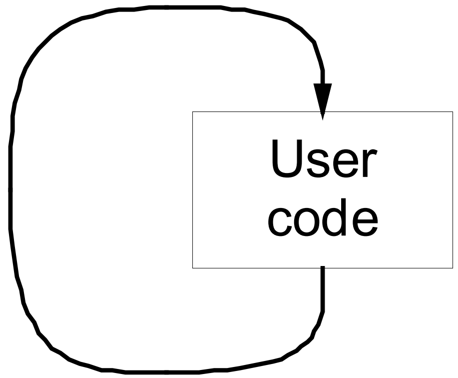
University of Strathclyde Engineering

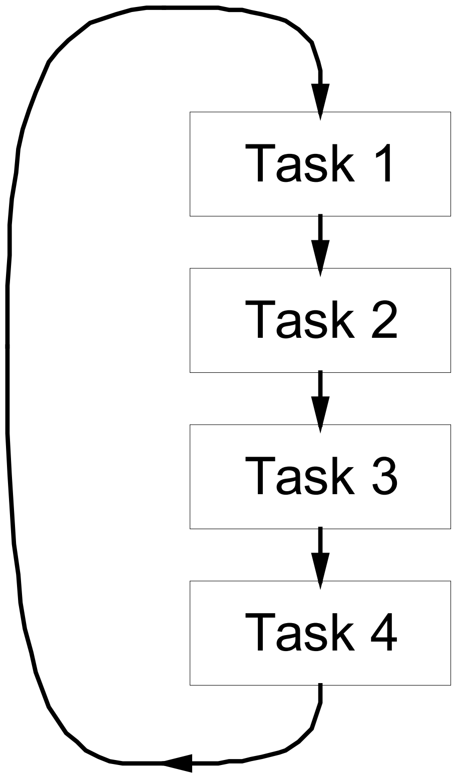
Lecture 5: Multitasking

James Irvine

Simple Microcontroller System



Multiple Tasks

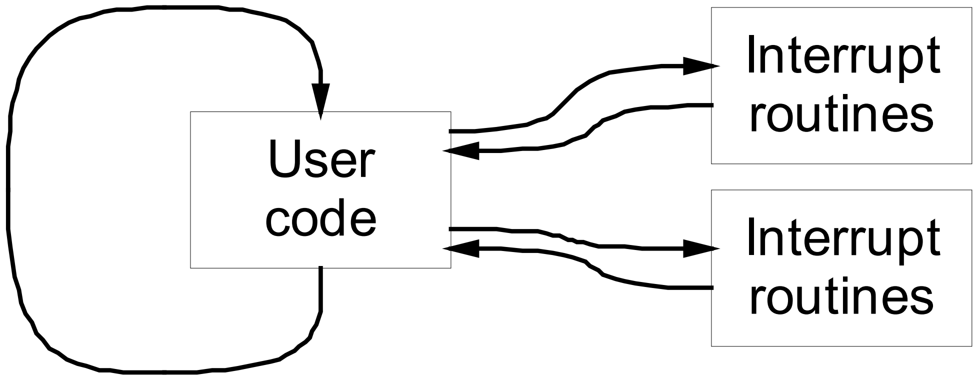


Cyclic Executive (Polling)

- Tasks execute sequentially
  - No arbitration/external control required
- Advantages
  - Simple to design
  - Highly predictable
- Disadvantages
  - Must operate in order
  - Doesn’t cope with infrequent events efficiently

Interrupts

University of Strathclyde Engineering

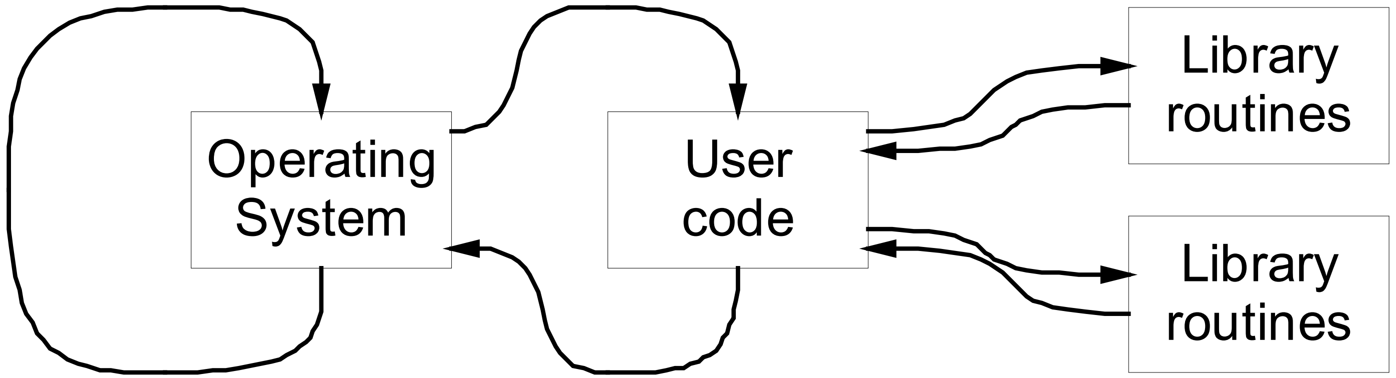


Using Interrupts

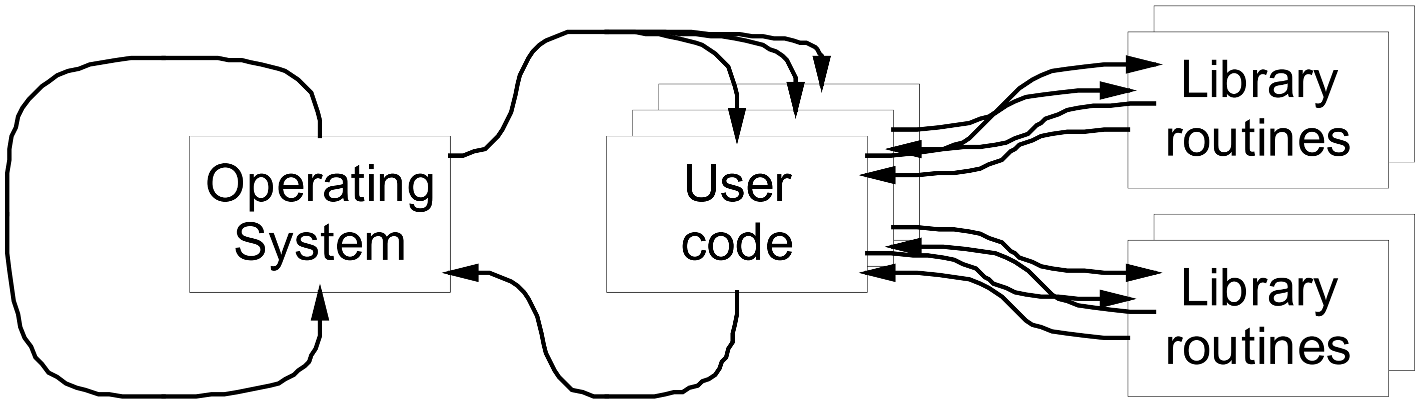
University of Strathclyde Engineering

- Allows basic cycle to be amended
- Copes with infrequent events well, but
  - Takes time to respond to the interrupt
  - Only very basic scheduling available (number of interrupt levels)
- Interrupt routine itself usually does relatively little
  - Sends message to main routine that an event occurred

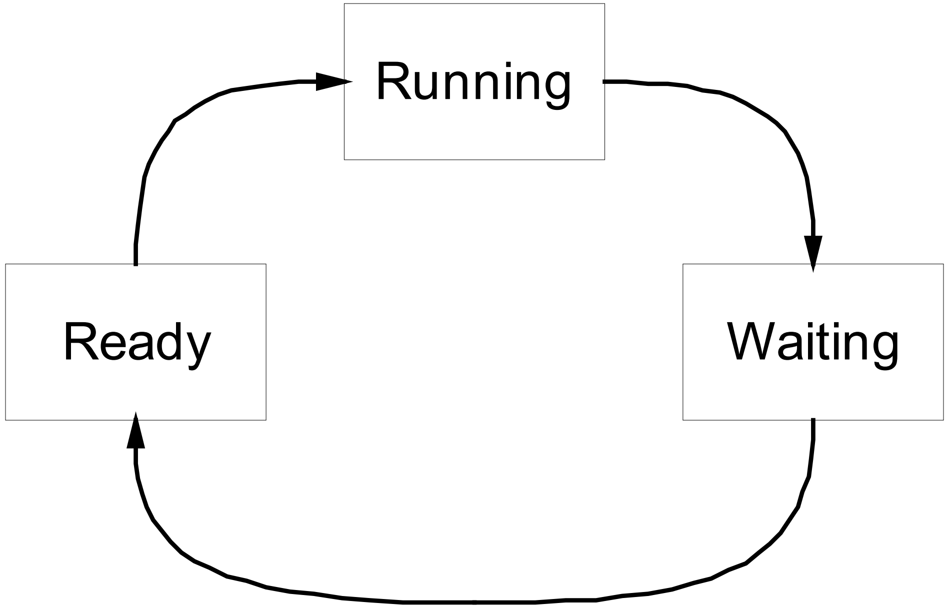
Using an Operating System



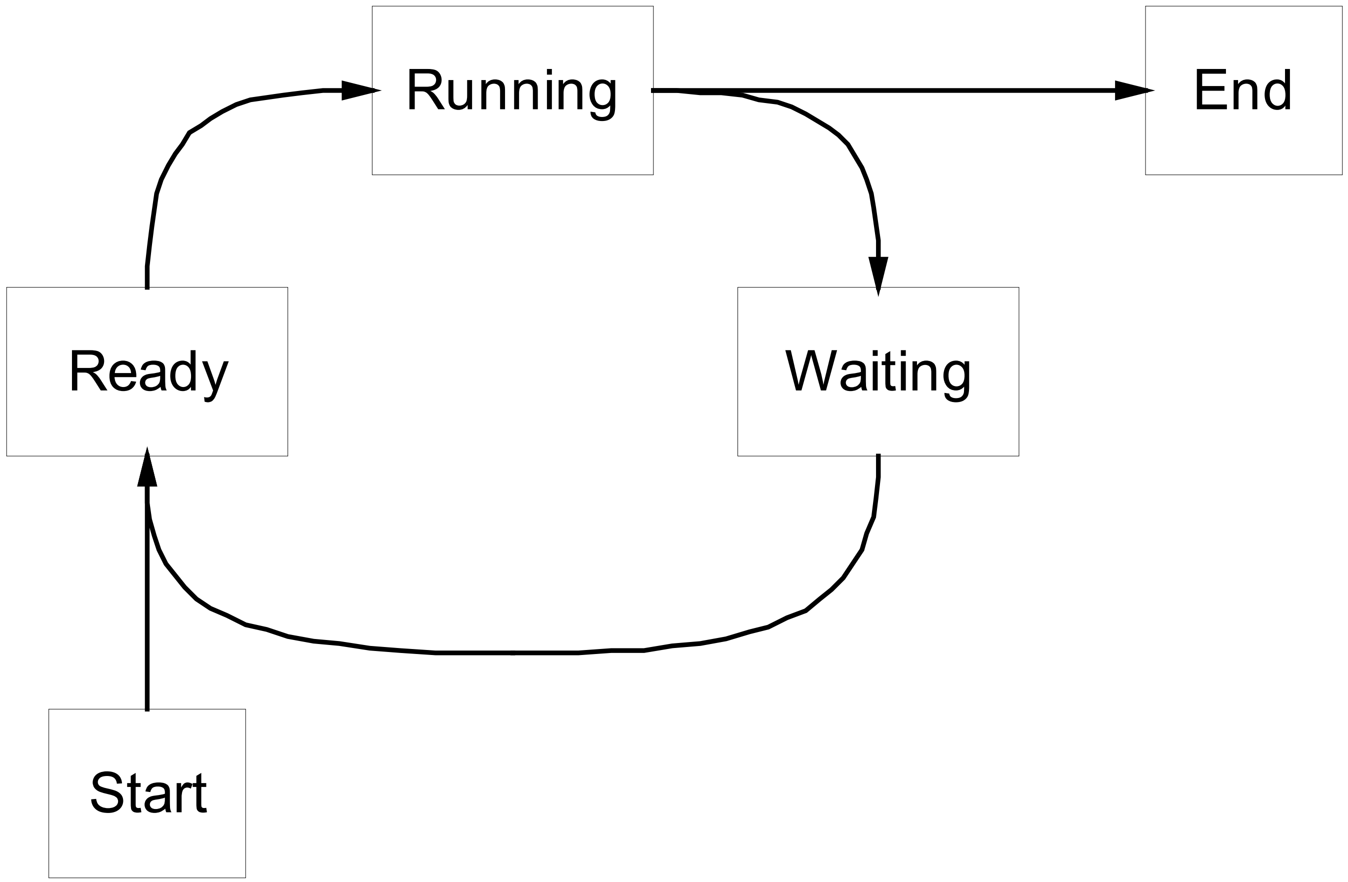
Multiple Tasks



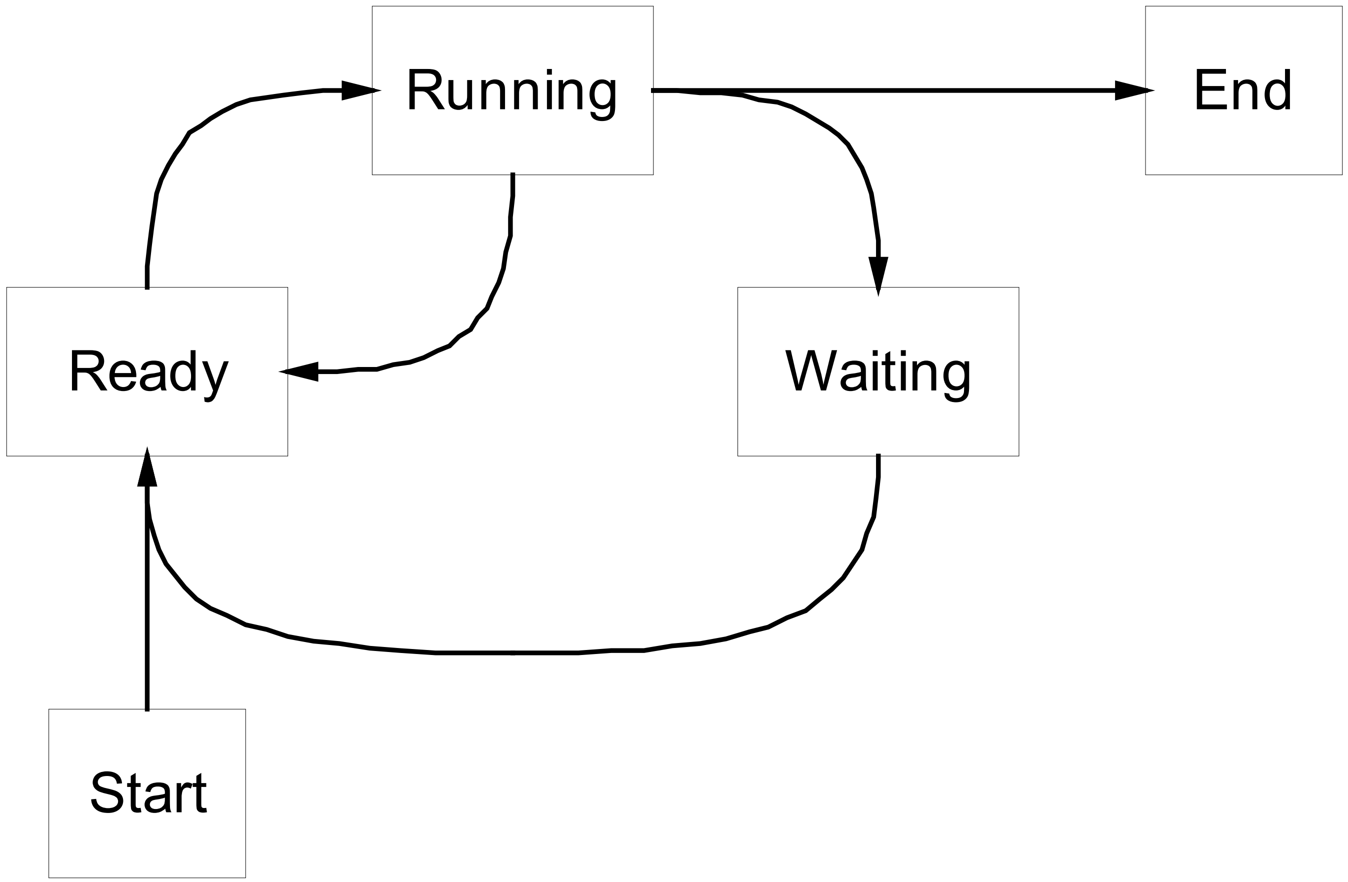
Basic Multitasking System (Co-operative Multitasking)



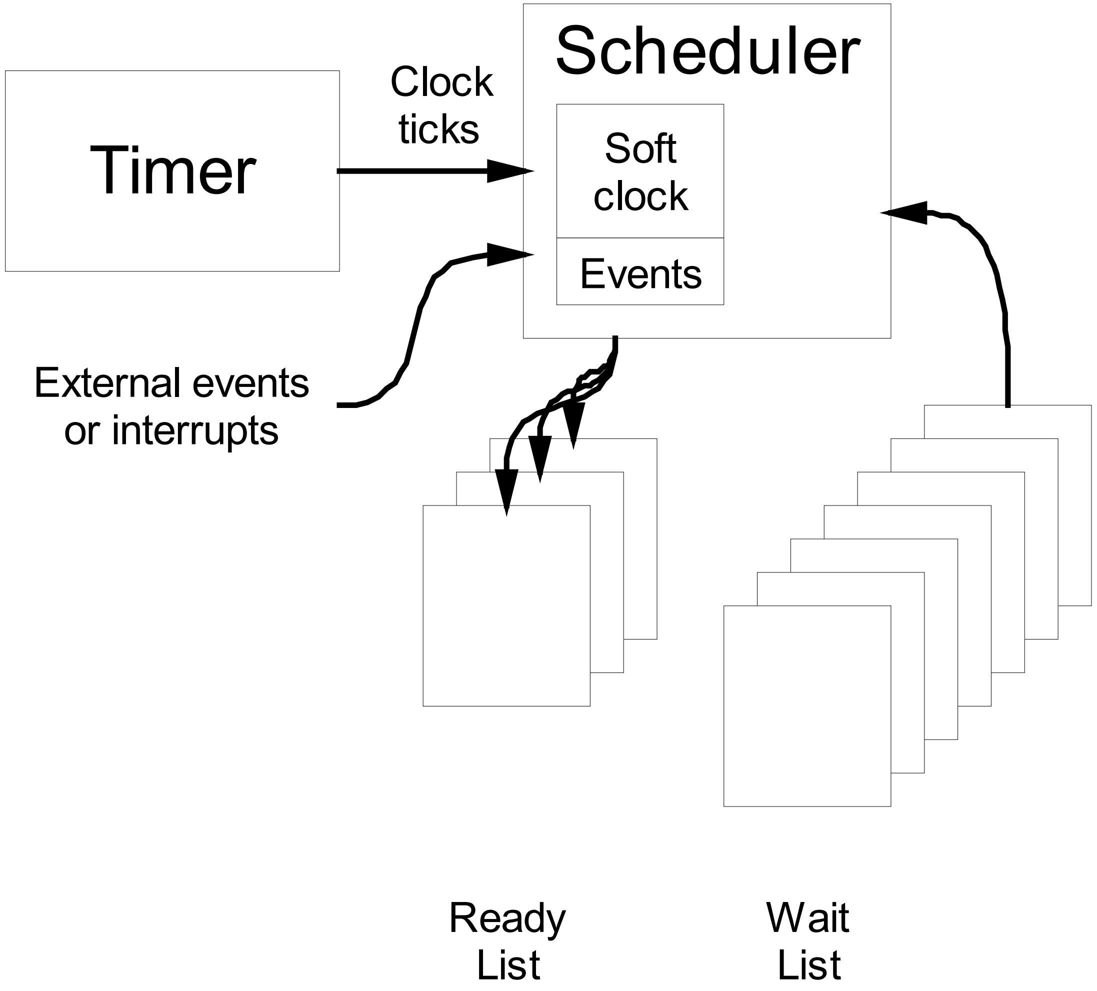
Starting and Stopping Tasks



True Multitasking



True Multi-tasking OS



Context

University of Strathclyde Engineering

Stored in a Task Control Block

- Program counter
- Status register
- Other registers
- **Stack pointer**

- Workspace (usually private, so not saved)

Task Scheduling

University of Strathclyde Engineering

- Non-priority
  - FIFO (co-operative multitask only)
  - Shortest job first (co-operative multitask only)
  - Round robin (True multitasking)
- Priority based (both true multitasking)
  - Interrupt based (fairly crude…)
  - Scheduled (can be very flexible, but watch latency)

Scheduler

- Schedule()
  - Reorder ready list based on priorities
  - Called whenever a task is created, deleted, blocked or released

- ClickTick()
  - Suspend current task
  - Start next task

  - Called from an interrupt routine (almost always a timer)

Task Switching

University of Strathclyde Engineering

In an interrupt

- Store the state of the current task on the stack
- Store the SP with the Task Control Block of the task
- Load the SP with the SP from the TCB of the new task
- Load the state of task B from the stack
- RETI
  - the PC will be restored based on the SP of the new task

Software Clock

University of Strathclyde Engineering

- Very useful (although not totally necessary addition)
- Allows tasks to suspend themselves for times greater than a cycle
- Suspend()
  - Add the calling task to the wait list for that time
  - Set the task state to waiting
  - Reschedule
- Each clock tick then checks to see if tasks waiting for that tick
  - If so, changes their state from waiting to ready, and reschedules

Task Synchronisation

- Tasks operate completely asynchronously
  - Totally different paradigm from cyclic executive
- Causes problems when tasks not independent
- Several tasks accessing one resource
- Obvious for resource use (printer, etc)
- More subtle for resource update

Task Synchronisation

- Consider Mr and Mrs Smith’s joint current account
- Withdrawing £100

```
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
```

Task Synchronisation

University of Strathclyde Engineering

- Consider Mr and Mrs Smith's joint current account
- Both withdraw £100

Mr Smith

```
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
```

Ms Smith

```
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
```

- If balance > £100 but < £200, the account will go overdrawn

Task Synchronisation

- Better code
- Lock account before change
- Respect locking
- Release after change

```
if (!smithaccount.lock)
  smithaccout.lock = TRUE;
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
Smithaccount.lock = FALSE;
```

Task Synchronisation

- Still doesn't work
- Both withdraw £100

Mr Smith

```
if (!smithaccount.lock)
  smithaccout.lock = TRUE;
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
Smithaccount.lock = FALSE;
```

Ms Smith

```
if (!smithaccount.lock)
  smithaccout.lock = TRUE;
if (smithaccount.balance > 100)
  smithaccount.balance -= 100;
Smithaccount.lock = FALSE;
```

- If balance > £100 but < £200, the account will go overdrawn

Task Synchronisation

- Fails because another task accessed the flag between the test and the update
- Solution – have atomic functions for flag updates; can't be interrupted

Task Synchronisation

- Mutex
  - One resource
- Semaphores
  - Many resources (all equivalent)
- Message queue
  - No blocking – one way information flow to a process

Mutex

University of Strathclyde Engineering

- Multitasking invariant context
  i.e. a semaphore which is preserved for all tasks
- Depends on an atomic function - cannot be interrupted (and therefore switched from)

Semaphore

University of Strathclyde Engineering

- Similar to a mutex, but allows for a number (>1) of equivalent resources to be secured
- Initialised to the number of resources available
- GetSemaphore:
  - If > 1, decrement and return
  - Else wait until >1, decrement and return
- ReleaseSemaphore
  - Increment and return

Mutex with Queue

- More complex system, consisting of

- Mutex (binary semaphore)
- List of waiting tasks

- Allows the calling function to get the mutex to be made blocking - it will always return with it

Get

- If the mutex is available, set it and return
- Else
  - Add the calling task to the wait list (either by priority of FIFO)
  - Set the task state to waiting
  - Reschedule

Release

- If ~held, return
- If waitlist is not empty,
  - Get first task in the list
  - Set its status to ready
  - Return
- Else
  - Set mutex state to available

Task Priorities

University of Strathclyde Engineering

- Tasks locking resources can give problems

- Runnable high priority task usually pre-empts low priority task

BUT

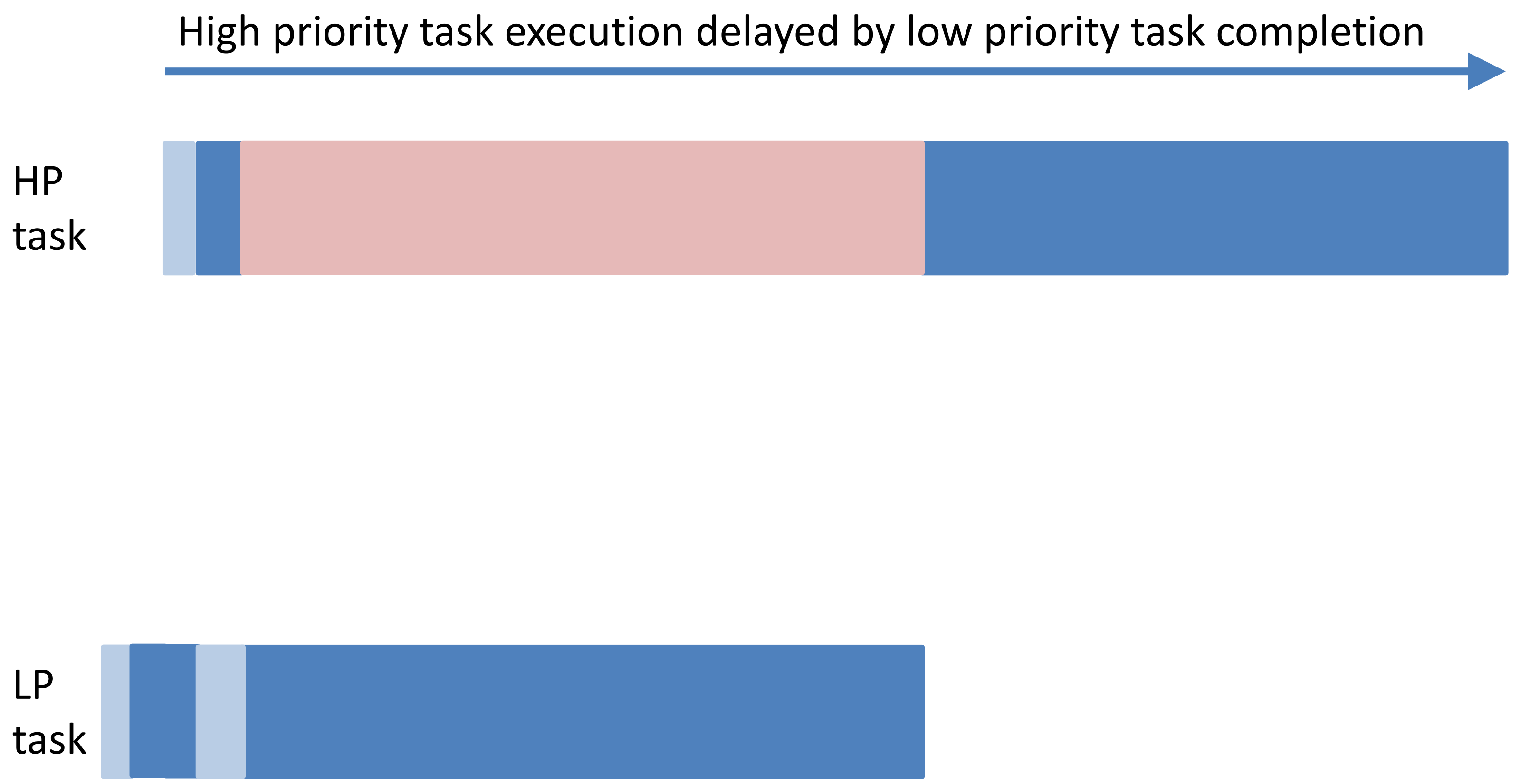
- High priority task must wait for low priority task, if it holds a resource it needs
- Low priority tasks need to lock resources for minimal time

Priority Inversion

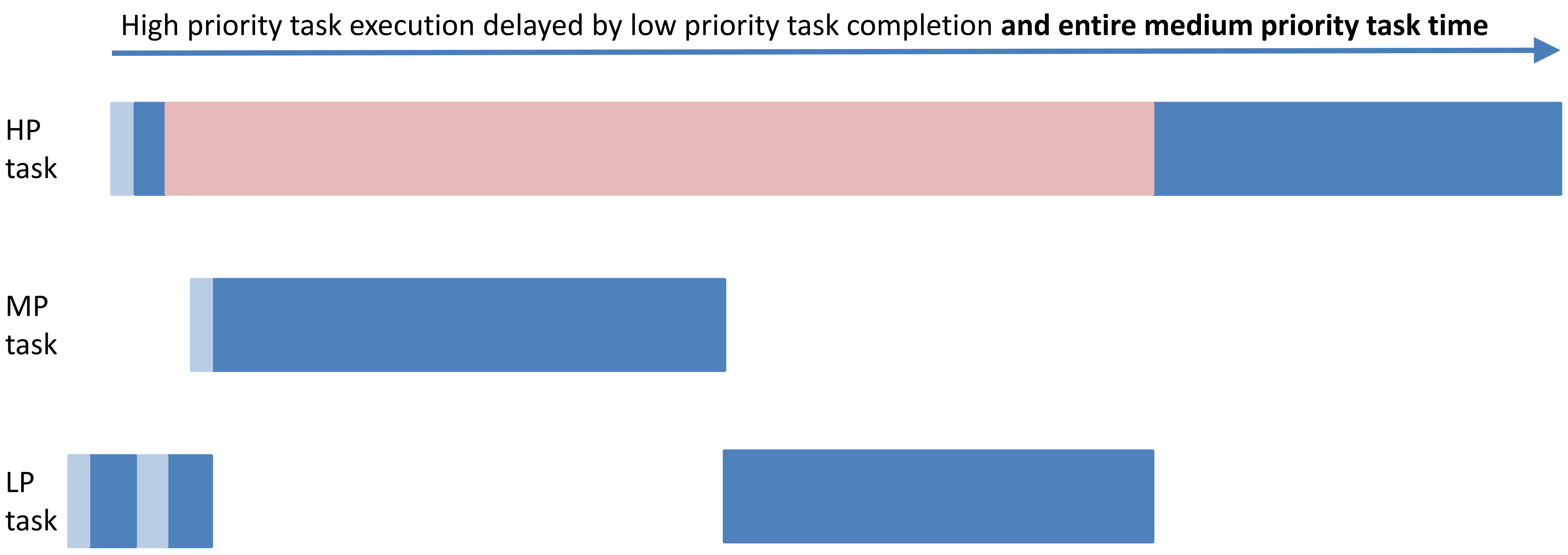
- High priority task delayed beyond the minimum time of running low priority task

- Low priority task runs and locks resource
- High priority task blocked, low priority continues
- Medium priority task not needing the resource pre-empts low priority task
- High priority task must now wait for *both* tasks to complete

Priority Inversion



Priority Inversion



Real Time Operating Systems

University of Strathclyde Engineering

- Complete OS with task scheduler, priorities, etc
- Available for most microcontrollers
  - TI has one for the MSP430 (although not the 2553)
  - Free RTOS has been ported to the MSP430

- The 2553 is very small though
  - Use the principles to build an optimised system for your program
  - Don't need to use a task queue for example, just an array

As an example…

- Write a program to alternately flash the red LED either every second or twice a second.
- The program should toggle between the flashing rate on each press of the switch.

What needs to happen…

- Implement a clock
- On switch press, change LED mode
- On LED time, toggle the LED

Activities

- On clock tick (1ms)
  - Update time, check for events
- On switch press
  - Start debounce
- On debounce end
  - Check valid
  - If so, change LED mode
- On LED time
  - Toggle LED, schedule next

Implementation

- Interrupts
  - Timer - 1ms - clock tick
    - Update time
    - Check events
  - Switch
    - schedule debounce
- Schedules
  - Debounce end
    - Check switch press valid
    - Change LED mode
    - Reset schedule
  - LED time end
    - Toggle LEDs
    - Schedule next time

Time Code & Macros

```
struct Time {
   int sec;
   int ms;
};

#define IsTime(X) ((CurrentTime.sec == X.sec) && (CurrentTime.ms == X.ms))

if (CurrentTime.ms++ == 1024)
 {
   CurrentTime.ms = 0;
   if (CurrentTime.sec++ == 60) CurrentTime.sec = 0;
 }
// Who cares what the minute is in this example?
```

```c
#include "io430.h"

#define IsTime(X) ((CurrentTime.sec == X.sec) && (CurrentTime.ms == X.ms))

#define LEDBit BIT0
#define SwitchBit BIT3
#define ClockPeriod 16                          //Clock tick internal in 16384's of a second

struct Time {
   int sec;
   int ms;
};

struct Time CurrentTime = {0,0};
struct Time ADCSchedule = {0,-1};
struct Time LEDSchedule = {0,512};
struct Time SwitchSchedule = {0,-1};

int LEDPeriod = 512;

//Function to return the current time plus the duration in ms

struct Time Schedule (int duration)
{
  struct Time newtime;
  newtime.sec = CurrentTime.sec;
  newtime.ms = CurrentTime.ms+duration;         //add in the duration in ms
  while ((newtime.ms-=1024) >= 0)               //subtract whole seconds until negative
    if (newtime.sec++ == 60) newtime.sec = 0;   //adding to seconds each time
                                                //roll seconds over at a minute
  newtime.ms+=1024;                             //add back the extra 1024 that made us negative
  return newtime;                               //and return that time
}
```

```
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= LEDBit;                          // Make the LED bit(s) output
  P1OUT |= LEDBit;                            // Start them high
  P1OUT |= SwitchBit;                       // Select pull up resistor on switch
  P1REN |= SwitchBit;                       // and enable
  P1IES |= SwitchBit;                       // Switch high to low edge
  P1IFG &= ~SwitchBit;                      // Switch IFG cleared
  P1IE |=  SwitchBit;                       // Switch interrupt enabled
  TA0CCTL0 = CCIE;                          // CCR0 interrupt enabled
  TA0CCR0 = ClockPeriod;                             // Have time tick every ~ms
  TA0CTL = TASSEL_1 + MC_1;                 // ACLK, upmode

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0 w/ interrupt
}
```

```
// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  if (CurrentTime.ms++ == 1024)                          // Update soft clock. Has second passed?
  {
    CurrentTime.ms = 0;                                  // If so, reset milliseconds and count a sec
    if (CurrentTime.sec++ == 60) CurrentTime.sec = 0;    // If now a minute, reset sec to 0
  }
                                                         // Check for scheduled events
  if (IsTime(LEDSchedule))                               // Is it time for the LED event?
  {
    P1OUT ^= LEDBit;                                     // If so, toggle the LED
    LEDSchedule = Schedule(LEDPeriod);
  }
  if (IsTime(SwitchSchedule))                            // Is it time for the switch debounce?
  {
    if (P1IN & SwitchBit)                                // If so, is it a valid press (still pressed)
    {
      LEDPeriod ^= 0x0180;                               // If so, toggle between 512 and 256
    }                                                    // Note will only affect next time
    SwitchSchedule.ms = -1;                              // Turn off the debounce event
  }
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port1 (void)
{
    P1IFG &= ~SwitchBit;                                 // Clear switch interrupt flag
    SwitchSchedule = Schedule(10);                       // Schedule an end debounce in ~10ms
}
```

More complex example…

University of Strathclyde Engineering

- Write a program to alternately flash either the red and green LEDs, or the multicolour LED either red or green.
- The speed of the flashing should be controlled by the pot and at one end of the range the flashing should stop.
- The program should toggle between flashing the red and green LEDs or the multcolour LED on each press of the switch.

What needs to happen…

- Implement a clock
- On switch press, change LED mode
- On LED time, toggle relevant LEDs
- Regularly update pot value

Activities

- On clock tick (1ms)
  - Update time, check for events
- On switch press
  - Start debounce
- On debounce end
  - Check valid
  - If so, change LED mode
- On LED time
  - Toggle relevant LED, schedule next
- Every half second
  - start ADC, schedule next
- On ADC complete
  - update LED time

Implementation

- Interrupts
  - Timer - 1ms - clock tick
    - Update time
    - Check events
  - Switch
    - schedule debounce
  - ADC
    - update LED time
- Schedules
  - Debounce end
    - Check switch press valid
    - Change LED mode
    - Reset schedule
  - LED time end
    - Toggle relevant LEDs
    - Schedule next time
  - Half second
    - Start ADC
    - Schedule next time

University of Strathclyde Glasgow