11/15/2015

Designing a Thread Scheduler for a Prototype OS

CSCE351 Project 2

Endi Xu
NUID: 53886514

Contents

Introduction 2

Project Goal 2

Problem Description 2

Project Management 2

Details of the Thread Scheduler 2

Overview 2

Body Part (In C Language) 2

Interrupt (In Assemble Language) 3

Summary 4

Task Accomplished 4

Knowledge Gained from the project 4

Something that can be improved 4

Reference 5

Introduction

Project Goal

The goal of this project is to design a thread scheduling subsystem for the prototype OS to create, schedule, maintain, and terminate multiple threads on an Altera DE2 board. This system will only support single-thread execution.

Problem Description

For this project, the focus is on designing a thread scheduling subsystem. The problem we need to solve is that creating each thread successfully, destroying each thread clean and gracefully after finishing using them, and scheduling them under different conditions.

Project Management

The following is the timeline of finishing this project:

11/8/2015	Read the Project II description and understand the basic ideas
11/9//2015	Create the project and establish the basic functions provided by the handout
11/11/2015	Writing the structure of TCB and the thread create function
11/12/2015	Writing the thread scheduler
11/13/2015	
11/14/2015	Debugs and fix the errors exist in the program
11/15/2015	
11/16/2015	
11/17/2015	
11/18/2015	Writing the final report

Table 1 Time Line

Details of the Thread Scheduler

Overview

In order to develop this system, the program is written in the combination of C and Assemble language. The body part is written in the C language. And the interrupt exception will be written in assemble language.

Body Part (In C Language)

As mentioned above, the body part of the program is written in C. Since this program needs to schedule the threads, a new data type thread control block is needed to be created. In order to to fulfill this purpose, a header file called “TCB.h” is designed. The header file includes the structure of the datatype TCB and the functions used to create, join and destroy each thread.

The TCB structure includes the 4 parts:

- Thread ID — The ID of each thread
- Scheduling Status — The status of each thread, which are ready, suspended and died
- Execution Information — The time of the thread executed
- Execution Context — It performs like a stack pointer and push necessary addresses on it

Table 2 is the function description of the "TCB.h".

Function Name	Function Description
mythread(int thread_id)	This function is what each thread needs to performed
TCB *mythread_create()	This function is used to create a thread
TCB *mythread_join()	This function is used to perform each thread one by one
TCB *mythread_destroy()	This function is used to destroy the thread which finishes running

Table 2 Description of Functions in TCB.h

The mythread_create is the key function. When this function is called, a new thread will be called. The thread ID will be assigned. The scheduling status will be assigned to Ready. The execution information will be initialized to 0 first. For the execution context, it will be an alt_u32 type array. It will be performed as a virtual stack. The address of the functions mythread_destroy and mythread will be stored on it. And then it will be called in the main part.

The main part contains four functions. Table 3 is the function description of the "main.c".

Function Name	Function Description
int main()	The main function
void os_prototype()	The OS, which will be called in the main
void *mythread_scheduler(void *context)	Used to scheduler the threads
alt_u32 myinterrupt_handler(void *context);	Use to start interrupt
int global_flag_checker();	Used to check the value of global flag

Table 3 Description of Functions in main.c

The only functions called in main is the os_prototype. In the os_prototyoe function, there are 12 threads will be created and joined. Then they will wait to be scheduled by the function mythread_scheduler. And the interrupt will be done by the myinterrupt_handler function and the assemble language file introduced below.

Interrupt (In Assemble Language)

There is a assemble file called alt_exception_entry.S has already been provided in the altera library. So, this file can be picked and used directly. However, in order to perform the mythread_scheduler function. The ".section .exeception.exit.user "xa"" part needs to be edited. However, the alt_execption_entry.S should not be overwritten. So a new asm file is created called myScheduler.S. And it will replace that part during that part.

The function of the myScheuler.S is let the mythread_scheduler in the main part being called when there is an exception occurred.

Figure 1 shows how the interrupt executes.

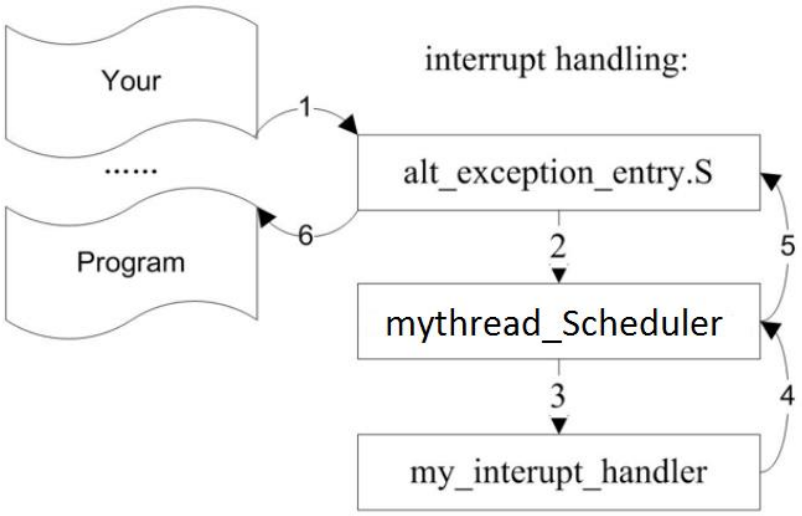

Figure 1 Interrupt Steps

In the figure 1, it is clear that when the interrupt occurred, the program will go through the assembler file and call the mythread_scheduler function.

Summary

Task Accomplished

For this project, there are 3 tasks that have been finished. The main thread can start and the timer interrupt is properly handled. Each thread can be run to completion individually. And the main can be run infinitely.

However, there is still some errors occurred in the program. For the mythread_destroy function, it does not work correctly. Also, the stack pointer in each TCB will get corrupted. So it does not load correctly at the second time running. This is the problems that needs to be fixed.

Knowledge Gained from the project

In the class, I have learnt the concepts of thread and semaphore. However, I have never practiced. This project provided the chances. Through this project, I have learnt how to schedule the thread in real board and how these concepts can be applied on them.

Something that can be improved

This project is good. Students can learn many things through this project. However, I still think that there are one thing that can be improved. Since this program combines the C language files with the assemble language file. This is the first time for us to do that. So if more instruction can be provided on that part could be better.

Reference

Figure 1: Interrupt Steps1
Source: https://my.unl.edu/bbcswebdav/pid-3194345-dt-content-rid-27615738_1/courses/CSCE351001.1158/description.pdf