Client Server Message Passing

Balaji Chandrasekaran
School of Computing, Informatics and Decision Systems
Arizona State University
Tempe, Arizona
bchandr6@asu.edu

***Abstract*—The project is made up of four phases. Each phase is interconnected and current phase depends on previous phases. In every phase, we build a feature which will all be ultimately used in the final phase to build our message passing program. The project is developed to account for the parallel computational environment. The three main concepts that it deals with is Multi – Threading, Synchronization and message passing. For achieving all these three concepts we develop a Scheduler, Semaphore and message passing interface respectively which comprises of the first three phases of project and the fourth phase deals with development of a client server system which comprises all the features those developed previously.**

Keywords—Message Passing; Scheduler; ucontext; Thread Synchronization; Semaphore; Multi-Threading; Mutex;

I. Introduction

The main objective of the project is to learn about three main concepts which are threading, synchronization and message passing Interface which is used for parallel applications. In order to learn the above concepts, this project was done in total of four phases in which we did a thorough study of each of the concepts. We used C language and developed this project. In the first phase, we learnt about multi-threading for which we developed a library for generating user level multithreads using the ucontext library that is available. The second phase consisted of developing a mechanism that could handle synchronization of the threads for which we used semaphores. We then used the semaphores that is synchronization and solved the reader's – writer's problem in this phase. The third phase consisted of creating a message passing system which used the result of the previous phases. The fourth and final phase consisted of creating a client server machine which was built over the message passing system. The project gave us in-depth knowledge and understanding of threads working mechanism. It also thought us the importance of synchronization and how message passing takes place in a parallel architecture without affecting its purpose.

II. Solution

A. *Phase I : Scheduler for User level Multi Threads.*

The first phase deals with the process of implementing a user level multi-threading. For the same purpose, we used ucontext library which is available. Uncontext stands for user context which consists of four important methods getcontext, setcontext, makecontext and swapcontext. We used three of these functions which are swapcontext, getcontext and makecontext. Getcontext function takes in a single parameter regarding the current context which is of struct type of ucontext_t which consists of four members which are "pointer to context that will be resumed – ucontext-t", "stack used by the context – stack-t"," blocked signals set - sigset-t", "machine specific representation of context – mcontext-t". The getcontext function sets its ucontext_t parameter with the current user context of the function/thread that calls the getcontext. Makecontext takes in a context, a function and several other arguments, it is used for modifying the context of the context present as a parameter of the function. When there is a call to the context again in the future the call is passed to the function that is present in as a parameter in the makecontext call. The swapcontext function takes in two parameters which are both context type and then swaps both of their context so that the running context is changed. In this program of phase I, we had three libraries that we had implemented and used for creating multiple threads. The first library was the "tcb" library which had the implementation of thread control block. Thread control block is nothing but a data structure which contains information regarding a thread to manage or control it properly. Some of the main information that are required to manage a thread would be its identifier, PC (program counter), Pointer to thread stack, thread state, pointer to PCB of its corresponding process. For obtaining and storing this information we used the ucontext library which consists of several functions as we have seen already for implementing TCB easily. The library also consists of a function init_TCB which is used for making context of threads and assigning the same to TCB type objects. Then we had the "q" library which had the implementation of creating and managing a circular doubly linked list that is a Queue. This Queue acted as a scheduler buffer which consisted of current and other threads that are to be executed in the order of execution. The library had InitQ, RotateQ, AddQ and DelQ function for initializing the queue, rotating the queue head, adding into the queue and deleting a queue node respectively. "threads" library is responsible for thread functionality like start_thread, run and yield function. Start thread makes call to init_TCB in the "tcb" library for initializing the context and adds the TCB to the queue. Run function gets the context and swaps it with the next context in the queue. Yield function makes a call to RotateQ for rotating the queue head and swaps the context to the next context in the queue. Then there is the testing program which has the main in it and consists of two functions namely add and mul. The purpose of the function is to do some mathematical operation and these function act as the context to the two threads that we are going to create and make them run.

B. Phase II: Semaphores and Reader's – Writer's Problem.

The second phase consisted of the process of implementing synchronization into the multi threads paradigm. We then used the implementation of this synchronization and solved the Reader's – Writer's problem. Synchronization is something that is very important when it comes to multithreaded paradigm and if not handled properly it could lead to undesirable consequences. The main objective of synchronization is to solve the problem of critical section. The critical section problem arises due to the multi-threaded environment. When say there are two processes which are trying to access the same memory and these processes are run in parallel then there could be situation when both the process could access the memory at the same time and destroy the meaningfulness of the data. This kind of situation is known as critical section problem and the shared memory related code is known as the critical section of the code. We need some kind of synchronization that is understanding between the processes accessing the critical section that only one can access such section of code at a time and the other process as to wait for the current process to complete execution. For solving this problem there was a protocol developed which had three basic steps to solve the problem:

- Entry: A process entering critical section must make sure no other process is accessing the same critical section.
- Critical Section: After gaining access and entering the critical section it can execute the section and other process requesting the section must wait for it to complete.
- Exit: The process leaving the critical section specifies the same and clears its access so that another process can access the section.

The problem implementing such a protocol in an effective way is a challenging task. For this task, we implemented semaphores. Semaphore are nothing but variables which will be used for controlling the access of the critical section in parallel processing multi-threaded environment. In this phase, we used all the libraries from the previous phase so that we can use the multi-threaded scheduler in the current phase as well. We then created a library called as "sem" which contained the implementation of the semaphore, it had three functions which are "CreateSem – which was used for creating the semaphore, P – which is used for wait functionality of a semaphore", "V – this function is used for signal functionality of a semaphore". For achieving synchronization, we use the signal and wait function by checking the semaphore variable for granting permission that is signaling to use a critical section or making the thread to wait in the queue. Using this library, we solve the reader's – writer's problem which uses three semaphore variables namely reader semaphore, write semaphore and mutually exclusive semaphore. Mutually exclusive semaphore ensures the calculation for wait and signal of various threads gets executed without the problem of critical section that is there can be only one thread in entry or exit section at a given time. The reader semaphore corresponds to the reader functions which are trying to read a critical section and the writer semaphore variable account for all the writer functions which is trying to write in a critical section of memory. The main issue with a reader writer problem is that when there is a share of critical section given to a reader there should be no other reader that is coming in for critical section should be kept waiting. Using the above mentioned three semaphore variables and the function in the "sem" library we resolve the reader writer problem.

C. Phase III: Message Passing Interface.

The objective of this phase was to develop a message passing system which was multi-threaded and synchronized. For this phase, we utilized the result of the previous two phases. The objective of a message passing system is when we run a process on a parallel system we should have definitive way for communication between the process that we execute in that architecture and they should be able to communicate with one another by sharing information of each other. In a message passing system there is no way two process can share same memory, they look to do so but they don't actually do it but rather have their own copy of variables to work with. We developed a library called as msg using the previous libraries. The objective of the library is to implement message passing system. We defined the port structure which contains a message which is a two-dimensional square array of size 10, each port contains three of the semaphores namely sem_send, sem_recv and mutex like the reader's – writer's problem we have seen in the previous phase. The purpose of sem_send and sem_recv is to take care of the synchronization at the port like the writer semaphore and reader semaphore respectively and muter semaphore for controlling the entry and exit sections of synchronization mechanism that is the P and V function. We declared array of ports of length 100 and used the same for message passing. We implemented three functions which constituted the two main functions of message passing which is send and receive, the third function was init function which was used for initializing the ports. In the send function, there would be the wait function (P) from the sem library which makes the process to wait until there is a signal of critical section being free. The process of changing the semaphore variables that is the sem_recv and sem_send in the send function acts as a critical section which is being taken care of by the mutex semaphore. Similarly, there is the receive function which handles receiving of the message sent by the port using the semaphore variables. Later these ports are used in the next phase to develop a client server system for communicating using message passing Interface that we built.

D. Phase IV: Client – Server Architecture.

We used the previous phase output which is the message passing interface and developed a client server architecture like system which would be able to communicate with each other using the message passing system. This phase we used the msg library and created a program which consisted of two function which are client and server. The client sent messages to the server, the messages sent was in a random fashion using a variable with modulo of three which was chosen randomly, three being the number of cases present in the client. The server receives the messages and stores them and sends back to the respective client from it. This way clients communicate with each other or with the server in this architecture. The client server functions act as the multi-threaded environment in this program and there must be certain synchronization mechanism which is being taken care of by the msg passing interface that we developed in the previous phase using the synchronization mechanism. These multi-level threads communicate with each other by passing messages. In this program, we created three client threads and one server thread which acted as the middle man for communication. It was very interesting to see how a message passing interface works at a very low level in the Operating System.

III. Results

A. *Phase I : Scheduler for User level Multi Threads.*

The result of the first phase was a library that would help us in implementing multiple threads for which we had to also implement a scheduler for handling the multiple threads. The scheduler was nothing but a queue. The result of this phase consisted of three libraries namely the "tcb", "q" and "threads". The tcb was for implementing the thread control block, q for the scheduler queue and threads for implementing the threads.

B. *Phase II: Semaphores and Reader's – Writer's Problem.*

The phase II result had an extra library to the already created libraries which provided the synchronization mechanism that is they provided an implementation of semaphores and their synchronization functions which are wait and signal function. The name of the library was sem. Using this library, we then created a program for resolving the reader's and writer's problem using the libraries.

C. *Phase III: Message Passing Interface.*

Result consisted of another library added to the previously created libraries which had the feature of message passing system. This library known as the msg library had the implementation of a message passing interface using the synchronization and scheduler that we already created. It had the two-main routine of send and receive for message passing between the processes or threads.

D. *Phase IV: Client – Server Architecture.*

The output of this phase had a program which consisted of the client server architecture using the message passing interface that we created in the previous phase. The program invoked three client threads and one of server thread. This paved way for the communication of the client and server once the threads where started. The result of this phase is that there is no guarantee which client's thread message gets delivered first. But when it comes to multi-threaded parallel computing paradigm the problem of resolving the critical section is solved by this system.

IV. Knowledge Acquired

The amount of knowledge that this project has thought us is enormous. It gave us profound knowledge on several aspects of multi-threaded environment and parallel computing paradigm.

A. *Phase I : Scheduler for User level Multi Threads.*

1) Multi Threads: This phase thought us on the understanding of multi threads. It gave us knowledge about the drawbacks or consequences of using multi threads.

2) Scheduler: This phase provided us with great insight on scheduler and how they work and how they actually schedule various jobs and maintain the queue. By implementing a scheduler on own this phase also provided us with practical experience of developing our own scheduler.

B. *Phase II: Semaphores and Reader's – Writer's Problem.*

1) Semaphores: This phase thought on resolving the drawback of using multi-threaded environment that we saw in previous phasse by usisng semaphores. It helped us to learn on how a semaphore can erradicate the problem of critical section and how to resolve any futher problem of implementing semaphore properly.

2) Reader's - Writer's : The practical example of solving a reader's – writer's problem helped us to a great extent on learning the implications of semaphores and how to properly implement one without the critical section problem.

3) Synchronization : This phase also thought us about the importance of synchronization and how it is achieved with the help of semaphores.

C. *Phase III: Message Passing Interface.*

1) Message Passing : In this phase we learnt about message passing and its importance in a parallel computing paradigm.

2) We also implemented a message passing system on our own that gave us practical knowledge on how to build a message passing system.

D. *Phase IV: Client – Server Architecture.*

1) This phase we built a client server architecture which gave us practical application of a message passsing system and how it works.

2) We saw how synchronization is used in realtime scenario.

The project also gave us lots of knowledge on implementing our own mechanisms for synchronization, multi-threading and message passing. It thought us about ucontext library in C and how it works. It provided us with in-depth understanding of C on pointer handling. It also gave us insight on how C is the perfect language when it comes to operating system related concept.

V. TEAM

Kandhan Sekar, Vimal Khanna Vadivelu