EE 472 – Embedded Systems

Dr. Shwetak Patel

Agenda

- Lab 2 overview
- Operating System
  - Tasks
  - Threads
  - Interrupts
  - Scheduling

Embedded Software

- Execute a specialized program
  - Perform calculations
  - Access memory
  - Communicate with external devices

Operating System

- Makes your life easier!
- Executes your code
- Does memory management
- Handles I/O and communications within your program
An Embedded Program

- Task
  - Unit of code
  - Think of it as a C function

Scheduler

- Part of the OS that execute the tasks so they all get a turn.
- Make decisions about when to run the tasks

Types of Tasks

- Periodic
  - Found in hard-real time applications
  - Three attributes:
    - P, Period
    - C, Computing Resources
    - D, Deadline

Types of Tasks

- Intermittent
  - Found in all applications
  - Two attributes:
    - C, Computing Resources
    - D, Deadline
Types of Tasks

- Background
  - Soft-real time
  - Lower priority
  - One attribute:
    - C, Computing Resources

- Complex
  - Continuous need for CPU
  - Frequent requests for I/O
  - Wait for user input

Multitasking

- Multiple processes running in a program

Task States

- Ready
- Running
- Waiting (for something other than CPU)
  - Preempted or blocked
- Inactive/Terminated
### Context Switch
- A new task is switched in
- Need to save the current context
  - Code
  - Data
  - Registers
  - Resources (memory, io, etc)

### Threads
- Smaller unit of computation
- Owned by a process
- Like pieces within a process
- Smaller context

### Processes and Threads

**Example**
- Web server
Real-Time Operating System
- Deterministic
- Rigid time constraints

Scheduler
- Responsible for dispatching/executing tasks
- When and how long to run a task
- Scheduling strategies
  - Time sharing
  - Priorities

Representing a Task
- Task Control Block (TCB)
  - Contains information about a task
  - Think of a struct

TCB

<table>
<thead>
<tr>
<th>Pointer</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process ID</td>
<td></td>
</tr>
<tr>
<td>Program Counter</td>
<td></td>
</tr>
<tr>
<td>Register Contents</td>
<td></td>
</tr>
<tr>
<td>Memory Limits</td>
<td></td>
</tr>
<tr>
<td>Open Files</td>
<td></td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>
TCB

// The task control block
struct TCB
{
    void (*taskPtr)(void* taskDataPtr);
    void* taskDataPtr;
    void* taskPtr;
    unsigned short priority;
    struct TCB* nextPtr;
    struct TCB* prevPtr;
};

// The task
void aTask(void* taskDataPtr)
{
    function body;
}

// This data passed into the task
struct taskData
{
    int taskData1;
    int taskData2;
    char taskData3;
};

Task or Job Queue

• Holds a collection of TCBs when a task enters a system
• Array or list of structs

// Building a simple OS kernel - step 2
#include <stdio.h>

// Declare the prototypes for the tasks
void aTask(void* taskData);
void bTask(void* taskData);
void cTask(void* taskData);

// Declare a TCB structure
typedef struct
{
    void* taskPtr;
    void* taskDataPtr;
}

void main(void)
{
    int i;
    Task* taskData = &taskData;
    intTaskPtr = &get;
    outTask.taskPtr = display;

    // Initialize the TCBs
    aTask(taskData); bTask(taskData); cTask(taskData);

    // Declare some TCBs
    TCB aTask;
    TCB bTask;
    TCB cTask;
    TCB* aTCBPtr;
    TCB* bTCBPtr;

    // Schedule and dispatch the tasks
    while(1)
    {
        aTCBPtr = queue[0];
        if(aTCBPtr->taskDataPtr)
        {
            aTCBPtr->taskPtr(aTCBPtr->taskDataPtr);
            i = (i+1)%3;
        }
        return;
    }
Problems with this approach?

- Wastes time waiting for input
- Need to use interrupts

Interrupts

- External event
- Causes a context switch
- Each interrupt has a specific ID
- Interrupts are found in an interrupt vector table

Interrupt Vector Table

- Interrupt number
- Interrupt service routines (ISR)
Questions?

- Next time:
  - Scheduling algorithms
  - Print out scheduling worksheet

![Scheduling Algorithm Worksheet](Image)