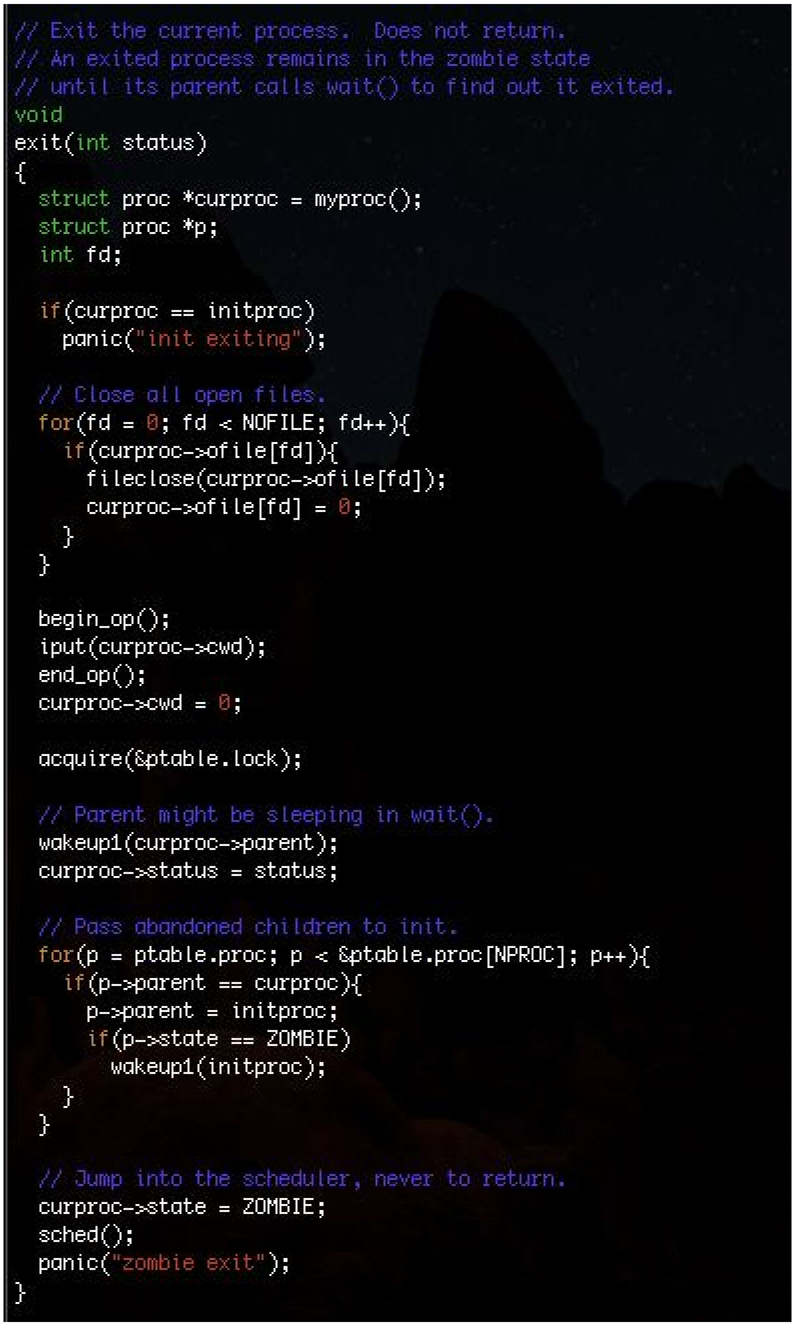
Lab 1 Report

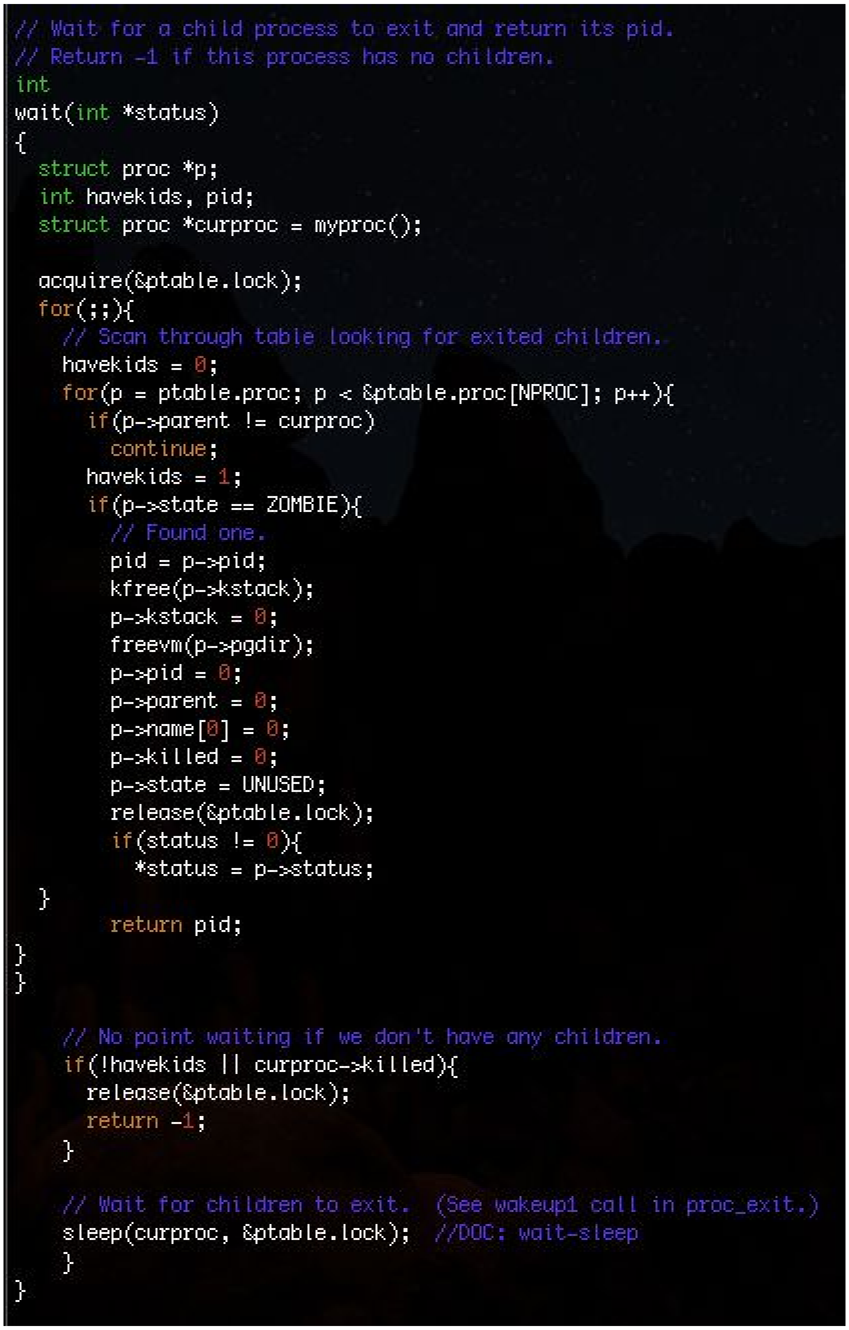
Celyna Su (SID: 862037643) & Wei Xin Koh (SID: 862191103)

For Lab 1, we have implemented the following changes to the code:

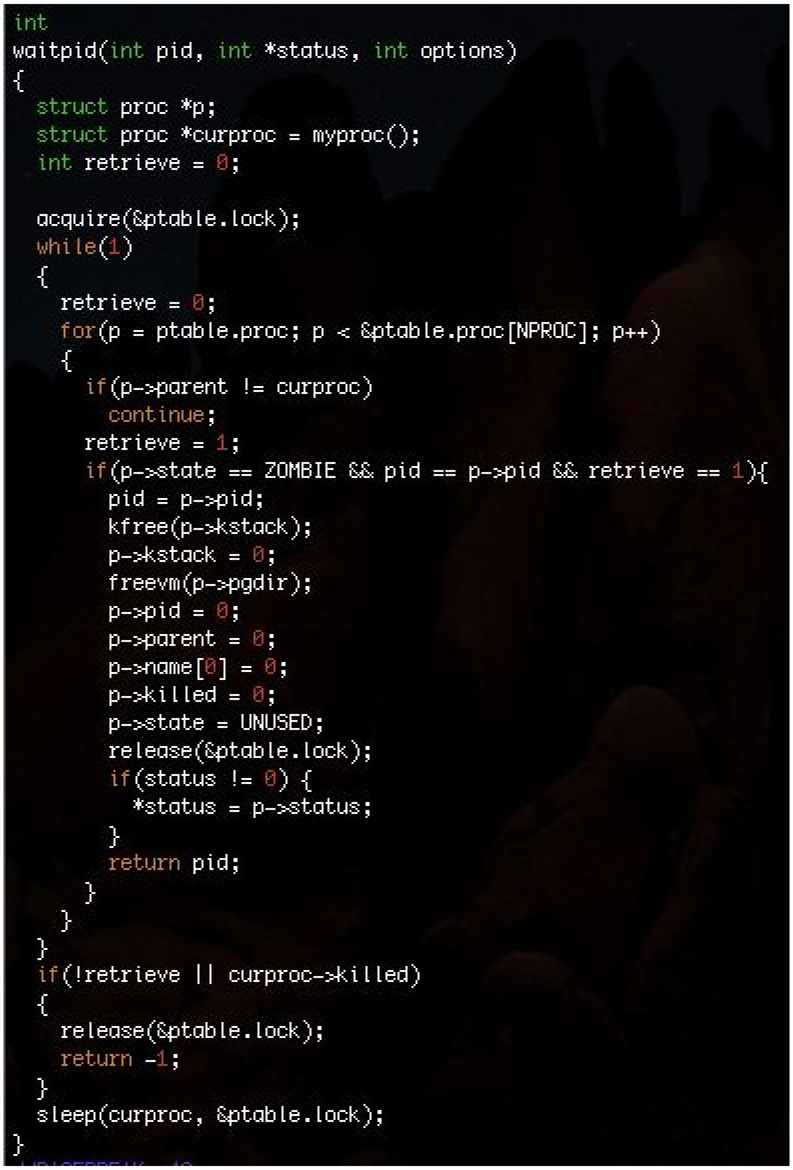
1. Changed exit system call signature to void exit(int status) and implemented this change in user.h, defs.h, sysproc.h, prof.h, and other files that called exit()
    a. We also changed the user space programs' call to this function from exit() to exit(0)
    b.



```
// 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;

  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);
  curproc->status = status;

  // 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");
}
```

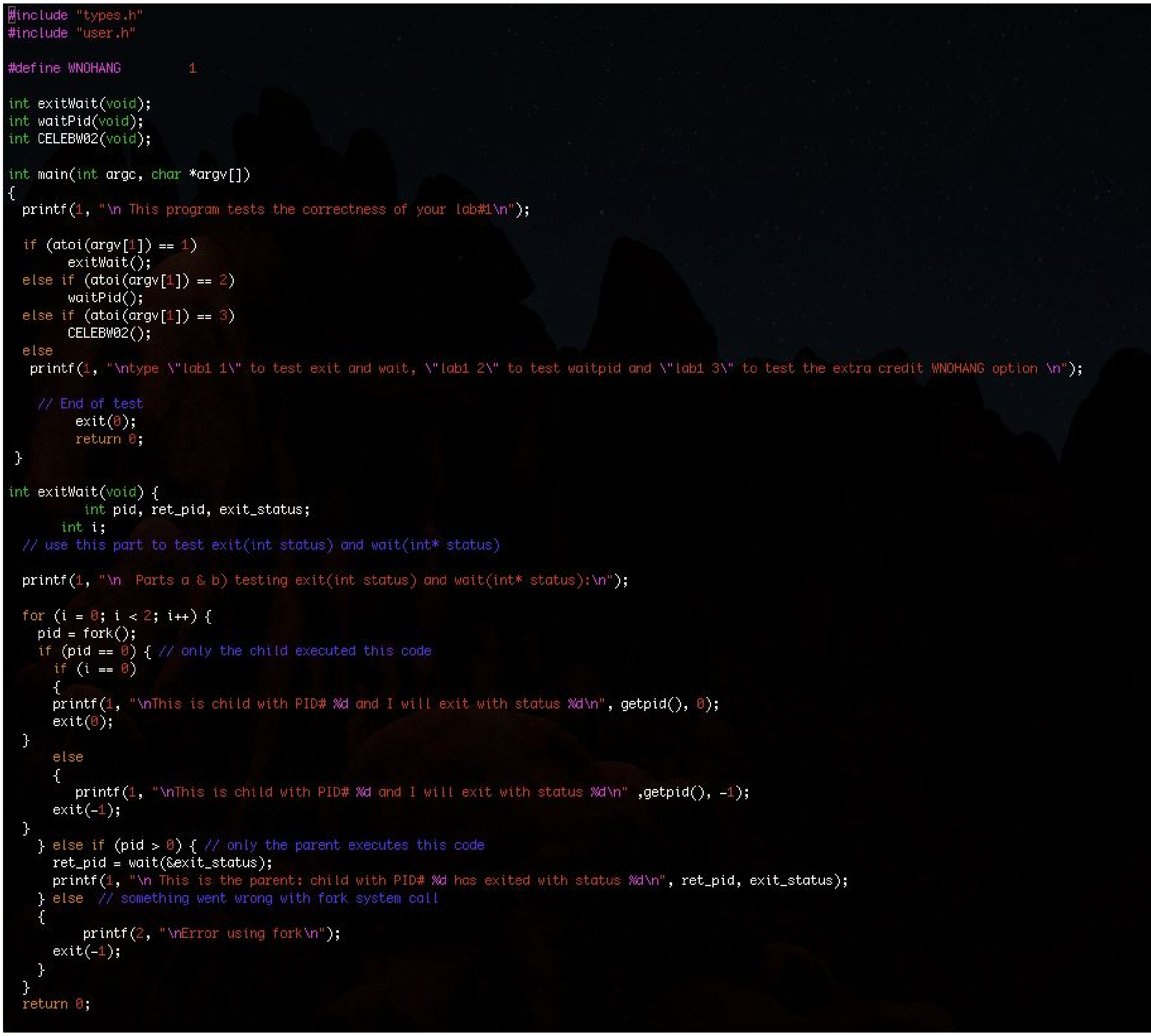
2. Changed wait system call signature to int wait(int* status)
    a. We deallocate the passed in status by setting *status = p->status in sysproc.h
    b. We also changed the wait() function call to wait(0) in user space programs
    c. 

```
// 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;
        p->state = UNUSED;
        release(&ptable.lock);
        if(status != 0){
          *status = p->status;
  }
        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
    }
}
```

3. Added a waitpid system call: int waitpid(int pid, int *status, int options)
    a. Added waitpid to SYSCALL to run it
    b. Added waitpid to header files as well
    c. 

```
int
waitpid(int pid, int *status, int options)
{
  struct proc *p;
  struct proc *curproc = myproc();
  int retrieve = 0;

  acquire(&ptable.lock);
  while(1)
  {
    retrieve = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    {
      if(p->parent != curproc)
        continue;
      retrieve = 1;
      if(p->state == ZOMBIE && pid == p->pid && retrieve == 1){
        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);
        if(status != 0) {
          *status = p->status;
        }
        return pid;
      }
    }
  }
  if(!retrieve || curproc->killed)
  {
    release(&ptable.lock);
    return -1;
  }
  sleep(curproc, &ptable.lock);
}
```

4. Wrote test files for waitpid
    1. Added a testbench titled lab1.c
    2. Modified the Makefile by adding lab1.c to UPROGS and Extra



```
#include "types.h"
#include "user.h"

#define WNOHANG         1

int exitWait(void);
int waitPid(void);
int CELEBW02(void);

int main(int argc, char *argv[])
{
  printf(1, "\n This program tests the correctness of your lab#1\n");

  if (atoi(argv[1]) == 1)
        exitWait();
  else if (atoi(argv[1]) == 2)
        waitPid();
  else if (atoi(argv[1]) == 3)
        CELEBW02();
  else
   printf(1, "\ntype \"lab1 1\" to test exit and wait, \"lab1 2\" to test waitpid and \"lab1 3\" to test the extra credit WNOHANG option \n");

    // End of test
         exit(0);
         return 0;
 }

int exitWait(void) {
          int pid, ret_pid, exit_status;
       int i;
  // use this part to test exit(int status) and wait(int* status)

  printf(1, "\n  Parts a & b) testing exit(int status) and wait(int* status):\n");

  for (i = 0; i < 2; i++) {
    pid = fork();
    if (pid == 0) { // only the child executed this code
      if (i == 0)
      {
      printf(1, "\nThis is child with PID# %d and I will exit with status %d\n", getpid(), 0);
      exit(0);
  }
      else
      {
         printf(1, "\nThis is child with PID# %d and I will exit with status %d\n" ,getpid(), -1);
      exit(-1);
  }
    } else if (pid > 0) { // only the parent executes this code
      ret_pid = wait(&exit_status);
      printf(1, "\n This is the parent: child with PID# %d has exited with status %d\n", ret_pid, exit_status);
    } else  // something went wrong with fork system call
    {
          printf(2, "\nError using fork\n");
      exit(-1);
    }
  }
  return 0;
```