**CS3377 Assignment#3 Due: 7/27 Monday Noon.**

Last Name: \_\_\_\_\_\_Pulliam\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

First Name: \_\_\_\_\_\_\_Drew\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NetID (email): \_\_\_\_DTP180003\_\_\_\_\_\_\_

Submission requirements.

Submit (1) a word document (this file with your answers and listing of any program & copy and paste its run-log in a professional and presentable format and style) and (2) a zip file of a folder which will contain all the codes (all the source codes and executables, Makefile, and the instruction for how to compile and run) and a document file (.doc or .docx).

This word document (this file) is your documentation (as a basis to add your answers) to be included here - all your answers, all your program-listings, instructions to compile and run [screen-shots, terminal text, or session log] to show your work done, including how to compile and run for each cases.

The word file should also have (1) in header with the course & section number, your name (Name: last name, first name), your netID (email), and Assignment #, (2) in footer with page number, (3) line number (restart from each page in left margin. Page Layout => Line Numbers). (4) Your program should have some comments (minimal or reasonable), to tell the code-reader what a segment of code is doing, and with the comment in the front/head of the program about you, this course and assignment, etc. (you may copy and paste some out of this document for your comment).

\*\* Your executable codes (that you submit) should run in cs1.utdallas.edu without any change or recompilation.

\*\* Bring this cover sheet (this page) to TA for your demo. (Please check with TA for demo or schedule).

\*\* Upload this document file (with your answers) and a zip file (containing all the codes [source and binary etc.].

**Scoresheet**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Parts** |  |  |  |  |  |  |  |  |
| **Part1**  **40%** | Week07 Lab1  APUE7.6 size  APUE7.7 -static | Week07 Lab2 fork exercises | Week07 Lab3 shell - fork & exec | Week08 Lab1 shell w signal | Week08 Lab2 shell list&print | Week08 Lab3 git & github | Week09 Lab1 Thread Prog lock &  Reader-Writer | Week09 Lab2 Thread Prog  with many threads |
| CLO | 4,5,7 | 4,5,7 | 4,5,7 | 4,5,7 | 4,5,6,7,8 | 9 | 4,5,7, | 4,5,7 |
| **Part2**  **60%** | #1  a3prog1.c  timer & signal | #2  a3prog2.c  file redirection | #3  Makefile |  |  |  |  | **Demo &**  **Document** |
| CLO | 4,5,7 | 4,5,6,7,8 |  |  |  |  |  |  |
| **Total** |  |  |  |  |  |  |  |  |

Note: CLO means Student Course Learning Objective (as listed in Syllabus)

**Student Learning Objectives/Outcomes**

1. Ability to use the UNIX operating system interactively as a user (commands)

2. Ability to express algorithmic solutions using shell scripting (utilities)

3. Ability to understand and use regular expressions

4. Ability to use the UNIX programming environment (editor, compiler and linker)

5. Ability to understand UNIX processes (creation and control)

6. Ability to perform input/output of binary files

7. Ability to use interprocess communication (pipes, sockets and signals)

8. Ability to understand the UNIX file system

9. Ability to understand and use version control system

|  |
| --- |
| Deduction - Documentation (this .doc file) and upload  Max -10% if not done or poorly prepared |
| Deduction for Demo (Demo time-slots may be scheduled by TA later, for you to do the demo).  Max -10%. |

Note. Any "poor" documentation (that is, this document with your answers etc.) may result in a penalty (up to -10%).

Upload this document file (this file with your answers) and a zip file (containing all the codes [source and binary etc.] and its run log or results. All the code should run in cs1 without any change).

\*\* A demo (of your assignment) will be announced and scheduled by TA. Your demo should be done within the week of the due date. For any scheduling conflict, please contact or consult with TA (or the instructor) for your situation, and/or for alternate time or discretion as soon as possible, before the due.

**Assignment 2 Part1**

Part1 consists of weekly activity items (labs) in Week04, Week05, and Week06 Activity folders:

Week07 Lab1 – APUE7.6 size & APUE 7.7 -static

Week07 Lab2 – fork exercises

Week07 Lab3 – shell with fork & exec

Week08 Lab1 – Shell & signal

Week08 Lab2 – Shell to combine ls and cat

Week08 Lab3 – VCS with git and github

Week09 Lab1 – Thread Prog 1 with read-writer lock

Week09 Lab2 – Thread Prog 2 with multiple threads

Provide each of your lab (that you have prepared and submitted) in your lab document below (copy and paste below).

Have a page break for each lab to start. Provide a proper heading and subheading of each lab (and each part of the lab) in a presentable manner. You may view and take the example of the layout and format from the sample lab report.

For the log or output of your session (e.g., in cs1), you should use the font (Courier New) to have an alignment of the text and the font size (8 to 10) as shown below.

{cslinux1:~} cat hello.c

#include <stdio.h>

#include <stdlib.h>

int main()

{

// this is richard min

printf("Hello World\n"); // OK waw hello Richard and ok Richard

exit(0); // end of the program

}

{cslinux1:~} gcc hello.c -o hello

{cslinux1:~} ./hello

Hello World

Place your lab reports in the next page and thereafter.

**Your Lab reports to be placed here.**

**CS3377.0W1 Week 6 Lab 1**

**Drew Pulliam - DTP180003**

**(Documentation with Terminal Log)**

┌────────────────────────────────────────────────────────────────────┐

│ • MobaXterm 20.2 • │

│ (SSH client, X-server and networking tools) │

│ │

│ → SSH session to dtp180003@cslinux1.utdallas.edu │

│ • SSH compression : v │

│ • SSH-browser : v │

│ • X11-forwarding : v (remote display is forwarded through SSH) │

│ • DISPLAY : v (automatically set on remote server) │

│ │

│ → For more info, ctrl+click on help or visit our website │

└────────────────────────────────────────────────────────────────────┘

1. **Uncompress gzip file**

{cslinux1:~/cs3377} gunzip src.3e.tar.gz

**2. Unzip tar file (output trimmed)**

{cslinux1:~/cs3377} tar -xvf src.3e.tar

apue.3e/

apue.3e/advio/

apue.3e/daemons/

...

**3. Cd to apue.3e**

{cslinux1:~/cs3377} cd apue.3e/

**4. Make executables (output trimmed)**

{cslinux1:~/cs3377/apue.3e} make

for i in lib intro sockets advio daemons datafiles db environ fileio filedir ipc1 ipc2 proc pty relation signals standards stdio termios threadctl threads printer exercises; do \

(cd $i && echo "making $i" && make ) || exit 1; \

done

making lib

make[1]: Entering directory `/home/013/d/dt/dtp180003/cs3377/apue.3e/lib'

gcc -ansi -I../include -Wall -DLINUX -D\_GNU\_SOURCE -c -o bufargs.o bufargs.c

gcc -ansi -I../include -Wall -DLINUX -D\_GNU\_SOURCE -c -o cliconn.o cliconn.c

gcc -ansi -I../include -Wall -DLINUX -D\_GNU\_SOURCE -c -o clrfl.o clrfl.c

...

**6. Cd intro**

{cslinux1:~/cs3377/apue.3e} cd intro/

**7. Cat hello.c to see contents**

{cslinux1:~/cs3377/apue.3e/intro} cat hello.c

#include "apue.h"

int

main(void)

{

printf("hello world from process ID %ld\n", (long)getpid());

exit(0);

}

**8. run hello.c**

{cslinux1:~/cs3377/apue.3e/intro} ./hello

hello world from process ID 3956

**9. run shell1.c**

{cslinux1:~/cs3377/apue.3e/intro} ./shell1

% ls

getcputc hello ls1 Makefile mycat.c shell1.c shell2.c testerror.c uidgid.c

getcputc.c hello.c ls1.c mycat shell1 shell2 testerror uidgid

% ^C

{cslinux1:~/cs3377/apue.3e/intro}

**CS3377.0W1 Week 6 Lab 2**

**Drew Pulliam - DTP180003**

**(Documentation with Terminal Log)**

┌────────────────────────────────────────────────────────────────────┐

│ • MobaXterm 20.2 • │

│ (SSH client, X-server and networking tools) │

│ │

│ → SSH session to dtp180003@cslinux1.utdallas.edu │

│ • SSH compression : v │

│ • SSH-browser : v │

│ • X11-forwarding : v (remote display is forwarded through SSH) │

│ • DISPLAY : v (automatically set on remote server) │

│ │

│ → For more info, ctrl+click on help or visit our website │

└────────────────────────────────────────────────────────────────────┘

1. **Create apue01 directory**

{cslinux1:~/cs3377} mkdir apue01

{cslinux1:~/cs3377} cd apue01/

1. **Copy chapter 1 sample programs to this directory**

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.10" fig1.10.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.3" fig1.3.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.4" fig1.4.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.5" fig1.5.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.6" fig1.6.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.7" fig1.7.c

{cslinux1:~/cs3377/apue01} cp

"/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.9" fig1.9.c

{cslinux1:~/cs3377/apue01} cp "/home/013/d/dt/dtp180003/cs3377/apue.3e/figlinks/fig1.8" fig1.8.c

{cslinux1:~/cs3377/apue01} ls

fig1.10.c fig1.3.c fig1.4.c fig1.5.c fig1.6.c fig1.7.c fig1.8.c fig1.9.c

1. **Compile all chapter 1 sample programs to fig1.x.exe**

{cslinux1:~/cs3377/apue01} gcc fig1.3.c -o fig1.3.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.4.c -o fig1.4.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.5.c -o fig1.5.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.6.c -o fig1.6.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.7.c -o fig1.7.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.8.c -o fig1.8.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.9.c -o fig1.9.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} gcc fig1.10.c -o fig1.10.exe -I ~/cs3377/apue.3e/include/ -L ~/cs3377/apue.3e/lib/ -lapue

{cslinux1:~/cs3377/apue01} ls

fig1.10.c fig1.3.c fig1.4.c fig1.5.c fig1.6.c fig1.7.c fig1.8.c fig1.9.c

fig1.10.exe fig1.3.exe fig1.4.exe fig1.5.exe fig1.6.exe fig1.7.exe fig1.8.exe fig1.9.exe

1. **Run fig1.3 (basically ls) on current directory**

{cslinux1:~/cs3377/apue01} ./fig1.3.exe ./

.

..

fig1.10.exe

fig1.7.c

fig1.6.c

fig1.10.c

fig1.4.exe

fig1.9.exe

fig1.3.c

fig1.8.c

fig1.9.c

fig1.4.c

fig1.6.exe

fig1.3.exe

fig1.5.c

fig1.7.exe

fig1.5.exe

fig1.8.exe

1. **Use fig1.4 to write standard input to standard output (in this case write terminal to file named output.txt)**

{cslinux1:~/cs3377/apue01} ./fig1.4.exe > output.txt

hello this is my 1.4 test

^C

{cslinux1:~/cs3377/apue01} cat output.txt

hello this is my 1.4 test

1. **Use fig1.5 to copy output.txt from fig1.4 to a new file named output2.txt**

{cslinux1:~/cs3377/apue01} ./fig1.5.exe < output.txt > output2.txt

{cslinux1:~/cs3377/apue01} cat output2.txt

hello this is my 1.4 test

1. **Execute fig1.6 (hello world + process ID)**

{cslinux1:~/cs3377/apue01} ./fig1.6.exe

hello world from process ID 31512

1. **Execute fig1.7 (which is basically a shell) run ls to check that it works then exit**

{cslinux1:~/cs3377/apue01} ./fig1.7.exe

% ls

fig1.10.c fig1.3.c fig1.4.c fig1.5.c fig1.6.c fig1.7.c fig1.8.c fig1.9.c output2.txt

fig1.10.exe fig1.3.exe fig1.4.exe fig1.5.exe fig1.6.exe fig1.7.exe fig1.8.exe fig1.9.exe output.txt

% ^C

1. **Execute fig1.8 (basically writes out errors for you)**

{cslinux1:~/cs3377/apue01} ./fig1.8.exe

EACCES: Permission denied

./fig1.8.exe: No such file or directory

1. **Execute fig1.9 (print user ID and group ID)**

{cslinux1:~/cs3377/apue01} ./fig1.9.exe

uid = 633083, gid = 215

1. **Execute fig1.10 (also a shell like 1.7, but it gives an interrupt message when finished)**

{cslinux1:~/cs3377/apue01} ./fig1.10.exe

% ls

fig1.10.c fig1.3.c fig1.4.c fig1.5.c fig1.6.c fig1.7.c fig1.8.c fig1.9.c output2.txt

fig1.10.exe fig1.3.exe fig1.4.exe fig1.5.exe fig1.6.exe fig1.7.exe fig1.8.exe fig1.9.exe output.txt

% ^Cinterrupt

**CS3377.0W1 Week 6 Lab 3**

**Drew Pulliam - DTP180003**

**(Documentation with Terminal Log)**

┌────────────────────────────────────────────────────────────────────┐

│ • MobaXterm 20.2 • │

│ (SSH client, X-server and networking tools) │

│ │

│ → SSH session to dtp180003@cslinux1.utdallas.edu │

│ • SSH compression : v │

│ • SSH-browser : v │

│ • X11-forwarding : v (remote display is forwarded through SSH) │

│ • DISPLAY : v (automatically set on remote server) │

│ │

│ → For more info, ctrl+click on help or visit our website │

└────────────────────────────────────────────────────────────────────┘

**0. Create archive directory**

{cslinux1:~/cs3377} mkdir archive

{cslinux1:~/cs3377} cd archive/

**1. zip hello and hello.c, then delete the files and unzip them**

{cslinux1:~/cs3377/archive} ls

hello hello.c

{cslinux1:~/cs3377/archive} zip hello.zip hello hello.c

adding: hello (deflated 72%)

adding: hello.c (deflated 4%)

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.zip

{cslinux1:~/cs3377/archive} rm hello

rm: remove regular file ‘hello’? y

{cslinux1:~/cs3377/archive} rm hello.c

rm: remove regular file ‘hello.c’? y

{cslinux1:~/cs3377/archive} ls

hello.zip

{cslinux1:~/cs3377/archive} unzip hello.zip

Archive: hello.zip

inflating: hello

inflating: hello.c

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.zip

**2. Create a tar with hello and hello.c, then delete the files and restore them**

{cslinux1:~/cs3377/archive} tar -cvf hello.tar hello hello.c

hello

hello.c

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.tar hello.zip

{cslinux1:~/cs3377/archive} rm hello

rm: remove regular file ‘hello’? y

{cslinux1:~/cs3377/archive} rm hello.c

rm: remove regular file ‘hello.c’? y

{cslinux1:~/cs3377/archive} ls

hello.tar hello.zip

{cslinux1:~/cs3377/archive} tar -xvf hello.tar

hello

hello.c

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.tar hello.zip

**3. Use gzip to create .gz files with hello and hello.c, then restore the files**

{cslinux1:~/cs3377/archive} gzip hello hello.c

{cslinux1:~/cs3377/archive} ls

hello.c.gz hello.gz hello.tar hello.zip

{cslinux1:~/cs3377/archive} gzip -d hello.c.gz

{cslinux1:~/cs3377/archive} gzip -d hello.gz

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.tar hello.zip

**4. Compress hello.tar into hello.tar.gz using gzip**

{cslinux1:~/cs3377/archive} gzip hello.tar

{cslinux1:~/cs3377/archive} ls

hello hello.c hello.c.gz hello.gz hello.tar hello.tar.gz hello.zip

**5. Compare file sizes of different options**

{cslinux1:~/cs3377/archive} ls -l

total 408

-rw------- 1 dtp180003 sn 8512 Jul 4 18:54 hello

-rw------- 1 dtp180003 sn 100 Jul 4 18:54 hello.c

-rw------- 1 dtp180003 sn 122 Jul 4 18:54 hello.c.gz

-rw------- 1 dtp180003 sn 2379 Jul 4 18:54 hello.gz

-rw------- 1 dtp180003 sn 20480 Jul 4 19:08 hello.tar

-rw------- 1 dtp180003 sn 2626 Jul 4 18:58 hello.tar.gz

-rw------- 1 dtp180003 sn 2752 Jul 4 18:55 hello.zip

**6. Best compression observations**

Tar is the worst obviously, which is why it can be compressed using gzip. Tar.gz is slightly smaller than zip compression. Simply using gzip without first using tar results in separate files that are individually smallest, but it removes the convenience of having everything in one folder. For these reasons tar.gz is best, but zip is not much worse.

**Assignment 3 Part2**

**First, create a folder as shown below.**

\*\* Do this part using cs1.utdallas.edu

Use the directory named cs3377 (or create it if you have not created it yet). All of your work for cs3377 assignment(s) should be placed here for cs3377.

Second, create a subdirectory A3part2-netid in cs3377 directory (where netid is your UTD netid, for example, A3part2-rkm010300). All of your work for Assignment3 Part2 should be placed here.

You should be able to go to A2-netid directory by typing: cd $HOME/cs3377/A3part2-netid

Note. Some sample codes (programs and code segment) are available for this part at the end of this document. You should use these sample codes as your base code for this part. **Otherwise, your grade for this part could be resulted in 0.**

Task#1.

**Timer-Setup and Interrupt-Handler.**

Design and implement your simple shell program to set a timer to terminate the program after one minute being idle. That is, if there is no command input within one minute after the prompt, then the program will be interrupted by the timer and the timer-handler will terminate the program. Further the program should take an input command to set a timer interval.

The following command will set the timer for 30 seconds: timer=30

When the program got the timer interrupt (for example, after 30 seconds of idle time), it prints the message saying: "the program is terminated due to the idle time over 30 seconds."

Name your program: a3prog1.c

Note. The parent process sets the timer (interrupt-handler) at the time of (or after) the command prompt. If there is no command input, then the timer is still in effect. If there is a command input, then the timer should be cleared out.

For example (and for your test cases) - your current working directory is A3part1.

1. whoami

2. date

3. pwd

4. echo "Assignment3 Part2 start"

5. timer=30

6. ls

After this, wait and check whether the program is terminated after 30 seconds of the idle time, after the prompt.

Task#2

**A Simple Shell for File Redirection.**

Design and implement a simple shell program to read a command with a file redirection. Here we use the following file-redirection symbols as follow: "in=infile" for input file named "infile", "out=outfile" for output file named "outfile", "app=appfile" for output to be appended a file named "appfile". For example, "ls out=out.txt" will output the result of the command to a file named out.txt.

Name your program: a3prog2.c

Note. You should use the sample program (provided below) as your base code to implement this part.

**Warning: Do** **not** use any "system(…)" function call throughout this assignment.

For example (and for your test cases). Your current working directory is A3part1.

1. whoami

2. date

3. pwd

4. echo "Assignment3 Part1 #2 start"

5. ls

6. ls out=p1out1.txt

7. wc in=p1out1.txt

8. cat in=p1out1.txt

9. date out=p1out2.txt

10. date app=p1out2.txt

11. cat in=p1out2.txt

12. wc in=p1out1.txt out=p1out4.txt

13. cat in=p1out4.txt

14. echo "Assignment3 Part1 #2 done"

15. date

Task#3

(1) Provide a Makefile file to compile your program.

(2) After all done for this part, provide the result of the following commands here to show that you have done this part.

cd $HOME/cs3377/A3part2-netid

date

whoami

uname -a

ls -lR

# Part 2 - Solution

**Your working directory is: A3part2**

Place your answer here: (a) the program listing and (b) the result of the program run with a proper headings for each part. Upload (submit) this document and a zip file (containing all your codes and results of execution).

Task#1

1a. List of your program: a3prog1.c

|  |
| --- |
| // Sample code to set a timer to interrupt after some time (to terminate the run).  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <signal.h>  #include <unistd.h>  #define MINUTE 60  static int timer\_expired = 0;  static void alarm\_handler(int sig)  {  printf("The program is terminated after idle time of %d seconds\n", (timer\_expired)?timer\_expired:MINUTE);  timer\_expired = 0;  exit(EXIT\_SUCCESS);  }    int main()  {  struct sigaction mySigAction;  mySigAction.sa\_handler = alarm\_handler;  sigaction(SIGALRM, &mySigAction, NULL); //set SIGALRM action to mySigAction  alarm(MINUTE); // default timer of one minute  int input=0;  while(1)  {  printf("Enter command: ");  input = scanf("timer=%d", &timer\_expired);  if(input)  alarm(timer\_expired);  while(getchar()!='\n');  }  return 0;  } |

1b. List of your program run (session/program-run log/history)

|  |
| --- |
| {cslinux1:~/cs3377/A3part2-DTP180003} whoami  dtp180003  {cslinux1:~/cs3377/A3part2-DTP180003} date  Mon Jul 27 09:22:02 CDT 2020  {cslinux1:~/cs3377/A3part2-DTP180003} pwd  /home/013/d/dt/dtp180003/cs3377/A3part2-DTP180003  {cslinux1:~/cs3377/A3part2-DTP180003} echo "Assignment3 Part2 start"  Assignment3 Part2 start  {cslinux1:~/cs3377/A3part2-DTP180003} ./a3prog1  Enter command: timer=30  Enter command: The program is terminated after idle time of 30 seconds |

1c. List of your validation with "ls -lR" command to list all its files and directories.

|  |
| --- |
|  |

Task#2

2a. List of your program: a3prog2.c

|  |
| --- |
| /\* World's simplest shell â€“ shell0.c \*/  #include <sys/types.h>  #include <sys/wait.h>  #include <errno.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sysexits.h>  #include <unistd.h>  #include <fcntl.h>  #include <sys/stat.h>  void sig\_int(int signo) {  printf("\nCaught SIGINT!\n");  }  char\* getinput(char \*linefer, size\_t linelen) {  printf("myShell > ");  return fgets(linefer, linelen, stdin);  }  int main(int argc, char \*\*argv) {  char line[1024];  pid\_t pid;  int status;  if (signal(SIGINT, sig\_int) == SIG\_ERR) {  fprintf(stderr, "signal error: %s\n", strerror(errno));  exit(1);  }    void redirect(char\* redirection, char\* file){  int fdout;  //checks if read, write, append for stdin/stdout  if(!strcmp(redirection, "in=")){  if ((fdout = open(file, O\_CREAT|O\_RDONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 0);  close(fdout);  }  else if(!strcmp(redirection, "out=")){  if ((fdout = open(file, O\_CREAT|O\_WRONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 1);  close(fdout);  }  else if(!strcmp(redirection, "app=")){  if ((fdout = open(file, O\_CREAT|O\_APPEND|O\_WRONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 1);  close(fdout);  }  }  while (getinput(line, sizeof(line))) {  line[strlen(line) - 1] = '\0';  printf(" Input command is: %s \n", line);  if (strcmp(line, "exit") == 0) /\* is it an "exit"? \*/  exit(0);    if (strstr(request, "in=") != NULL) {  // contains    }    if((pid=fork()) == -1) {  fprintf(stderr, "shell: can't fork: %s\n",  strerror(errno));  continue;  } else if (pid == 0) {  /\* child process to run the command \*/  execlp(line, line, (char \*)0);  fprintf(stderr, "shell: couldn't exec %s: %s\n",  line, strerror(errno));  exit(EX\_DATAERR);  }  // parent waits till the child process done.  if ((pid=waitpid(pid, &status, 0)) < 0)  fprintf(stderr, "shell: waitpid error: %s\n",  strerror(errno));  }  exit(EX\_OK);  } |

2b. List of your program run (session/program-run log/history)

|  |
| --- |
| myShell > whoami  Input command is: whoami  dtp180003  myShell > date  Input command is: date  Mon Jul 27 11:28:59 CDT 2020  myShell > pwd  Input command is: pwd  /home/013/d/dt/dtp180003/cs3377/A3part2-DTP180003 |

Task#3. List of Makefile

|  |
| --- |
| a3prog2.o: a3prog2.c  gcc -o a3prog2 a3prog2.c  a3prog1.o: a3prog1.c  gcc -o a3prog1 a3prog1.c |

After all done for this part, provide the result of the following commands here to show that you have done this part.

cd $HOME/cs3377/A3part2-netid

date

whoami

uname -a

ls -lR

Copy and paste the run-log of your command and its output here (console output for this task)

|  |
| --- |
| {cslinux1:~/cs3377/A3part2-DTP180003} cd $HOME/cs3377/A3part2-DTP180003  {cslinux1:~/cs3377/A3part2-DTP180003} date  Mon Jul 27 11:32:24 CDT 2020  {cslinux1:~/cs3377/A3part2-DTP180003} whoami  dtp180003  {cslinux1:~/cs3377/A3part2-DTP180003} uname -a  Linux cslinux1.utdallas.edu 3.10.0-1127.8.2.el7.x86\_64 #1 SMP Tue May 12 16:57:42 UTC 2020 x86\_64 x86\_64 x86\_64 GNU/Linux  {cslinux1:~/cs3377/A3part2-DTP180003} ls -lR  .:  total 288  -rwx--x--x 1 dtp180003 sn 8704 Jul 27 09:32 a3prog1  -rw------- 1 dtp180003 sn 900 Jul 27 09:18 a3prog1.c  -rwx--x--x 1 dtp180003 sn 9128 Jul 27 11:15 a3prog2  -rw------- 1 dtp180003 sn 2070 Jul 27 11:15 a3prog2.c  -rw------- 1 dtp180003 sn 98 Jul 27 11:13 Makefile  {cslinux1:~/cs3377/A3part2-DTP180003} |

Submit two files: (1) this word document with your solution and (2) a zip file of A2-netid (containing all the programs, output files, and log of program run).

Warning. Your program should not use any of "system( … )" calls for this part (otherwise, your grade for this part will be 0).

Any "poor" documentation (this document with your answer) may result in a penalty (up to 10%).

Any "missing" submission of the files (including the zip file for this part) may result in 0 for this assignment grade.

\*\* A demo will be announced and scheduled by TA. For any scheduling conflict, please contact or consult with TA (or the instructor) for your situation, and/or for alternate time or discretion as soon as possible, before the due.

Hints for Part2

A sample code to set a timer for 10 seconds.

|  |
| --- |
| // Sample code to set a timer to interrupt after some time (to terminate the run).  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <signal.h>  #include <unistd.h>  static int timer\_expired = 0;  static void alarm\_handler(int sig)  {  timer\_expired=10;  }    int main()  {  // Set up a signal hander to call alarm\_handler()  // when SIGALRM is raised  sigaction(SIGALRM, &(struct sigaction){.sa\_handler = alarm\_handler}, NULL);    time\_t t;  srand((unsigned) time(&t));    timer\_expired = 0;  alarm(1); // raise SIGALRM in 1 seconds  while (!timer\_expired) {  int s=(rand()%4)+1;  printf("%d", s);  }  return 0;  }  // check APUE Chapter 10 Section 10 & Figure 10.1 for SIGALRM for timer |

The sample programs for your base code for this part.

The first sample code is a simple shell. Warning. You should use the sample code, as your base code, provided for this part. Otherwise, your score will be set to be 0.

|  |
| --- |
| /\* World's simplest shell – shell0.c \*/  #include <sys/types.h>  #include <sys/wait.h>  #include <errno.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sysexits.h>  #include <unistd.h>  void sig\_int(int signo) {  printf("\nCaught SIGINT!\n");  }  char\* getinput(char \*linefer, size\_t linelen) {  printf("myShell > ");  return fgets(linefer, linelen, stdin);  }  int main(int argc, char \*\*argv) {  char line[1024];  pid\_t pid;  int status;  if (signal(SIGINT, sig\_int) == SIG\_ERR) {  fprintf(stderr, "signal error: %s\n", strerror(errno));  exit(1);  }  while (getinput(line, sizeof(line))) {  line[strlen(line) - 1] = '\0';  printf(" Input command is: %s \n", line);  if (strcmp(line, "exit") == 0) /\* is it an "exit"? \*/  exit(0);    if((pid=fork()) == -1) {  fprintf(stderr, "shell: can't fork: %s\n",  strerror(errno));  continue;  } else if (pid == 0) {  /\* child process to run the command \*/  execlp(line, line, (char \*)0);  fprintf(stderr, "shell: couldn't exec %s: %s\n",  line, strerror(errno));  exit(EX\_DATAERR);  }  // parent waits till the child process done.  if ((pid=waitpid(pid, &status, 0)) < 0)  fprintf(stderr, "shell: waitpid error: %s\n",  strerror(errno));  }  exit(EX\_OK);  } |

And here is the sample code for your base code for file-redirection – "<" for input, ">" for output, and ">>" for append. You need to modify this sample for this part.

|  |
| --- |
| void redirect(char\* redirection, char\* file){  int fdout;  //checks if read, write, append for stdin/stdout  if(!strcmp(redirection, "<")){  if ((fdout = open(file, O\_CREAT|O\_RDONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 0);  close(fdout);  }  else if(!strcmp(redirection, ">")){  if ((fdout = open(file, O\_CREAT|O\_WRONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 1);  close(fdout);  }  else if(!strcmp(redirection, ">>")){  if ((fdout = open(file, O\_CREAT|O\_APPEND|O\_WRONLY)) < 0) {  perror(file); /\* open failed \*/  }  dup2(fdout, 1);  close(fdout);  }  } |