CS 365
Challenge #2
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):

2. Explain what happens in the following code snippet.

```c
// 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):

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):

4. Explain what happens in the following code snippet.

```c
// 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):

5. Explain what happens in the following code snippet.

```c
// 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):

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):

7. Explain what happens in the following code snippet.

```c
// 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):

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.

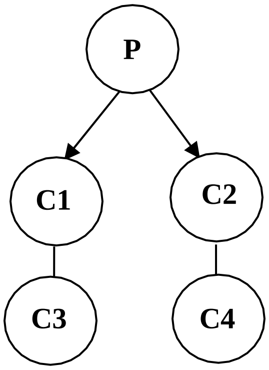
Sample execution (assuming the executable is called xsumfact):

```
$ ./xsumfact    5

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

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):

```
$ ./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).