*Özkan Gezmiş 150200033*

BLG312E Computer Operating Systems

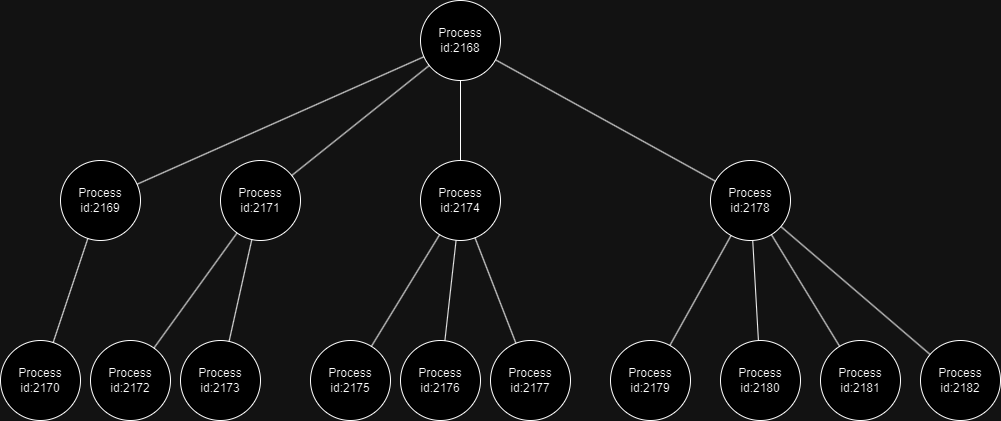
ASSIGNMENT – 1

Part 1

1. **The number of times the fork() function called is 14.**
2. In each iteration of first loop, the main process creates a new child. After child creation newly created child creates i+1 times children.

In the first run newly created child creates 1 child, in the second run 2 new children created etc.

In total, the main process creates 4 children, and each children creates 1,2,3,4 children respectively. **That makes totally 15 processes.**

****

1. When the fork() function called it creates new process and it behave exactly same as the parent process. In the given code actual parent process creates new 4 process and they also create their children. If we didn’t use exit() system call these created processes wouldn’t terminate and they would behave like our actual process and at each iteration they would create a similar process tree like I draw above.

If we want to create identical process and make multiple operation then we may use the children processes longer and don’t use exit() call.

Part 2

In the homework it is asked to write a C code that find the maximum element of an array with multiple threads. Since there is no given array, I made the array size 108 to have enough operations. And I filled array with the random numbers.

Here is the explanation of my code:

The NUM\_OF\_ELEMENTS macro is defined as 100000000, which is the number of elements in the array.

* *max\_num\_array* is a global integer pointer, which will be used to store the maximum number found by each thread.
* *array\_size* is a global integer, which will be used to store the size of the array that each thread will work on.
* A struct named *parameters* is defined, which contains an integer pointer array and an integer thread\_id. This struct will be used to pass parameters to the threads.
* The find\_max function is a thread function. It takes a void pointer as a parameter, which is cast to a *parameters* pointer. This function finds the maximum number in the array and stores it in max\_num\_array at the index corresponding to the thread's ID. It also prints the thread ID and the maximum number found by the thread.
* The find\_max function uses a for loop to iterate over the array. If it finds a number greater than the current maximum, it updates the maximum.
* After finding the maximum number, the function prints the thread ID and the maximum number, stores the maximum number in max\_num\_array at the index corresponding to the thread ID, and then exits the thread.
* The main function is where the execution of the program begins. It first initializes the max\_num\_array and array\_size variables. The array\_size is calculated by dividing NUM\_OF\_ELEMENTS by the number of threads. The max\_num\_array is allocated memory to store the maximum number found by each thread.
* An array of pthread\_t (which represents a thread) is created to hold the threads. A for loop is used to create the threads. For each thread, a parameters struct is created and initialized with the array and the thread ID. The pthread\_create function is then called to create the thread, passing the find\_max function and the parameters struct as arguments.
* After all the threads have been created, another for loop is used to join the threads. The pthread\_join function is called for each thread, which causes the main thread to wait for the thread to finish execution.
* After all the threads have finished execution, the maximum number found by all the threads is calculated by iterating over the max\_num\_array and finding the maximum number.
* Finally, the maximum number is printed and all the allocated memory spaces are freed.

Performance of the Code:

To be honest, I don't know why, but I can't see the performance increase that I want, when I increase the number of threads. The time is takes to run the code is almost same for when thread number is between 1 and 10. When I run the code with *time* command, they give almost same real time.

However, the performance drop of the code when the number of threads is considerably large can be seen.

When the thread number is 10 real time is 1.624 seconds and system time is 0.349 seconds.

If the number of threads is 100 even though user time decrease a little bit the system time really starts to increase. When it is 100 the system time is 0.532 seconds

If I continue to increase the thread number the system time continues to increase. When it is 1000 the system time is 0.949 seconds.

The system time is 5.074 seconds when the thread number is 10k and the user time is 1.268 seconds.

If the thread number is 100k then users time still approximately same, 1.527 seconds, but the system time is 16.646 seconds.

I don’t know the reason but if I make the thread number 200k, I get an error which is “pthread\_create failure”