Deverick Simpson & Yuekun Yang
CSE 351-OS Kernels

CSE 351- OS Kernels
Project Part 2
Thread Management of Prototype OS

Introduction

To continue our understanding of the OS kernel, the objective of the assignment is to design and implement a thread scheduling subsystem to allow our prototype OS to create, schedule, maintain, and terminate multiple threads at runtime. The thread scheduler will mimic some of the main features used within the pThread library of Linux. Groups are provided a DE2 NIOS board to implement the design. The DE2 board provides an alt_alarm() function that will be manipulated within the project to enable a scheduling system for the purpose of concurrent threading. To avoid compatibility issues between machines, the programming environment will primarily be within the CSE server on campus.

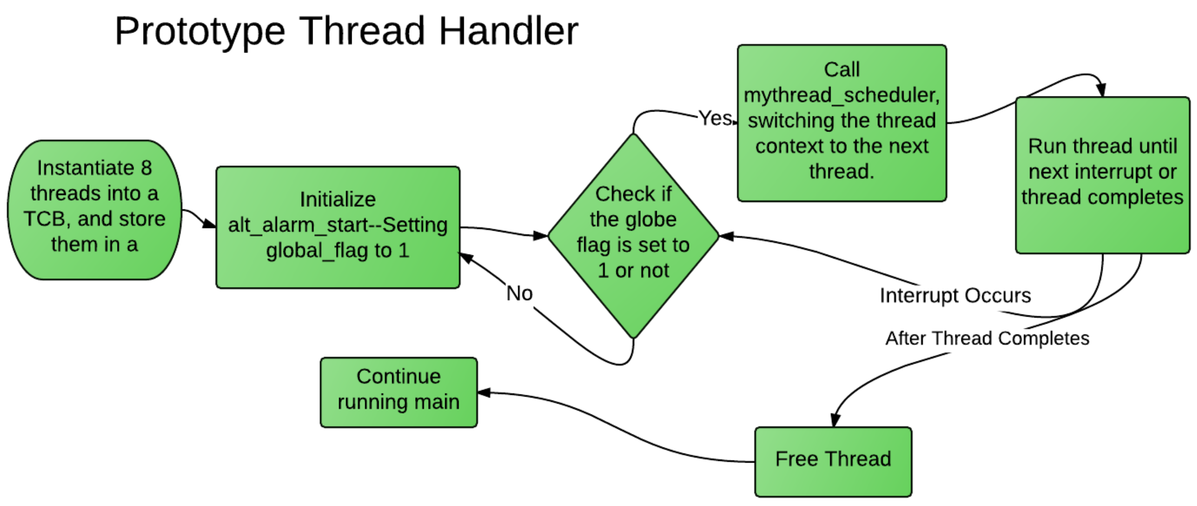
Project Management

To begin the project, each partner researched materials related to threads and set up environments on local machines. References to the lecture notes and videos were made as well as to forums and other pseudo code related to the material online. Lastly, classmates provided useful high-level discussions concerning stack structure and debugging strategies.

An attempt at pair programming was made however scheduling conflicts required partners to divide the work to become more efficient. Towards the middle of implementation, it became clear Deverick would be completing the implementation and Yuektan would complete the paper. Great risk was created with this decision as the implementation was designed from one individual's interpretation while the partner is left to fend for himself. To defend this, it almost seemed necessary to change our developing strategy to divide the work as it constantly seemed one partner would complete the work while the other would observe. Another obstacle to overcome was the language barrier between my Yuektan and I. It has been quite a struggle for Yuektan to follow the material through out the course, requiring him to put more time into understanding what is being said before being able to understand the material.

Strategy for Solution

Developing a complete solution for the implementation of the thread handler required considering a few topics that hadn't been a main focus of many other courses. These topics include theories such as the stack, the context of a thread, and manipulating the alt_alarm_start() function within the Altera library. More specifically, we look to utilize the alt_alarm_start to initiate the saving and restoring of the context of running threads within the program. Each thread will be simulated by a provided mythread() function which produces a print to the console log of the current running thread. The project will look to initialize a queue struct, TCBQueue, to contain the running threads. The queue will follow a round robin scheduling principle to allow for sequential thread processing. Please refer to the diagram below for a workflow of the program.


Prototype Thread Handler
Instantiate 8 threads into a TCB, and store them in a
Initialize alt_alarm_start--Setting global_flag to 1
Check if the globe flag is set to 1 or not
Yes
No
Call mythread_scheduler, switching the thread context to the next thread.
Run thread until next interrupt or thread completes
Interrupt Occurs
After Thread Completes
Free Thread
Continue running main

Deverick Simpson & Yuekun Yang
CSE 351-OS Kernels

Summary

The project introduces important concepts within the Operating System essential to its functionality. Designing and implementing a thread handler library emphasized the importance of its ability to sucessfully manage threads sharing a similar stack space. Such capabilities within the machine are important as it permits efficient data managing in memory. Without such an implementation, processes would have to complete before the next thread would be allowed to run. This can prove unsuccessful, if for instance, the first running thread requires a large length of time to complete. Processes further towards the end of the queue will wait until all threads ahead have completed. From this, It can be understood that a successful thread handler will be able to share a resource location among multiple threads, allowing each to run for some intermediate length of time before switching context after an interrupt occurs.

The documentation outlines 5 steps to follow while developing the prototype thread handler. As summarized within the workflow, the program creates eight threads during run time, actives the alarm for interrupt handler, schedules them and switch their context properly, and lastly returning to main after threads have completed. Overall, the project proved difficult, however, during the struggling time, we understood further what is discussed within the course, i.e., the underlying processes of an operating system.

The ideas used to create the thread handler are fairly simple in concept, with a few structs to contain data and manipulating alt_alarm_start to store and restore the context of a simulated thread. The implementation of the program proved a bit challenging for a few different reasons. Prior to the coursework, there has been little involvement with C programming and Altera. Also, there was a bit of difficulty for the functionality within the proposed TCBQueue. More specifically, the enqueue and dequeue of our TCBQueue had a few bugs within it.

A downside to the project is that the organization of the program is not well maintained. Development was pushed frequently to github to manage the versioning of the program, however all development stayed within two files, an assembly file and a c file. It could be more manageable if header files were created for each c file. Also, it would be ideal to separate the alarm functions and the queue struct from the main functionality of the program. In the next iteration of the project, it will be proposed that such organization is structured within the code.

Overall, the project is fairly clear and concise. The documentation provides most materials useful for the project. Outside of the given documentation and Alteras resources, little research was required to complete the project. For future thread handler projects, a similar documentation could be provided with little hesitation.

The project provides great emphasis on the principles studied within the course. The development of the thread scheduling system enables or prototype OS to create, schedule, maintain, and terminate multiple threads at runtime. Although it is defined as the most challenging project among the three, it seems to be one of the more important principles discussed.

Deverick Simpson & Yuekun Yang
CSE 351-OS Kernels