# CS347m Operating Systems
# Spring 2020-21
# Lab 1

1. This lab consists of questions related to processes and basic system calls.
2. **Write the programs in C** and follow the guidelines mentioned in the questions strictly. Not following the instructions may result in deduction in the marks. If your code does not compile, then straight up zero marks for that question.
3. Strictly follow the submission guidelines mentioned on the last page.
4. Total Marks: 20
5. **Due date: 29th January, 11.55 pm (via Moodle)**

---

### Some Basics

*System calls* are required for the problems of this lab. System calls are a mechanism for user programs to interact and request services from the operating system. The system call interface is provided as wrapper functions as part of the standard C library.

You can read the man pages for the system call functions for information on semantics and usage of each.
Use the command "**man <system call>**" on Linux. For example, man fork
An (incomplete) list of useful system calls for this lab are as follows:

1. fork
2. wait
3. getpid
4. getppid
5. exec, execvp, execve, execl
6. open
7. close
8. read
9. write
10. waitpid

Two C files, runme.c and sample.c, are provided in the Lab1 handout. The two C files will be used in problems of the Lab.

### Common command line operations

- `gcc [programName].c -o outputName`

The above command creates an executable with name outputName which needs to be executed .

- `./outputName`

To run the executable file

---

**Questions**

Q1.a) **Baby steps 1.0** **[Marks: 1.0]**

Write a program **p_1a.c** that forks a child. Both parent and child processes should print messages.

The parent process should print:

Parent: My process ID: <parent pid>

Parent: Child's process ID: <child pid>

The child process should print:

Child: My process ID: <child pid>

Child: Parent's process ID: <parent pid>

**Running Commands:**

gcc p_1a.c -o p_1a

./p_1a

**Sample Output :**

Parent: My process ID: 9648

Parent: Child's process ID: 9649

Child: My process ID: 9649

Child: Parent's process ID: 9648

Note : Add code in the "parent" part after printing that waits for any character as an input. Give input only after the child has given it's output. This will ensure that the child gets a correct parent PID.

Q1.b) **Baby steps 2.0** **[Marks: 2.0]**

Write a program **p_1b.c** that does exactly the same as the previous question, but the parent must print its message only after the child has terminated. The parent must **wait** for the child to terminate, and then print its message.

The child should print:

Child process ID: <child pid>

The parent should print:

The child process with process ID <child pid> has terminated.

**Running Commands:**

gcc p_1b.c -o p_1b

./p_1b

**Sample Output :**
Child process ID: 24116
The child process with process ID 24116 has terminated.

Q2) **Mixing things up** **[Marks: 3.0]**
Write a program **p_2.c** that creates 1 parent process and 3 child process and prints

I am parent with PID 1234
I am child with PID 4532 from PPID 1234
I am child with PID 4543 from PPID 1234
I am child with PID 4534 from PPID 1234

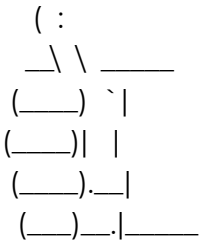
**Running Commands:**
gcc p_2.c -o p_2
./p_2

**Sample Output :**
I am parent with PID 1234
I am child with PID 4532 from PPID 1234
I am child with PID 4543 from PPID 1234
I am child with PID 4534 from PPID 1234

Q3) **To exec is to compute** **[Marks: 3.0]**
Provided in the handout is a c file, **runme.c**, which prints an ascii art as given in the sample output of this question. Write a program **p_3.c** that forks a child and the child process **executes** the binary program of **runme.c** (compile runme.c beforehand). The parent process waits for the child process to terminate.

**Running Commands:**
gcc runme.c -o runme
gcc p_3.c -o p_3
./p_3

**Sample Output :**



Q4) **Let there be write!** **[Marks: 3.0]**

Write a program p_4.c which takes a filename as command line argument. The program should **open** the file, and then fork a child. Both parent and the child should **write** contents to the file. Verify that the child can write to the file without opening it.

The parent should write
Hello world! I am parent
The child should write
Hello world! I am child

The parent should wait for the child to terminate and print the child exit message
The child process with process ID <child pid> has terminated.

**Running commands:**
gcc p_4.c -o p_4
./p_4 <filename>

**Sample Output:**
After running "./p1_4 hello.txt", the file hello.txt will contain:

Hello world! I am parent
Hello world! I am child

Q5) **Input redirection magic** **[Marks: 3.0]**

Write a program **p_5.c** that takes a filename as command line argument. The program opens the file and then forks a child. The objective here is to print the contents of the file to stdout. The child process should execute the binary program of **sample.c** (compile it beforehand), it reads input from stdin and prints to stdout. The parent should wait for the child to terminate before exiting.

**Constraints**: Parent of the child **cannot** use printf, scanf, read, write, cin.
Hint1: **close** of a file results in its descriptor to be used in subsequent open.
Note: No changes to be made in **sample.c**

**Running commands:**
gcc sample.c -o sample
gcc p_5.c -o p_5
./p_5 <filename>
**Sample Output:**
Say a file “abc.txt” contains
Hello world! This is abc.txt
After running the command “./p1_5 abc.txt”, the terminal stdout will be
Hello world! This is abc.txt

Q6) **Orphan Process** **[Marks: 2.0]**
Write a program **p_6.c** that forks a child. The child should print
Child: Child process ID: <child pid>
Child: Parent process ID: <parent pid>
and then sleep for **2 seconds** and print the same message again after waking up. The parent should wait for **1 second** and then print
Parent: Parent process ID: <parent pid>
and exit.
**Running Commands:**
gcc p_6.c -o p_6
./p_6

**Sample Output :**
Child: Child process ID: 20967
Child: Parent process ID: 20966
Parent: Parent process ID: 20966
Child: Child process ID: 20967
Child: Parent process ID: 1849

Q7) **Zombie Process** **[Marks: 3.0]**
Write a program **p_7.c** that forks a child. The parent should print its process ID as “Parent: <pid>” and then sleep for 1 minute, and then wait for the child process to exit and print “Exiting Parent: <pid>” at the end of parent execution. The child process should print its process ID as “Child: <pid>” and wait for keyboard input.

Check status of a process using the command **ps -o pid,stat <pid>**

Check the status of the child process **while it is waiting for the keyboard input and after providing keyboard input**. Copy the ps command outputs to a p_7.txt file. The final p_7.txt file should have two lines of text, corresponding to the ps command outputs (1) before keyboard input is provided (2) after keyboard input is provided.

**Running Commands:**

```
gcc p_7.c -o p_7
./p_7
```

**Sample solution text in  p_7.txt :**

```
85194 S+
85194 Z+
```

## Submission Guidelines

1. Make a new directory with your roll number as its name.
2. Create a README file and place any references to external websites/videos...etc that you have referred for this assignment, along with any instructions specific to compiling and running your submission (if any).
3. Add the honor code text(given on the next page) in HonorCode.txt
4. Place your submission files

   The directory structure look something like this:

```
[roll_number-lab1]
  |
  +-- README
  +-- HonorCode.txt
  +-- p_1a.c
  +-- p_1b.c
  +-- p_2.c
  +-- p_3.c
  +-- p_4.c
  +-- p_5.c
  +-- p_6.c
  +-- p_7.c
  +-- p_7.txt
```

5. Compress the main submission directory using

   ```
   tar -cvf [roll_number-lab1].tar.gz [roll_number-lab1]
   ```

6. Submit the compressed tar file to Moodle.

**(PS: Make sure to see the final page (next page) in this doc !!!)**

***** IMPORTANT *****

Please cross-check that your tar file has all the necessary files in it (and that they are the ones you want to submit) before submitting. You can check it by extracting the tar file by doing the following:

```
tar -xvf [roll_number-lab1].tar.gz
```

**Cribs related to code changes, compilation errors, missing files, corrupt files etc. will not be entertained after the deadline has passed.**

## Honor Code:

**The following text needs to be added in HonorCode.txt**

I, _____, confirm that my submission has no material, in total or in parts, that is plagiarized or copied.
All the work (documents, code) that I am submitting is my own and is prepared and written by me alone.
None of the submission is copy in full or in parts of any submission that does not belong to and has been prepared by me.

**Note: Discussions with peers and study of material from books and online resources is acceptable and encouraged.**