CS 365 Operating Systems & Networking
Assignment #6
Creating Processes in Unix

INSTRUCTIONS

In this challenge you will practice creating processes using the fork system call and coordinating the parent and child execution.

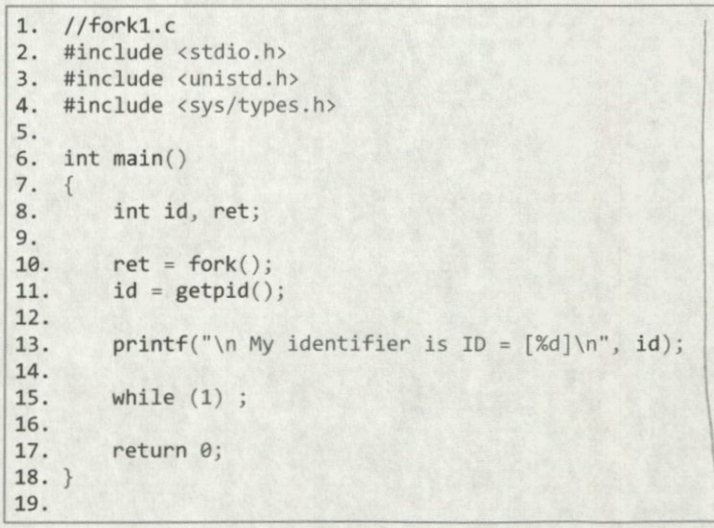
For exercises 1 - 7, pay careful attention to the output of the code and take the time to understand what happens. I highly recommend verifying your understanding by writing, compiling, and executing the code in your environment.

For exercises 8 and 9, write the code and capture your output.

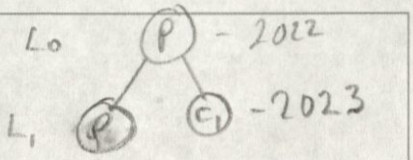
SUBMISSION INSTRUCTIONS

Create a Word document or PDF with your answers and submit to Canvas by the date/time specified.

1. Explain what happens in the following code snippet.



```c
//fork1.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    int id, ret;

    ret = fork();
    id = getpid();

    printf("\n My identifier is ID = [%d]\n", id);

    while (1) ;

    return 0;
}

```



```
*****************************************************
Compile:                    gcc -o xfork1 fork1.c
Run in background:          ./xfork1 &
List Processes:             ps -f
Kill the processes:         kill -9 process_id1
                            kill -9 process_id2
*****************************************************
```

Output (trace the code to understand the output):

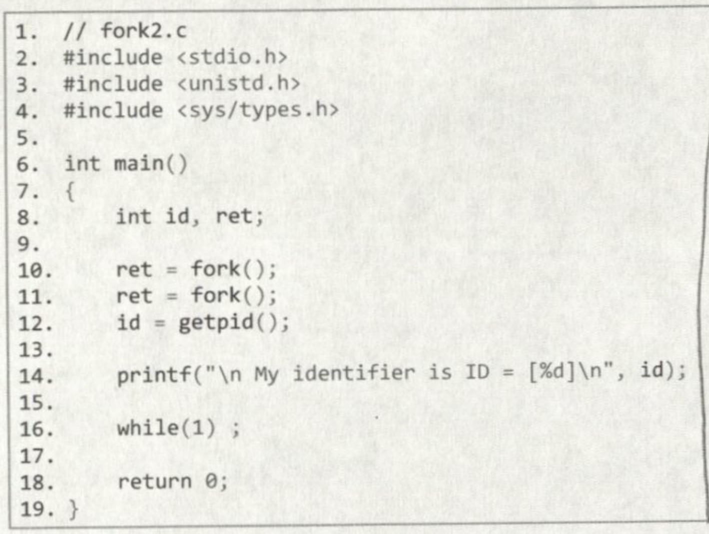
could be swapped

OUTPut

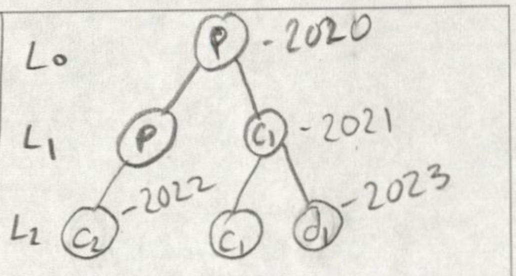
my ID is 2022

my ID is 2023

2. Explain what happens in the following code snippet.



```
// fork2.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    int id, ret;

    ret = fork();
    ret = fork();
    id = getpid();

    printf("\n My identifier is ID = [%d]\n", id);

    while(1) ;

    return 0;
}
```



```
****************************************************
Compile:                 gcc -o xfork2 fork2.c
Run in background:       ./xfork2 &
List Processes:          ps -f
Kill your processes as before
****************************************************
```

Output (trace the code and draw the tree of processes to understand the output):

Output

My ID is 2020

My ID is 2021

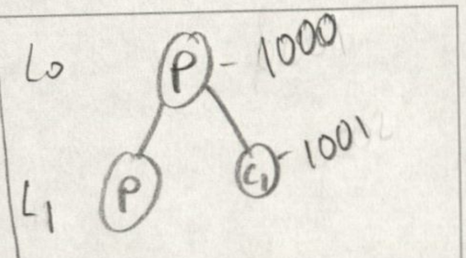
My ID is 2022

My ID is 2023

could be any order

3. Explain what happens in the following code snippet.

```c
// fork3.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork3()
{
     int ret;

     ret = fork();

     if (ret == 0)
          printf("\n [%d] Hello from child", getpid());
     else
          printf("\n [%d] Hello from parent", getpid());
}

int main ()
{
     fork3();
     return 0;
}
```



```
*************************************************
Compile:                  gcc -o xfork3 fork3.c
Run:                      ./xfork3
*************************************************
```

Output (trace the code and draw the tree of processes to understand the output):

1000 Hello from parent

1001 Hello from Child

Could be any order

4. Explain what happens in the following code snippet.

```
// fork4.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork4()
{
     printf("\n [%d] L0 \n", getpid());
     fork();
     printf("\n [%d] L1 \n", getpid());
     fork();
     printf("\n [%d] Bye \n", getpid());
}

int main ()
{
     fork4();
     return 0;
}
```

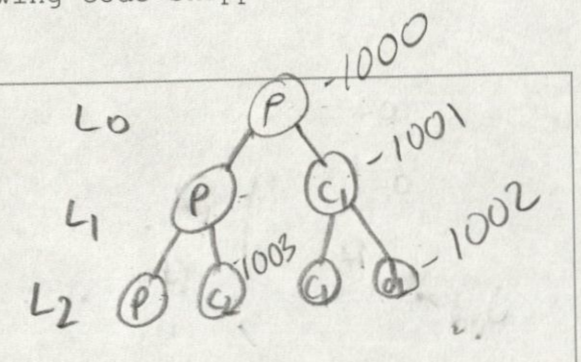
L0 1000

L1 1000

Bye 1000

L1 1001

how [illegible]



```
*************************************************
Compile:                  gcc -o xfork4 fork4.c
Run:                      ./xfork4
*************************************************
```

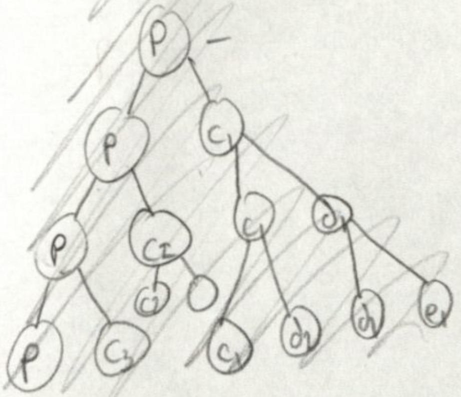
Output (trace the code and draw the tree of processes to understand the output):

ex)

ret = fork();

ret = fork();

ret = fork();



Output

1000 L0 - Prints first

1000 L1

1000 Bye

1001 L1

1001 Bye

1002 Bye

1003 Bye

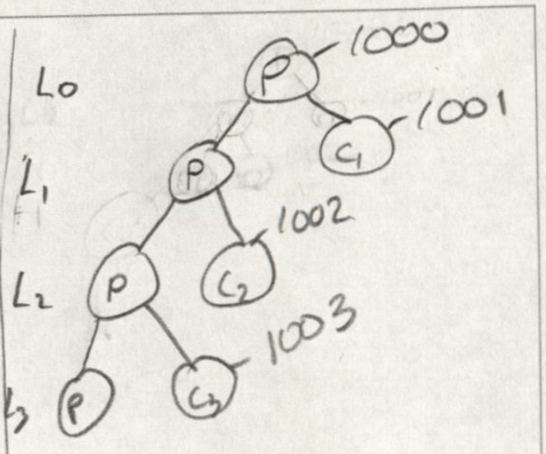
rest Could be any order

5. Explain what happens in the following code snippet.

ex) of a fail

why would it fail?

```
// fork5.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork5()
{
    printf("\n[% d] L0 \n", getpid());
    if (fork() != 0)  - if fails/parent
    {
        printf("\n[% d] L1 \n", getpid());
        if (fork() != 0)
        {
            printf("\n[% d] L2 \n", getpid());
            fork();
        }
    }
    printf("\n[% d] Bye \n", getpid());
}

int main()
{
    fork5();
    return 0;
}

```



****

```
Compile:                gcc -o xfork5 fork5.c
Run:                    ./xfork5
```

****

Output (trace the code and draw the tree of processes to understand the output):

output

1000 L0 ← prints first

1001 Bye

1000 L1

1002 Bye

1000 L2

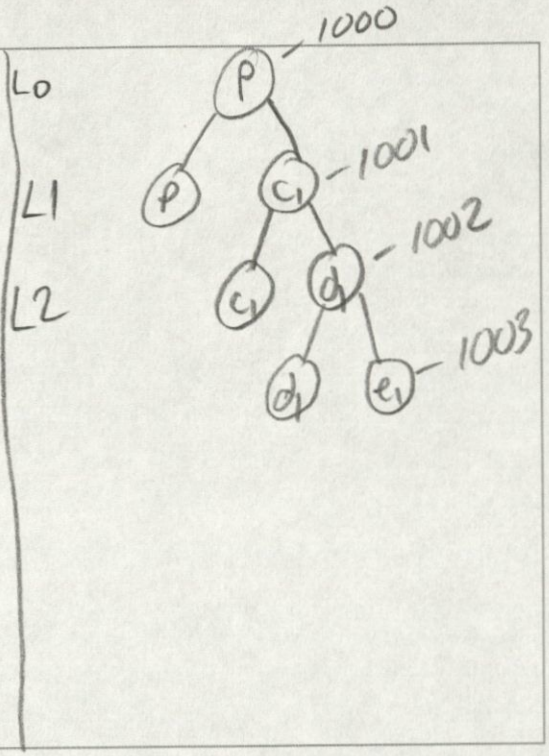
1000 Bye

1003 Bye

rest prints any order?

6. Explain what happens in the following code snippet.

```c
// fork6.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void fork6()
{
    printf("\n[% d] L0 \n", getpid());
    if (fork() == 0)
    {
        printf("\n[% d] L1 \n", getpid());
        if (fork() == 0)
        {
            printf("\n[% d] L2 \n", getpid());
            fork();
        }
    }
    printf("\n[% d] Bye \n", getpid());
}

int main()
{
    fork6();
    return 0;
}
```



```
*************************************************
Compile:                gcc -o xfork6 fork6.c
Run:                    ./xfork6
*************************************************
```

Output (trace the code and draw the tree of processes to understand the output):

<u>Output</u>

1000 L0 ← prints first

1000 Bye

1001 L1

1001 Bye

1002 L2

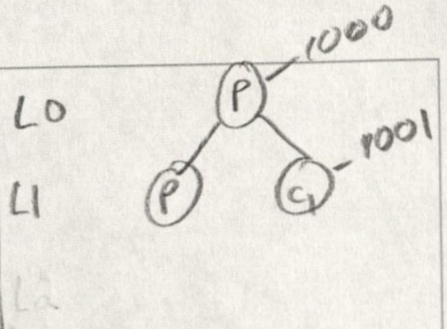
1002 Bye

1003 Bye

rest print Any order

7. Explain what happens in the following code snippet.

```
// fork7.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

void fork7()
{
    int ret;
    ret = fork();

    if (ret == 0)
    {
        printf("\n [%d] Running Child \n", getpid());
        sleep(2);
        printf("\n [%d] Ending Child \n", getpid());
    }
    else
    {
        printf("\n [%d] Waiting Parent \n", getpid());
        wait(NULL);
        printf("\n [%d] Ending Parent \n", getpid());
    }
}

int main()
{
    fork7();
    return 0;
}

```



```
*****************************************************
Compile:                    gcc -o xfork7 fork6.c
Run:                        ./xfork7
*****************************************************
```

Output (trace the code and draw the tree of processes to understand the output):

Output

1001 Running Child } Any order and possibly 1001 also

1000 Waiting Parent }

1001 Ending Child ← might come after "1000 waiting parent" depends on how long sleep(2) takes

1000 Ending Parent ← will be last since it must wait for 1001 (Child) to terminate.