DV1629 Lab 1

Emil Karlström and Samuel Jonsson

November 16, 2021

1 Task 1

Which process is the parent and which is the child process?
The one printing "B" is the parent.

The variable `i` is used in both processes. Is the value of `i` in one process affected when the other process increments it? Why/Why not?
No, because when they fork, each process gets "its own `i`", which is incremented individually.

Which are the process identities (pid) of the child processes?
`parent_id` + 1, resp. + 2.

2 Task 3

What are the initial values of the semaphores `sem_id1` and `sem_id2`, respectively?
`my_sema1` and `my_sema2`.

3 Task 5

Why do you need to start `msgqsend` first?
Because the file `"touch msgq.txt"` is created in `msgqsend.c`.

4 Task 7

What does the program print when you execute it?
"This is the parent (main) thread" and "This is the child thread".

5 Task 8

Why do we need to create a new `struct threadArgs` for each thread we create?
Because all threads share memory.

6 Task 10

Does the program execute correctly? Why/why not?
No, because two threads can read the data at the same time and therefore they don't take each other's computation in to account.

7 Task 12

An implementation done in line with the description above is not deadlock-free, i.e., it may result in a deadlock. Explain why, i.e., which conditions lead to the deadlock?
Circular waiting - There is a risk that all professors pick up their left chopstick at the same time, leading to no professor having access to their right chopstick.

8 Task 13

Which are the four conditions for deadlock to occur?

1. Mutual exclusion, when a process is using a resource, keeping another process from using it.
2. Hold and wait, when a process is requesting a resource held by another process, but that process requires another resource to release the first one.
3. No preemption, a thread can hoard resources, not allowing others to access them.
4. Circular waiting, when all processes are waiting for a resource kept by other processes.

Which of the four conditions did you remove in order to get a deadlock-free program, and how did you do that in your program?
We removed the No Preemption condition by returning the left chopstick if the right is not available.

9 Task 14

How long time did it take to execute the program?
6.2 seconds.

10 Task 15

How long was the execution time of your parallel program?
1.9 seconds.

Which speedup did you get (as compared to the execution time of the sequential version, where $Speedup = T_{sequential}/T_{parallel}$)?
$\frac{6.2}{1.9} = 3.26$.

11 Task 16

How fast did the program execute now?
2.0 seconds.

Did the program run faster or slower now, and why?
It ran slightly slower because running the initialization with threads creates more resources for the compiler to create, which slows down the compilation.

12 Task 17

Which is the execution time and speedup for the application now?
Execution time (4 threads): 1.7 seconds
Speedup: $\frac{6.2}{1.7} = 3.65$
Because we have less scheduling overhead.

Do these execution times differ from those in Task 15? If so, why? If not, why?
A little bit, due to less overhead and that constantly creating threads might make more work for the processor, making the it slower.

Does it make any difference now if `init_matrix` is parallelized or not?
No, because of the overhead.