CSC 262 Homework #2
Creating Processes in Unix

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

Open a Ubuntu shell using Multipass, WSL, or Crostini. In your systems directory create a directory called `unixfork`. Change your current directory to `~/systems/unixfork`, then complete the exercises below.

Pay careful attention to the output of the code for each of the first seven exercises, and take the time to understand what happens. Then write the code for the last two exercises.

SUBMISSION INSTRUCTIONS

Leave your code for all exercises in your `~/systems/unixfork` directory. Do not make any changes to these files after the due date for this assignment. Be ready to demonstrate your code to your instructor upon request.

If you wish to continue working on these programs after the due date, make a copy of your directory `unixfork` using the following Unix command:

```
cp -r ~/systems/unixfork ~/systems/unixfork-copy
```

A new directory called `unixfork-copy` will be created in your `systems` directory. You may now make any changes you want to the files in your `unixfork-copy` directory, at any time.

WHAT TO SUBMIT

1. Answers to questions 1 – 7, including process trees for 2 – 7, as an annotated pdf, docx, scanned image, etc.
2. Source code for question 8 as a .c file
3. Source code for question 9 as a .c file

Exercise 1 – What happens?

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):

Exercise 2 – What happens?

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):

Exercise 3 – Parent vs. Child Process

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):

Exercise 4 – Parent and Child Continue fork-ing

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;
}
```

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

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

Exercise 5 – Parent Continues fork-ing

fork5.c

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

void fork5()
{
	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 ()
{
	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):

Exercise 6 – Child Continues fork-ing

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)::

Exercise 7 – Synchronizing Parent with Child

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 fork7.c
Run:                 ./xfork7 &
List Processes:      ps –f
Repeat List:         ps –f
****************************************************
```

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

Exercise 8 – Programming with fork

Write a C program called `sumfact.c` that does the following:

1. Takes an integer argument (say, **`N1`**) from the command line.
2. Forks two children processes
   a. First child computes **`1+2+...+N1`** (sum of positive integers up to **`N1`**) and prints out the result and its own identifier.
   b. Second child computes **`1*2*...*N1`** (the factorial of **`N1`**) and prints out the result and its own identifier.
3. Parent waits until both children are finished, then prints out the message "***Done***" and its own identifier.

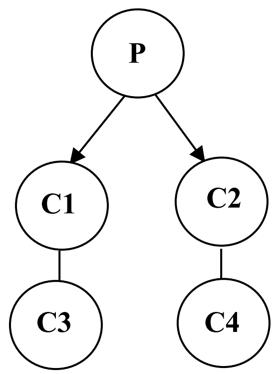
Sample execution (assuming the executable is called `xsumfact`):

```
bash$ ./xsumfact  5

[ID = 101] Sum of positive integers up to 5 is 15
[ID = 102] Factorial of 5 is 120
[ID = 100] Done
```

Exercise 9 [Process Tree]

Write a program `tree.c` that creates the tree of processes illustrated below. Each process in the tree should print its own identifier.



Sample execution (assuming the executable is called `xtree`):

```
bash$ ./xtree

[ID = 100] I am the root parent
[ID = 101] My parent is [100]
[ID = 102] My parent is [100]
[ID = 103] My parent is [102]
[ID = 104] My parent is [101]
```

Note that the output lines may appear in a different order, depending on the order in which processes are scheduled to run (an operating system decision).