Report Checkpoint 3

Jonathan Sutedjo 鄭安良 111006207

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

In the preemptive.h file, I make changes to it for the semaphore by adding the SemaphoreWait which is to call the SemaphoreWaitBody and pass the parameter with the unique integer \_\_COUNTER\_\_. In the SemaphoreWaitBody, it first move the value of S, then check if the ACC is equal to zero or not, the PC will jump back to MOV ACC, CNAME(S) if it is equal to zero. This will make a loop. In the condition that it is not equal then it will check the 7th bit of ACC if it is 1 or not, where this 7th bis it to know if it is a negative number or not. If it is 1 then it will jump back to the top to be a loop, but if it is not then the PC will continue and the S will be decrease by 1. Here there is also the SemaphoreSignal which is to increase the S by 1.

A screenshot of a computer code

Description automatically generated

A purple screen with text

Description automatically generated

In the testpreempt.c file, I add mutex, full, empty, tail, and head, where the tail and head is for the shared buffer. Then The var to keep the A-Z is the “TheChar”.

The SemaphoreCreate is to put a value in char to the semaphore. Here I use a pointer so that the one that I update is the value from the source address of the semaphore.

A screenshot of a computer program

Description automatically generated

For the producer part, now I use the bounded buffer such that I need to wait for the semaphore to be ready before doing anything to the shared buffer. The producer will need to wait for the “empty” to be not 0 and the “mutex” to be 1 before it can produce more character. After the character is produced of course it is assigned to the SharedBuffer[tail]. After it is done, it will return the value of mutex and signal full. The full signal is so that the consumer know there is new value inside the shared buffer. The mutex is so that other thread can use it.

A screenshot of a computer program

Description automatically generated

In the consumer part, I use the bounded buffer such that the consumer will have to wait for the producer to signal full to get the new value in the shared buffer. After the signal is received, it will wait for the mutex. Consumer will signal mutex to let other thread run after it is done. It will also signal empty to let the producer know the sharedbuffer have spot available for new characters now.

A screenshot of a computer program

Description automatically generated

Here, I need to first initialized the semaphores by using the SemaphoreCreate for full, empty, and mutex. The value of full will be 0 because it means there are no new characters there. The value for mutex is so that only 1 thread run at a time. The empty will be 3 because there are 3 slots available for new characters and that is what we want (3-deep).

A screenshot of a computer

Description automatically generated

This is some of the map file content.

Image Context 1

A screenshot of a computer program

Description automatically generated

Image Context 2

A screenshot of a computer

Description automatically generated

Image Context 3

A screenshot of a computer

Description automatically generated

Image Context 4

A screenshot of a computer

Description automatically generated

Image Context 5

A screenshot of a computer

Description automatically generated

For context, the mutex is 0x25, full us 0x26, and empty is 0x27.

From Image context 1-5, we can see that the value of full will be increase by 1 every time a new character is produced by the producer. While the full is increased by 1, the empty will decrease by 1.

Image Context 6

A screenshot of a computer

Description automatically generated

Image context 7

A screenshot of a computer program

Description automatically generated

From image context 5-7 we can see that the full will decrease by 1 every time when it print out 1 character. The empty will increase by 1 also when the full is decreasing.