Pipes

You may work in groups of up to 3 students for this lab.

Instructions

1. Please make a copy of this lab (in google drive) and provide your answers as a group of 3 in the designated areas below. Please provide your name and email address in the slots below.

Name	Email
Bhanoday Mandla	bmandla@umass.edu
Sahil Kunati	skunati@umass.edu

2. Complete each of the sections below.
3. At the end of the lab, download a PDF version of this document and submit the PDF to the correct assignment in Gradescope. Please see Submitting an Assignment and Adding Group Members to ensure that you are submitting properly.

Before you do anything, make a copy of this google document. You will be required to submit a PDF

Lab Profile

The goal of this lab is to become familiar with the basics of the **pthread_create()** function. As you know, processes are created with the **fork()** system call. This allows the creation of concurrently executing processes. There are several limitations (rightly so) in the process model. Threads are another form of concurrent execution. This lab introduces you to the basic creation of threads.

Rubric

You will be scored from 0-4 for this lab. 0 indicates that you did not submit or a group did not include you in the submission. 4 indicates that your team completed most requirements. The following table summarizes how we will score you team lab submission.

Exceeding = 4	Meeting = 3	Approaching = 2	Beginning = 1	No Submission = 0
● Completed between 90-100% of the requirements ● Delivered on time, and in correct format	● Completed between 80-90% of the requirements ● Delivered on time, and in correct format	● Completed between 70-80% of the requirements ● Delivered on time, and in correct format	● Completed less than 70% of the requirements. ● Delivered on time, and in correct format	● No submission

Part 1: Research

Take a moment to read through the manual for pipe(2). This manual page describes the **pipe** function which creates a new "pipe", a unidirectional data channel that can be used for interprocess communication. This lab will focus on reviewing the

documentation of this function and examining an example program. Talk with your group about what you read in the documentation for this function. Provide in the answer box below any notes from the manual page that are interesting to you. specifically mention the return value/behaviour, and a brief (1-2 lines) summary of the function's description. The goal of this question is to try to understand what this system call does. You are welcome to research more details on the web to see what you come up with.

On success, zero is returned. On error, -1 is returned and pipefd is left unchanged

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

Part 2: Explore

Review the following code example:

```c
int child_task(int read_pipe) {
  char to_msg[25];
  int nr;

  if ((nr = read(read_pipe, to_msg, 25)) == -1) {
    perror("child failed to read from pipe");
    return 1;
  }

  printf("child received message from parent.\n");
  printf("bytes: %d\n", nr);
  printf("message received: %s\n", to_msg);

  return 0;
}

int parent_task(int write_pipe) {
  int nw;
  char from_msg[] = "hello from parent.";

  if ((nw = write(write_pipe, from_msg, strlen(from_msg) + 1)) == -1) {
    perror("parent failed to write to pipe");
    exit(1);
  }

  printf("parent sent message to child.\n");
  printf("message sent: %s.\n", from_msg);
  printf("bytes written: %d\n", nw);

  return 0;
}
```

```c
int main() {
  int parent_to_child_pipe[2];

  // Create the pipe
  if (pipe(parent_to_child_pipe) == -1) {
    perror("could not create pipe");
    exit(1);
  }

  pid_t pid;
  if ((pid = fork()) == 0) {
    // Child process...
    // First we close the write end of the pipe.
    close(parent_to_child_pipe[1]);
    exit(child_task(parent_to_child_pipe[0]));
  } else {
    // Parent process...
    // First, we close the read end of the pipe.
    close(parent_to_child_pipe[0]);
    exit(parent_task(parent_to_child_pipe[1]));

    // Wait on child process to complete
    wait(NULL);
    printf("parent process completing.\n");
  }
}
```

Create a new C file called **lab.c** in your course programming environment (vagrant) with the contents of the code above. You will probably want to type this in directly (rather than copy and paste) as your editor may not recognize all the symbols. After you type in the program into a corresponding C file, you must compile the code with the following command:

$ **gcc lab.c -o lab**

You will notice that it does not compile because of missing symbols. Figure out which header files you need to include in order to get this program to compile. You should start with the man page for pipe(2). After you are able to get this program to compile, run the program to observe the results:

$ **./lab**

In the answer slot below, describe to the best of your understanding, what this code is doing.

The code uses a pipe to send a message between processes.
The parent process writes a message to the pipe which is then read by the child process.

Part 3: Apply

Lastly, make a change to your program such that it can send multiple messages from the parent to the child process. You have the freedom to do this in any way you would like. The two messages you should send to the child process from the parent are "Thank you for the money!" and "Wish you were here in Paris with me". What changes were you required to make to this program work properly? Please provide a short response in the answer slot below.

Added another pipe and sent 2nd message through the 2nd pipe. Had to write twice and read twice.

Submission

You must submit a PDF file of this document to Gradescope by the assigned due date.

Note: All in class and lab activities for this course must be submitted through Gradescope. You must **submit your work as groups** and depending on the assignment you may need to submit either a PDF or code. Group members can be easily added through the Gradescope interface after the submission has been uploaded. You should spend some time reviewing the Student Workflow on the Gradescope website to better understand the submission process. There is also a video tutorial on how to submit a PDF-based assignment that might be helpful.