Name: ID#: Lab Time:

CS 4061: Practice Exam 1 SOLUTION

Spring 2019
University of Minnesota

Exam period: 30 minutes
Points available: 40

Problem 1 (10 pts): Examine the code to the right and describe what you expect its output to be. Explain why or why not you would expect to see any specific ordering in the output of the program.

SOLUTION: The processes will fork out in a "line" or "list" rather than branching in a tree. This is because each parent process falls into the `if()` *consequence and will break out of the loop so will have only a single child. The order of output will start with the last child in the last iteration and proceed backwards to the original parent due to the placement of* `wait()`. *Example:*

```
> a.out
iter 4, 13374 from 13373
iter 3, 13373 from 13372
iter 2, 13372 from 13371
iter 1, 13371 from 13370
iter 0, 13370 from 26018
```

```
#include "headers.h" // standard headers
int main(){
  for(int i=0; i<5; i++){
    pid_t p = fork();
    if(p  != 0){
      wait(NULL);
      printf("iter %d, %d from %d\n",
             i,getpid(),getppid());
      fflush(stdout);
      break;
    }
  }
  exit(0);
}
```

Problem 2 (10 pts): Write the function `total_doubs()` described and demonstrated below.

```
#include <stdio.h>
int total_doubs(char *fname,      // file to read
                double *total);   // set to total
// Read doubles from the named file which contains
// binary doubles. Sum all the numbers in the file
// and set the double pointed to by total to this
// value.  Return the count of numbers or -1 if
// the file could not be opened.

int main(){
  char *fname = "nums.dat";
  int n_doubs = -1;
  double total = -1;
  n_doubs = total_doubs("nums.dat", &total);
  printf("%d nums read totaling %lf\n",
         n_doubs, total);
  return 0;
}
```

SOLUTION

```
#include "total_doubs.h"
#define BUFCOUNT 4
#define BUFSIZE (sizeof(double) * BUFCOUNT)

int total_doubs(char *fname, double *total){
  int fd = open(fname, O_RDONLY);
  if( fd == -1 ){
    perror("Couldn't open file");
    return -1;
  }
  double buf[BUFCOUNT], tot = 0.0;
  int total_bytes = 0;
  while(1){
    int bytes_read = read(fd, buf, BUFSIZE);
    if(bytes_read == 0){
      break;
    }
    total_bytes += bytes_read;
    int count = bytes_read / sizeof(double);
    for(int i=0; i<count; i++){
      tot += buf[i];
    }
  }
  close(fd);
  *total = tot;
  return (total_bytes / sizeof(double));
}
```

Problem 3 (5 pts): Stan Dardin is interactively testing his implementation of `commando` by punching in commands like `list, output-for 2, gcc test.c` by hand. He is finding that he keeps making mistakes while entering commands causing him to have to restart the program. He is not ready to run the full tests provided by Prof. Coffmalevolent, just the few things he knows works but the tedium is making it hard to proceed. Suggest an easy way for Stan to enter his commands more easily using tools and syntax provided in every Unix shell.

SOLUTION: Place the commands in a text file then use output redirection to feed the commands to commando. Either of these would work:

`./commando --echo < input.txt` *OR* `cat input.txt | ./commando --echo`

Background: If a `commando` user mistypes the name of a program like `fcc` rather than `gcc` strange output might occur. The next problems address this by making changes to `commando` so that it behaves like the example to the right.

```
> commando
@> fcc
@>
@!!! fcc[#9620]: EXEC FAIL
@> flarb
@>
@!!! flarb[#9621]: EXEC FAIL
@> list
JOB  #PID      STAT   STR_STAT OUTB COMMAND
0    #9620      128  EXEC FAIL   52 fcc
1    #9621      128  EXEC FAIL   52 flarb
@> output-for 0
@<<< Output for fcc[#9620] (52 bytes):
----------------------------------------
Cmd failed to start exec: No such file or directory
----------------------------------------
@> exit
>
```

Problem 4 (15 pts): Standard output for commando programs is directed into a pipe but standard error is not. Add code to `cmd_start()` in the template given which directs standard error into the output pipe associated with the `cmd_t`.
Add code to the template for `cmd_update_state()` that checks for special exit codes that signify exec problems and sets fields of the `cmd_t` appropriately. Some code for this problem may be identical to what is already in your project.

SOLUTION

```
#define EXEC_FAIL 128
void cmd_start(cmd_t *cmd){
  ...  // VARIOUS SETUP
  cmd->pid = fork();
  ...
  if(cmd->pid != 0){...}  // PARENT
  else{                   // CHILD
    // normal standard output redirection set up
    ...
    // YOUR CODE FOR STDERR REDIRECTION HERE
    dup2(cmd->output_pipe[PWRITE], STDERR_FILENO);

    // FINISHED WITH STDERR REDIRECTION

    // Call to execvp(), on error, call perror()
    // and exit with code EXEC_FAIL.
    // YOUR CODE FOR EXEC/RETURN HERE

    int ret = execvp(...);
    if(ret == -1){
      perror("Exec failed");
      exit(EXEC_FAIL);
    }

  }
}

```

SOLUTION

```
#define EXEC_FAIL 128
void cmd_update_state(cmd_t *cmd, int block){
  ...  // VARIOUS SETUP
  pid_t ret = waitpid(cmd->pid, &status, block);
  ...
  // CHECK IF status INDICATES CHILD HAS EXITED
  // SET DATA ASSOCIATED WITH IT APPROPRIATELY
  // CHECK FOR SPECIAL RETURN VALUE EXEC_FAIL
  // AND SET THE str_status FIELD TO "EXEC FAIL"
  // OTHERWISE SET DATA AS NORMAL IN PROJECT
  if(  ..check for exit as normal.. ){
    int exit_status =
      ..use macro to set status with exit code..;
    cmd->status = exit_status;
    cmd->finished = 1;

    if(exit_status == EXEC_FAIL){
      snprintf(cmd->str_status, STATUS_LEN, "EXEC FAIL");
    }

    else{
      ..set str_status as normal..;
    }

  }  // FINISHED CHECKING ON EXIT
  ... // PRINT ALERTS
}

```

(Note: In a normal exam, more space will be provided for answers.)