Daniel Cender

Jan 19, 2020

CST-221 O500

Wk 2: Semaphores and Monitors

Semaphores

**Summary**

Using the *pthreads* and *semaphores* libraries in C , I have implemented a basic example of a solution to this assignment. A single processing thread function will attempt to synchronize and print out the entire alphabet, char by char.

Two threads are spun up using the function *printAlphabet*, which will then place a lock on the *sem\_t* variable while it runs.

**Result**

The output from this program is a list of *printf* calls made from *printAlphabet* which show when the thread is entering the sensitive zone, exiting it, and printing it’s relevant alphabet character. The screenshots below are portions of the final output from this program.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

There should be no cases where the printAlphabet function either enters or exits the printing block twice in a row. As seen in the output, both threads take equal turns, one after the other, to print out the entire alphabet. My use of semaphores was influenced heavily by the simple example given by Kapse from Tutorialspoint (2019).

Used this: <https://www.softprayog.in/programming/posix-semaphores>

And this answer: <https://stackoverflow.com/questions/1413785/sem-init-on-os-x>

**Code**

Even though the entire code solution is below, it may also be found uploaded to my GitHub repository for this course, here: <https://github.com/DanielCender/CST-221/tree/master/Wk2/MonitorsAndSemaphores>

The below code was compiled from the command line and run with the following commands:

* ‘gcc -o Semaphores.a Semaphores.c’, then
* ‘./Semaphores.a’

/\*

*\* Author: Daniel cender*

*\* Date: 01/19/2020*

*\* Basic program that utilizes pthreads and semaphores to*

*\* produce a program that prints out the entire alphabet using multiple threads in sync.*

*\**

*\**

\*/

#*include* <*stdio.h*>

#*include* <*pthread.h*>

#*include* <*semaphore.h*>

#*include* <*unistd.h*>

sem\_t mutex;

*char* alphabet*[]* = {'*a*','*b*','*c*','*d*','*e*','*f*','*g*','*h*','*i*','*j*','*k*','*l*','*m*','*n*','*o*','*p*','*q*','*r*','*s*','*t*','*u*','*v*','*w*','*x*','*y*','*z*'};

*int* idx;

/\*\*

*\* Func that prints out 2 chars of the alphabet array*

\*/

*void\** printAlphabet(*void\** *arg*) { //*function which act like thread*

*while*(idx <= 25) {

sem\_wait(*&*mutex); //*wait state on semaphore*

*if*(idx > 25) *break*;

printf("*\nEntered..\n*");

printf("*%c\n*", alphabet[idx]);

printf("*Index at: %i*", idx);

++idx; // *increment index*

printf("*\nJust Exiting...\n*");

sem\_post(*&*mutex); //*send message to free up mutex*

}

pthread\_exit(0);

}

main() {

idx = 0; // *max of 25, total length of alphabet*

// *init semaphore to 0, for only this process*

sem\_init(*&*mutex, 0, 1);

*pthread\_t* th1,th2;

//*Create threards*

pthread\_create(*&*th1,*NULL*,printAlphabet,*NULL*);

pthread\_create(*&*th2,*NULL*,printAlphabet,*NULL*);

//*Join threads with the main thread*

pthread\_join(th1,*NULL*);

pthread\_join(th2,*NULL*);

sem\_destroy(*&*mutex);

}

Monitors

**Summary**

Based on my limited understanding of monitors, they exist as more of an abstract language construct, which give a different experience to the developer than a language that uses traditional thread-based synchronization. Monitors manage blocks of code logic so that they never execute simultaneously using the same critical resources.

From this, I tried to re-format my code from the previous program to simulate how a monitor would appear to a developer, although it is “hacky” to implement or experiment with quickly in C.

**Results**

No actionable results worth mentioning.

**Code**

No code was gotten to the point of running successful. I would have needed more time to complete a workable monitor structure using *pthreads*.

/\*

*\* Author: Daniel cender*

*\* Date: 01/19/2020*

*\* Basic program that utilizes pthreads and semaphores to*

*\* produce a program that prints out the entire alphabet using multiple threads in sync.*

*\**

*\**

\*/

#*include* <*stdio.h*>

#*include* <*pthread.h*>

#*include* <*semaphore.h*>

#*include* <*unistd.h*>

// *sem\_t mutex;*

*char* alphabet*[]* = {'*a*','*b*','*c*','*d*','*e*','*f*','*g*','*h*','*i*','*j*','*k*','*l*','*m*','*n*','*o*','*p*','*q*','*r*','*s*','*t*','*u*','*v*','*w*','*x*','*y*','*z*'};

*int* idx;

*pthread\_t* monitor\_ts[2]; // *Maybe hold couple of pthreads to handle/wait based on sync blocks?*

*struct* MONITOR {

sem\_t startSync;

sem\_t endSync;

*int* firstAction;

*int* secondAction;

}

*struct* MONITOR monitor;

*int* monitor\_init() {

// *Initialize the structure*

monitor.firstAction = 0;

monitor.secondAction = 0;

// *initialize the semaphores*

*if*(sem\_init(*&*(monitor\_data.startSync), 0, 1) == 0 &&

sem\_init(*&*(monitor\_data.endSync), 0, 1) == 0){

// *Init and join threads here maybe??*

*return* 0;

} *else* {

printf("*Unable to initialize semaphores\n*");

}

*return* 1;

}

*void* monitor\_destroy() {

sem\_destroy(*&*(monitor.startSync));

sem\_destroy(*&*(monitor.endSync));

}

/\*\*

*\* Func that prints out 2 chars of the alphabet array, shouldn't need*

*\* to wait or post to semaphore if working in monitor structure*

\*/

*void\** printAlphabet(*void\** *arg*) { //*function which act like thread*

*while*(idx <= 25) {

*if*(idx > 25) *break*;

printf("*\nEntered..\n*");

printf("*%c\n*", alphabet[idx]);

printf("*Index at: %i*", idx);

++idx; // *increment index*

printf("*\nJust Exiting...\n*");

}

pthread\_exit(0);

}

main() {

idx = 0; // *max of 25, total length of alphabet*

*if*(monitor\_init() == 0) {

// *Ready to define synchronized program logic*

monitor\_startSync(); // *TODO*

// *Process 1*

wait(3); // *Would ordinarily cause second thread to run first*

print("*Would print out some stuff here, to print first....*");

monitor\_endSync(); // *TODO*

/\* *\*\*\*\** \*/

monitor\_startSync(); // *TODO*

// *Process 2*

print("*Would print out some stuff here, to print second....*");

monitor\_endSync(); // *TODO*

};

monitor\_destroy();

}

Resources

Kapse, S. (2019, May 6). How to use POSIX semaphores in C language. Retrieved January 20, 2020, from https://www.tutorialspoint.com/how-to-use-posix-semaphores-in-c-language

Tanenbaum, A. S., & Bos, H. (2017). *Modern operating systems*. Vancouver, B.C.: Langara College.