Ahmad Tarshehani
Samer Albardan
DVAMI 19

Task 1.

Which process is the parent, and which is the child process?
A=Child because when process id equal to zero, B = Parent process

The variable I is used in both processes: Is the value of I in one process affected when the other process increments? Why/Why not?
No, because the child process has its unique id not the same as the parent process and the I in both processes are unique variables with unique pointer addresses.

Which are the process identities (pid) of the child processes?
child one id is: 129451
child two id is: 129452
Perant ID is: 129450

Home Assignment 4

The program connects the parent process and child process by shared memory. First we shared memory struct with the data(buffer) we want the two processes to have access to and empty to see that can other process can enter and get the data. So when the parent as producer sends the number it stores it in the memory and the child as the consumer receives the number from memory and prints it out(fflush).

Home Assignment 6

The program *semaphore.c* make sure that processes that run the code run it in order by waiting for the other process to finish his work and then the other process can print his data and it is happen by using sem_wait() and sem_post(). The program print out A and B from the parent and child process. And then close the order by sem_close().

Task 3. Semaphore example

What are the initial values of the semaphores sem_id1and sem_id2, respectively?
sem_id1= sem_open(semName1, O_CREAT, O_RDWR, 1); initial value 1
sem_id2 = sem_open(semName2, O_CREAT, O_RDWR, 0); initial value 0

Home Assignment 8

The program *msgqsend.c* create a message queue with a unique process id by using ftok for inter processing communication and create file msgq.txt with this unique id.The program first reads the input from the user,if there are any new line at the end of the string it will remove it. Then the program sends the input into the message queue and the program repeats that until it gets (CTRL D) as an input. The program will end once the end of the input and the program will send the end as a message and that will also be the end of the process. The *msgqrecv.c* program will first read the message from the queue and then display the message. Lastly, the program will end the process when the received message has ended.

Task 5. Message queue example

Why do you need to start msgqsend first?
Because you have to create the message queue and the file unique process id for communication so that *msgqrecv.c* find the id to connect to and resv the message.

Task 7.pthread_create()example

What does the program print when you execute it?
The main thread prints “This is the parent (main) thread.”
The child thread prints “This is the child thread.”
When the thread creates and thread run the child function the main thread prints the parent main thread and when the thread joined so the child thread prints the child thread.

Task 8.pthread_create()example 2

Why do we need to create a new struct threadArgs for each thread we create?
You may get Violation segment (`core 'generated) error if we don't have new struct threadArgs for each thread we create and we new struct threadArgs because every thread can get their own value so we need a new struct so we don’t change the value of another thread. When we want to return a value from a child thread, the function must have return type void. This means that the return statement must pass the arguments in the struct passed to the child thread.

Task 10. Bank account

Does the program execute correctly? Why/why not?
The program did not execute correctly because two (or more) threads try to read and update the same variable at the same time, and their accesses to the shared variable are interleaved. When updating a variable, a thread reads the value of the variable from the memory, updates the value (a local operation in a processor register), and finally writes the new value back to memory. When two threads do this concurrently, the execution may have a race condition.

Task 12. Dining Professors, implementation

Explain why, i.e., which conditions lead to the deadlock?
The deadlock will happen when all professors take the left fork at the same time or two professors sitting beside each other take the available fork that they have at the same time. Whenever the professor tries to take the right fork they will get stuck. And all professors start to eat even if they are not their time to eat.

Task 13. Dining Professors, deadlock-free implementation

Which are the four conditions for deadlock to occur?
1. Mutual exclusion condition.
2. Hold-and-wait condition.
3. No-preemption condition
4. Circular wait condition.

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?
Mutual exclusion condition. We avoided this condition by using a semaphore for each thread that we initialized and with this semaphore solution. We did the algorithm that professors do not eat simultaneously and just two professors enter the room and pick the left and right chopstick and start to

eat while the other professors wait to the eating professors to finish then one of the waiting professors jump in and start eating and so on.
Example professor 0 enters the room, professor 0 thinking, professor 3 enters the room, professor 3 thinking, professor 0 picks left and right chopsticks,professor 3 picks left and right chopsticks, professor 4 and 5 are ready to eat they just wait until 0 or 3 finish eating, when 0 finish eating and drop the chopsticks professor 4 start to eat and so on.

Task 14. Sequential matrix multiplication

How long time did it take to execute the program?
After i run the program 6 times the real time i get between 3.618 s to 3.899 s in average 3,776 s

Task 15. Parallel matrix multiplication

How long was the execution time of your parallel program?
After I run the task 15 program 7 times, I get the time between 1.461 s to 1.542 s and on average 1,498 s which is much faster than the sequential method.

Which speedup did you get (as compared to the execution time of the sequential version, where $Speedup=T_{sequential}/T_{parallel}$)?
Min speedup = 3.618 / 1.461 = 2,476, max speedup = 3.899 / 1.542 = 2.528 and on average speedup is 3,776 / 1,498 = 2,520 the speedup we get when we run parallel matrix multiplication.

Task 16. Parallelization of the initialization

How fast did the program execute now?
After i run the program 7 times the real time i get between 1.413 s to 1.582 s on average 1.495 s so actually not very fast it is the same if you just have parallel matrix multiplication.

Did the program run faster or slower now, and why?
It is not slower but if you compare the average 1.498 s on parallel matrix multiplication and 1498 on parallelization of the initialization 1.495 s so the program run a little bit faster in 0.002 s faster and why because we have only thread that is initializing each row in the matrices A and B.

Task 17. Changing granularity

Which is the execution time and speedup for the application now?
I run the program 7 times and got the time between 1.524 s to 1.691 s on average 1.596 s so the min speedup is 3.618/1.524 is 2.374, max speedup is 3.899/1.691 is 2.375 and on average speedup 3,776/ 1.596 is 2.365.

Do these execution times differ from those in Task 15? If so, why? If not, why?
Yes is differ because why use the threads to run matrix multiplication and initialization simultaneously which make it faster by 2.2 seconds and that what differ.

Does it make any difference now ifinit_matrixis parallelized or not?
No, it does not make any difference if ifinit_matrixis parallelized or not as we can sew in out time checking