 22
```

syscall.c

```
[SYS_waitpid] sys_waitpid,
```

```
extern int sys_waitpid(void);
```

usys.S

```
SYSCALL(waitpid)
```