Lab 1 Report

Steven Nguyen - snguy057

Brittney Mun - bmun001

Changes

proc.c

Changes to exit function:

- Line 229: Changed exit(void) to exit(int status) // Passed in exit status
- Line 226: Added curproc->exitStatus = status; // Saves exit status to the proc struct

Changes to wait function:

- Line 228: changed wait(void) to wait(int *status) // variable to save child's status
- Line 303: added *status = p->exitStatus; // Saves child's exit status to status

Implemented waitpid with WNOHANG option. Most of the code copied from wait().

- Lines 320-367 Implemented here
- Line 328: changed havekids to pidFound // naming
- Line 328: removed pid, not necessary anymore
- Line 334,338: changed havekids to pidFound
- Line 336: changed to if(p->pid == pid) //checks if the process's pid is the pid we are waiting on.
- Lines 363-367: Implemented WNOHANG

```
224   // Exit the current process.  Does not return.
225   // An exited process remains in the zombie state
226   // until its parent calls wait(&status) to find out it exited.
227   // Lab1: saves status of the exited process
228   void
229   exit(int status)
230   {
```

```
231   struct proc *curproc = myproc();
232   struct proc *p;
233   int fd;
234
235   if(curproc == initproc)
236     panic("init exiting");
237
238   // Close all open files.
239   for(fd = 0; fd < NOFILE; fd++){
240     if(curproc->ofile[fd]){
241       fileclose(curproc->ofile[fd]);
242       curproc->ofile[fd] = 0;
243     }
244   }
245
246   begin_op();
247   iput(curproc->cwd);
248   end_op();
249   curproc->cwd = 0;
250
251   acquire(&ptable.lock);
252
253   // Parent might be sleeping in wait(&status).
254   wakeup1(curproc->parent);
255
256   // Pass abandoned children to init.
257   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
258     if(p->parent == curproc){
259       p->parent = initproc;
260       if(p->state == ZOMBIE)
261         wakeup1(initproc);
262     }
263   }
264
265   // Lab1: Save process's exit status so parent can retrieve
```

```
265   // Lab1: Save process's exit status so parent can retrieve
266   curproc->exitStatus = status;
267
268   // Jump into the scheduler, never to return.
269   curproc->state = ZOMBIE;
270   sched();
271   panic("zombie exit");
272 }
273
274 // Wait for a child process to exit and return its pid.
275 // Return -1 if this process has no children.
276 // Lab1: Changed to return child's exit status
277 int
278 wait(int *status)
279 {
280   struct proc *p;
281   int havekids, pid;
282   struct proc *curproc = myproc();
283
284   acquire(&ptable.lock);
285   for(;;){
286     // Scan through table looking for exited children.
287     havekids = 0;
288     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
289       if(p->parent != curproc)
290         continue;
291       havekids = 1;
292       if(p->state == ZOMBIE){
293         // Found one.
294         pid = p->pid;
295         kfree(p->kstack);
296         p->kstack = 0;
297         freevm(p->pgdir);
298         p->pid = 0;
299         p->parent = 0;
300         p->name[0] = 0;
301         p->killed = 0;
302         p->state = UNUSED;
303         *status = p->exitStatus; //L1: Retrieve child's exit status
```

```
304           release(&ptable.lock);
305           return pid;
306         }
307       }
308
309       // No point waiting if we don't have any children.
310       if(!havekids || curproc->killed){
311         release(&ptable.lock);
312         return -1;
313       }
314
315       // Wait for children to exit.  (See wakeup1 call in proc_exit.)
316       sleep(curproc, &ptable.lock);  //DOC: wait-sleep
317     }
318   }

320   // Lab 1
321   // Wait for a process with a specific pid to exit
322   // Return -1 if PID is not found
323   // Implemented WNOHANG
324   int
325   waitpid(int pid, int *status, int options)
326   {
327     struct proc *p;
328     int pidFound; // pid; //no longer need pid
329     struct proc *curproc = myproc();
330
331     acquire(&ptable.lock);
332     for(;;){
333       // Scan through table looking for the exited process.
334       pidFound = 0;
335       for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
336         if(p->pid != pid) //check if pid of process is the one we are
337           continue;       //     waiting for else continue searching.
338         pidFound = 1;     // found the process!
339         if(p->state == ZOMBIE){ // process has exited
340           // Found one.
341           //pid = p->pid; // we know this from the function call
```

```
342           kfree(p->kstack);
343           p->kstack = 0;
344           freevm(p->pgdir);
345           p->pid = 0;
346           p->parent = 0;
347           p->name[0] = 0;
348           p->killed = 0;
349           p->state = UNUSED;
350           *status = p->exitStatus; // Retrieve process's exit status
351           release(&ptable.lock);
352           return pid;
353         }
354       }
355
356       // No point waiting if process does not exist.
357       if(!pidFound || curproc->killed){
358         release(&ptable.lock);
359         return -1;
360       }
361
362       // WNOHANG, if PID was found and the child has not exited yet
363       if(options == 1 && pidFound == 1) {
364         release(&ptable.lock); // release lock
365         *status = -1; // Has not exited yet
366         return 0; //
367       }
368
369       // Wait for process to exit.  (See wakeup1 call in proc_exit.)
370       sleep(curproc, &ptable.lock);  //DOC: wait-sleep
371     }
372   }
```

sysproc.c

Changes to sys_exit

- Line 20: added status to read in the process's exit status
- Line 21: use argint to get status
- Line 22: changed exit() to exit(status), calls exit function

Changes to sys_wait

- Line 30: added status
- Line 31: use argptr to get pointer of status variable
- Line 32: call wait() to wait(status)

Added sys_waitpid(void)

- Lines 36-46: Implemented here

```
16  int
17  sys_exit(void)
18  {
19    // Lab1: Added to return status
20    int status = 0;
21    argint(0, &status);
22    exit(status);
23    return 0;
24  }
25
26  int
27  sys_wait(void)
28  {
29    // Lab 1: Added to return status
30    int* status;
31    argptr(0, (char**) &status, sizeof(int*));
32    return wait(status);
33  }
34
35  // Lab 1: New syscall that waits on a specific PID
36  int
37  sys_waitpid(void)
38  {
39    int pid;
40    int* status;
41    int options;
42    argint(0, &pid);
43    argptr(1, (char**) &status, sizeof(int*));
44    argint(2, &options);
45    return waitpid(pid, status, options);
46  }
```

defs.h

- Line 107: exit() to exit(int)
- Line 120: wait() to wait(int*)
- Line 212: added waitpid(int, int*, int)

user.h

- Line 6: exit() to exit(int)
- Line 7: wait() to wait(int*)
- Line 8: added int waitpid(int, int*, int)

syscall.c

- Line 106: added extern int sys_waitpid(void);
- Line 130: added [SYS_waitpid] sys_waitpid

proc.h

In proc struct:

- Line 52: added int exitStatus //new variable for storing the status's exit status

syscall.h

- Line 23: added define for waitpid

usys.S

- Line 32: added syscall for waitpid

exit() -> exit(0)

Changed in:

- cat.c
- echo.c
- forktest.c
- grep.c
- init.c

- kill.c
- ln.c
- ls.c
- mkdir.c
- proc.c
- rm.c
- sh.c
- stressfs.c
- trap.c
- usertests.c
- wc.c
- zombie.c

wait() -> wait(0)

Changed in:

- sh.c
- stressfs.c
- usertests.c