Istabul Technical University
Spring 2019
BLG 312E – Computer Operating Systems

Homework 3

Melik Mehmet BIYIK

INTRODUCTION

This is a painting shop simulation project.In this project there are some boxes to paint and there is a paint station which has a capacity of two.There is a parent (master) process and there are some children (one for each box).Synchronization is necessary to painting box and the number of paint changeovers must be minimum.Because of we use child process instead of thread, we must take advantage of shared memory.Because if we don't use shared memory Semaphore values cannot be reachable by all children processes.So there cannot be synchronization.

LOGIC OF THE CODE

In this situation I used a two-capacity semaphore and I used a pointer list which is attached to shared memory.Semaphore is created in shared memory with a key value.Processes reach the semaphore with this key, so all child-processes know there is a critical section and if it is full they have to wait until it is empty.The synchronization I designed is that

-First arrived color has higher priority

-First arrived box among the same colors has higher priority

For example:

R G Y R Y G

In this order boxes have R color has higher priority after R, G and Y accordingly.In the group of R color first arrived R has higher priority.

However, it is possible to paint two boxes in the same time, there is one resource to write and I have to arrange this synchronization like this logic:

Two boxes can enter in the critical section in the same time but if I don't arrange the code the order written in the output file could be random.So I calculate the priority of the color group this means all same colors has same priority.And I calculate a second priority which I called it order in the code, I used this order variable in the same color group and first arrived box has higher priority in its color group.

***priority_flag variable (in shared memory)**

After calculate these values I defined a pointer integer in the shared memory.It is name is priority_flag which is initially zero, I used this variable as priority counter.So it means I increase value of that variable after every color group is painted.Because of this variable starts from zero first higher group is painted firstly.So all boxes (processes) waits in a while loop for priority order.For example in RGPRYG order priority_flag is 0 until all R boxes are painted, and when they are painted priority_flag is 1 and the shop starts painting G boxes and so on.

***(priority_flag+1) variable (in shared memory)**

As I mentioned earlier I need a secondary priority and a variable in the shared memory to keep the value of this priority.Because of I define priority_flag pointer integer, I can use this like a list and I stored the value of secondary priority in near the address of priority_flag.It means in C I stored that value in *(priority_flag+1) variable.And I increase that variable for every pointing process.But I restart this variable from zero when a color group is painted.Therefore every process knows the order in the same color group.

***(priority_flag+2) variable (in shared memory)**

Lastly, I used a variable which I think it is important and I have to mention about is a variable such that counts processes in the critical section.So I can use a boolean

variable to express that if once painting shop is full then no one can enter in the painting shop until it is totally empty.This counter variable is *(priority_flag+2).I increase every critical entrance and decrease when processes leaves the critical section.So processes know that if this variable is 2 then the shop is full and it is 0 then shop is empty.But what if it is 1?If it is 1 there are two situation, as I said I used a boolean variable whose name is intruder and I assign it to true if *(priority_flag+2) is 0 and it remains until *(priority_flag+2) is 2.When *(priority_flag+2) is 2 then I assign this boolean to false, it remains false until *(priority_flag+2) is 0.So once shop is full, no one can enter into critical section until the shop is totally empty.Instead of this variable I could use sem_getvalue( ) function but I did try use this it didn't run because of some warnings.I think there are some missing in my c libraries at the end I decide to use this variable.

Functions I used is following:

write_shm(char *str) : Creates a shared memory and writes the given string into that shared memory.

read_shm( ) : Reads from the created shared memory by the write_shm function and stores that inputs into the read_str char list which is defined statically.

is_there_pair(char ch, int x, int n) : It takes three inputs such that first a char second an integer as index of this char and lastly an integer as length of the read_str array.It searches the given char into the read_str array from the given index +1 to the length of the read_str array.And it returns the number of the same char in that scope.

first_write_to_output(char output_file[ ],int index) :This function takes output file name and open a file with write operation this means it deletes everything in output file and then write the given number which is changeover number.

write_to_output(char output_file[],int index, char color) :This function takes output file name and open a file with append operation and writes the given index and char at the end of the output file.

I used these functions because every time I need to write the file I have to open the file and close the file for preventing any confusion and provide the synchronization.Thats why it is important to use these functions.Open and especially close file operations is the key for the providing the synchronization.

WHAT WAS IT HARD FOR ME ?

It was easy to define a semaphore in the shared memory for all child processes, but I think it was hard to set a priority and decide which process will enter in the critical section.Most of my time is taken by this issue.At the end, I overcamed by using a while loop encapsulating critical section.

CONCLUSION

This project was really nice.I learned lots of things.I have got synchronization experience and I realize the difference between threads and processes, threads shared same data section so there is no need of shared memory.But processes have different data section and communication is done by shared memory and massage passing.I learned difference between mutex and semaphores and their difference.I think the most valuable thing I learned from this project is set a priority and use of the critical section.Thanks a lot.