Group 25: Alvin Zheng (dongxuez) Minghao Wang (minghaow) Yining Yang (yiningy) **Date:** 2013 – 12 – 04

Lab 4: Real-Time Operating Systems

18342 Fundamentals of Embedded Systems

1. Introduction

In this project, we have ported our SWI/IRQ handler, I/O syscalls and timer driver from Lab 3 to this new infra-structure. We build up a real-time operating system by implementing scheduler, dispatcher, run queue, and device emulator. Also, we write the mutex and UB test function to enforce concurrency and schedulability. Finally, we run the test cases to successfully demonstrate our system.

1.1 Overview

Our kernel implements 5 new features as follow:

- Scheduler
  - sched.c
- Dispather
  - ctx_switch.c
  - ctx_switch_asm.S
- Run queue
  - run_queue.c
- Device emulator
  - device.c
- Mutex
  - mutex.c

And 2 new syscall functions: event_wait, task_create and ub_test.

2. Detailed Design

2.1 Scheduler

allocate_tasks(task_t tasks, size_t num_tasks)**

1) Allocate user-stacks and initializes the kernel contexts of the given threads.
2) During initialization, r4 is assigned lambda, r5 data, r6 stack position, lr launch_task, sp kstack_high.
3) lr assigned the address of launch_task makes it possible to run launch_task when the task is scheduled for the first time.
4) Add the task into the run queue to make it runnable.

sched_init(void)

1) Initialize the idle task and add it into the run queue.
2) Call the dispatch_init function to turn the real-time system on.

2.2 Dispatcher

dispatch_init(tcb_t* idle)

1) Set the initialization thread's priority to IDLE.
2) Call dispatch_nosave to context switch to the highest priority available task.

dispatch_nosave(void)

1) Set the current tcb to be the next highest priority task that is not this task.
2) Context switch by calling ctx_swich_half function.

dispatch_save(void)

1) Check if the current task is the highest priority task, if so, nothing needs to do.
2) Set the current tcb to be the next highest priority task that is not this task.
3) Context switch by calling ctx_swich_full function.

dispatch_sleep(void)

1) Remove current task from run_queue
2) Set the current tcb to be the next highest priority task that is not this task.
3) Context switch by calling ctx_swich_full function.

ctx_switch_half(new_context, old_context)

Save the old context by using “stmia” instruction and load the new one by “ldmia”.

ctx_switch_full(new_context)

Load the new context by using “ldmia” instruction.

2.3 Run queue

runqueue_add(tcb_t* tcb, uint8_t prio)

runqueue_remove(uint8_t prio)

Adapt the run_bits, group_run_bits and run_list for the input tcb using OSTCBX and OSTCBY methods.

highest_prio(void)

Retrieve the highest priority using the pre-defined priority map table.

2.4 Device emulator

dev_wait(unsigned dev)

1) Add the current task to the device sleep queue.
2) Call dispatch_sleep the context switch to the next highest priority task.

dev_update(size_t millis)

1) Check if any device frequency corresponds to the current time, if not, nothing needs to do.
2) Update the device next match value based on the current time.
3) Transfer all the tasks in the device sleep queue to the run queue.
4) If any task is transferred, call the dispatch_save function to context switch to the highest priority tasks, if necessary.

2.5 Mutex

mutex_init(void)

Initialize all the mutex and the mutex counter. Specifically, make the bAvailable and bLock flags FALSE, pHolding_Tcb and pSleep_queuue pointers NULL.

mutex_create(void)

Add the mutex counter and fix the bAvailable flag to be TRUE.

mutex_lock(int mutex)

1) Verify the mutex number and current locking aspects.
2) If the mutex is free, set the flags and pump the current task priority up to the highest one.
3) If the mutex is busy, add the current task to this mutex sleep queue and call the dispatch_sleep to context switch to the next highest priority task.

mutex_unlock(int mutex)

1) Verify the mutex number and current locking aspects.
2) If no other tasks waiting, set the flags and jump to the cleanup.

3) If any task waiting for the lock, release one task from the sleep queue and add it to the run queue.
4) Cleanup: recover the current task priority if it has released all the locks declared before.

2.6 Syscalls

task_create(task_t* tasks, size_t num_tasks)

1) Verify the tasks structure and num_tasks.
2) Call assign_schedule function to sort the task and perform UB test.
3) Call allocate_tasks to initialize the tcb structures.

event_wait(unsigned int dev)

1) Verify the dev number and check if the current task holds a mutex lock.
2) Call dev_wait function to add the current task to the device sleep queue.

2.7 UB test

assign_schedule(task_t tasks, size_t num_tasks)**

1) Outside the function, we have a static pre-defined table of UB test values.
2) Sort the tasks structures in priority order.
3) Compute the iterative sum of C/T, and the total of the sum and B/T. Once if the total exceeds the corresponding UB test value, the UB test fails.

APPENDIX 1 – Memory Layout