IC221: Systems Programming
12-Week Written Exam

April 2, 2014

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. Show all work, but please be legible. *Microscopic writing will not be graded*.

You are allowed a single crib sheet for this exam on an 8.5"x11" sheet of paper, hand written. You must turn in your crib sheet with your exam.

Name: ______________________________

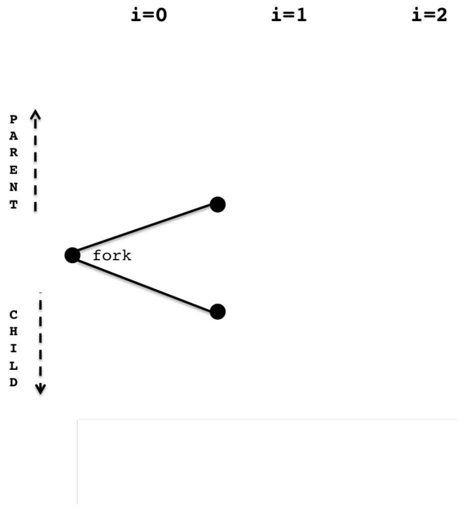
Section: ____________________________

Alpha: ______________________________

Question	Points	Score
1	15	
2	15	
3	15	
4	15	
5	15	
Total:	75	

1. Consider the following program:

```
int main(){
 int i, status;

 for(i=0;i<3;i++){
  if(fork() == 0){
   /*child*/
   printf("Child: %d: Hello World!", i);
   _Exit(0);
  }else{
   /*parent*/
    printf("Parent: %d: Hello World!\n",i);
  }
 }

 wait(&status);
 if(WIFSIGNALED(status)){
   printf("Process signaled\n");
 }
 return 0;
}
```



(a) [**4 points**] To the right of the program, **complete the drawing of the process tree above**, where each dot represents a process and a fork results in a split, up for the parent and down for the child. **Below answer:** How many total processes are created, including the initial parent process? How many total processes are running after the loop? Explain.

(b) [**4 points**] How many times does "`Child: ...`" print and how many times does "`Parent: ...`" print to standard output? Explain.

(c) [**3 points**] Does the message "`Process signaled`" print? Why or why not based on the value of `status`?

(d) [**4 points**] At the end of the program, are there any zombie processes? If so, why are there zombies and what happens to them when the parent terminates? If not, why aren't there any zombie processes?

2. (a) [**3 points**] Consider the execution of the following commands, what is the output of jobs?

```
#> sleep 200 &
#> head -c 10 /dev/urandom > rand
#> cat &
#> sleep 10
^Z
#> bg
```

```
[1] Running sleep 200 &
```

(b) [**3 points**] Based on the above commands, what shell command will bring `sleep 200` to the foreground?

(c) [**3 points**] For how many seconds does the following pipeline run for? Explain.

```
sleep 1000 | cat /proc/cpuinfo | grep processor | sleep 10
```

(d) [**3 points**] Using the following pipeline of commands and their `pid` (indicated in italics above each command), what is the `pgid` of each process in the pipeline? Will all processes terminate after a `Ctrl-C`? Why or why not?

```
   1994       1995       1996       1997
sleep 20 | sleep 30 | sleep 40 | sleep 50
```

(e) [**3 points**] At the mark, what process state would the parent process be in, Ready and Waiting, Ready and Running, Blocked and Waiting? Explain.

```
int main(){
 if(fork() == 0){
   while(1);
 }else{
   wait(NULL); //MARK
 }
}
```

3.

```
int main(int argc, char * argv[]){

  char str1[] = " Go Navy! ";
  char str2[] = " Beat Army! ";
  char buffer[1024];
  int pfd1[2], pfd2[2], n;


  pipe(pfd1);  pipe(pfd2);

  if( fork() == 0){ /* Child */

    //MARK 1
    close(pfd1[0]);
    close(pfd2[1]);

    //MARK 2
    write(pfd1[1], str1, strlen(str1));

    //MARK 3
    n = read(pfd2[0], buffer, 1024);
    write(1, buffer, n);

  }else{  /* Parent */

    //MARK 4
    close(pfd1[1]);
    close(pfd2[0]);

    //MARK 5
    n = read(pfd1[0], buffer, 1024);
    write(1, buffer, n);

    //MARK 6
    write(pfd2[1], str2, strlen(str2));

    wait(NULL);
  }

  return 0;
}
```

(a) **[3 points]** In the program to the right, two pipes are opened before `fork()` how is it that the child also has access to them to communicate with the parent?

(b) **[4 points]** In the program to the right, which system call will complete first, `MARK 2` in the child or `MARK 5` in the parent? Explain.

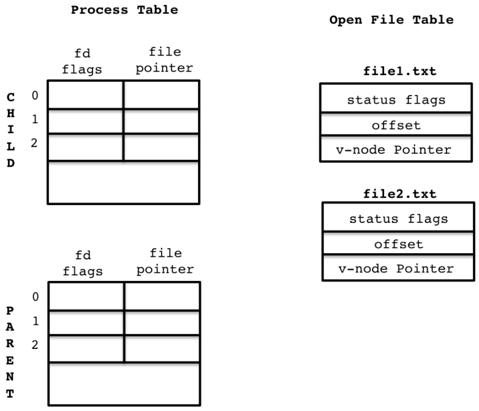
(c) **[4 points]** In the above program, what is `MARK 1` and `MARK 4` doing to the pipe? Why are the operations mirrored in parent and child?

(d) **[4 points]** Consider the following change in the program at `MARK 2` where `str1` is written continually to the pipe in a loop. Will the child ever reach `MARK 3`? If yes, why, and if no, why not?

```
//MARK 2
while(1){
 write(pfd[1], str1, strlen(str1));
}
```

4. Consider the following program

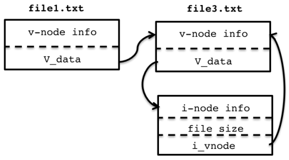
```
int main(){
  int fd1 = open("file1.txt", O_RDONLY);
  int fd2 = open("file2.txt", O_WRONLY | O_CREAT, 0640);
  pid_t cpid;
  char c;

  while(1){
    if( (cpid = fork()) == 0){
      /*Child*/

      close(1);
      dup2(fd2,1);

      if(read(fd1, &c, 1) > 0){ //read 1 byte from file
        write(1, &c, 1); //write 1 byte to stdout
      }
      _exit(0); //exit hard
    }else{
      /*Parent*/
      wait(NULL); //wait for child to exit

      if( read(fd1, &c, 1) <= 0){//read 1 byte from file
        break; //exit on EOF or read fail
      }else{
        write(1, &c, 1);//otherwise write 1 byte to stdout
      }
    }
  }

  return 0;
}
```



(a) **[4 points]** Complete the figure of the Process Table and Open File Table entries for the above program, drawing arrows for references and including new file descriptor table entries. You only need to complete one of the children since they are all the same.

(b) **[4 points]** If `file1.txt` contained the phrase “Go_Navy!_Beat_Army!”, where “_” indicates a space, what is written to standard output and what is written to `file2.txt`? Explain.

(c) **[4 points]** If `file1.txt` was linked in the v-node/i-node structure like below, would the functionality of the program change if `file1.txt` were replaced by `file3.txt`? Explain, and what kind of link is this?



(d) **[3 points]** In which node, v-node or i-node, is device specific information stored for reading and writing the file from/to the device?

5. Consider the following program

```
int ticks=0;
void handler(int signum){
  ticks++;
  printf("tick tock: %d\n", ticks);


  if(ticks < 10){
    alarm(1);
  }else{
    raise(9); //<-- MARK 1
  }
}

int main(){

  signal(SIGALRM, handler); //<-- MARK 2
  alarm(1);

  //MARK 3
  while(1){
    pause(); //<-- MARK 3
  }
}
```

(a) [**3 points**] At `MARK 2`, what is the purpose of the call to `signal()` with respect to future deliveries of `SIGALRM` from the O.S.?

(b) [**4 points**] What is the result of the system call at `MARK 1`? What is the *name* of the signal being raised?

(c) [**4 points**] How many `SIGALRM` signals are delivered to the program? Explain.

(d) [**4 points**] What is the purpose of `pause()` at `MARK 3`? Is it better or worse then replacing it with the following code without `pause()`?

```
//MARK 3
while(1);
```