Fixed scheduling
In advance a priori, during the design phase
   Thread sequence
   Allocated time-slices

Like
   creating the city bus schedule, or
   routing packages through a warehouse
   Construction project.

Fundamental principles
   Gather reliable information about the tasks
   Build slack into the plan; expect delays; anticipate problems
   Just in time

What we do first is create a list of tasks to perform
   1. Assigning a priority to each task,
   2. Defining the resources required for each task,
   3. Determining how often each task is to run, and
   4. Estimating how long each task will require to complete.

Consider resources required versus available
   Processor
   Memory
   I/O channels
   Data

Objectives
   guarantee performance (latency, bandwidth)
   utilization
   Maximize profit

Strategy
   schedule highest priority tasks first,
   100% satisfaction guaranteed
   then schedule all real-time tasks
   shuffle the assignments like placing pieces in a puzzle
   maximizing objectives
   The tasks that are not real-time can be scheduled in the remaining slots.

Design example
   finite state machine (FSM),
   proportional-integral-derivative controller (PID),
   data acquisition system (DAS),
   non-real-time task, PAN
Figure 4.16. Real-time specifications for these three tasks.

\[ \sum_{i=0}^{n-1} \frac{E_i}{T_i} = \sum_{i=0}^{n} \left( \frac{100}{2000} + \frac{300}{1000} + \frac{50}{1500} \right) = 0.38 \leq n \left( \frac{2^{1/n} - 1}{3} \right) = \left( \frac{2^{1/3} - 1}{3} \right) = 0.78 \]

To guarantee tasks will run on time, consider the maximum times

Process

Look for a repeating pattern least common multiple of 2000, 1000, and 1500

- time-shift the second and third tasks
- Avoid overlaps,
  - start with the most frequent task (or most frequent task)

Figure 4.17. Repeating pattern to schedule these three real-time tasks.

How it works

OS_Suspend

Cooperatively stops a real-time task
Runs a non real-time task

Timer interrupt

- Occurs when it is time to run a real-time task
- Suspends a non-real-time task
- Runs the next real-time task

//******************FSM**************************
void FSM(void){ StatePtr Pt; unsigned char in;
  Pt = SA; // Initial State
  for(;;) {
    OS_Suspend(); // Runs every 2ms
    Port_Out(Pt->Out); // Output depends on the current state
}
in = Port_In();
Pt = Pt->Next[in];     // Next state depends on the input
}

void PID(void){ unsigned char speed,power;
PID_Init();              // Initialize
for(;;) {
    OS_Suspend();          // Runs every 1ms
    speed = PID_In();      // read tachometer
    power = PID_Calc(speed);
    PID_Out(power);        // adjust power to motor
}

void DAS(void){ unsigned char raw;
DAS_Init();            // Initialize
for(;;) {
    OS_Suspend();        // Runs every 1.5ms
    raw = DAS_In();      // read ADC
    Result = DAS_Calc(raw);
}

void PAN(void){ unsigned char input;
PAN_Init();            // Initialize
for(;;) {
    input = PAN_In();    // front panel input
    if(input){
        PAN_Out(input);    // process
    }
}
}

Program 4.19. Four user threads.

struct TCB{
    unsigned long *StackPt;       // Stack Pointer
    unsigned long MoreStack[84];  // 400 bytes of stack
    unsigned long InitialReg[14]; // R4-R11,R0-R3,R12,R14
    unsigned long InitialPC;      // pointer to program to execute
    unsigned long InitialPSR;     // 0x01000000
};
typedef struct TCB TCBType;
TCBType *RunPt;            // thread currently running
#define TheFSM &sys[0]       // finite state machine
#define ThePID &sys[1]       // proportional-integral-derivative
#define TheDAS &sys[2]       // data acquisition system
#define ThePAN &sys[3]       // front panel
TCBType sys[4]={[
    { &sys[0].InitialReg[0],{ 0 }, FSM, 0x01000000},
    { &sys[1].InitialReg[0],{ 0 }, PID, 0x01000000},
    { &sys[2].InitialReg[0],{ 0 }, DAS, 0x01000000},
    { &sys[3].InitialReg[0],{ 0 }, PAN, 0x01000000}
];

Program 4.20. The thread control blocks.

by Jonathan W. Valvano
struct Node{
    struct Node *Next;        // circular linked list
    TCBType *ThreadPt;        // which thread to run
    unsigned short TimeSlice; // how long to run it
};

typedef struct Node NodeType;
 NodeType *NodePt;
 NodeType Schedule[22]={
    { &Schedule[1], ThePID, 300}, // interval 0, 300
    { &Schedule[2], ThePID, 300}, // interval 300, 400
    { &Schedule[3], TheDAS, 500}, // interval 400, 450
    { &Schedule[4], TheFSM, 550}, // interval 450, 1000
    { &Schedule[5], ThePID, 300}, // interval 1000, 1300
    { &Schedule[6], ThePAN, 600}, // interval 1300, 1900
    { &Schedule[7], TheDAS, 50}, // interval 1900, 1950
    { &Schedule[8], ThePAN, 50}, // interval 1950, 2000
    { &Schedule[9], ThePID, 300}, // interval 2000, 2300
    { &Schedule[10], TheFSM, 100}, // interval 2300, 2400
    { &Schedule[11], ThePAN, 600}, // interval 2400, 3000
    { &Schedule[12], ThePID, 300}, // interval 3000, 3300
    { &Schedule[13], ThePAN, 100}, // interval 3300, 3400
    { &Schedule[14], TheDAS, 50}, // interval 3400, 3450
    { &Schedule[15], ThePAN, 550}, // interval 3450, 4000
    { &Schedule[16], ThePID, 300}, // interval 4000, 4300
    { &Schedule[17], TheFSM, 100}, // interval 4300, 4400
    { &Schedule[18], ThePAN, 500}, // interval 4400, 4900
    { &Schedule[19], TheDAS, 50}, // interval 4900, 4950
    { &Schedule[20], ThePAN, 50}, // interval 4950, 5000
    { &Schedule[21], ThePID, 300}, // interval 5000, 5300
    { &Schedule[0], ThePAN, 700}  // interval 5300, 6000
};

Program 4.21. The scheduler defines both the thread and the duration.

Results (time measurements in 1µs units)

<table>
<thead>
<tr>
<th>Finite State Machine</th>
<th>PID Controller</th>
<th>DAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>42637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>44637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>46638</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>48638</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>50638</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>52637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>54637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>56637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>58638</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>60637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>62638</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>64637</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1102</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>3101</td>
<td>10</td>
</tr>
</tbody>
</table>

by Jonathan W. Valvano
Question: Could we solve this problem with regular periodic interrupts?

See data sheet Section 11.3.4 Synchronizing GP Timer Blocks

Rate Monotonic Scheduling Theorem

- All n tasks are periodic
  - Priority based on period of $T_i$
  - Maximum execution time $E_i$
- No synchronization between tasks
- Execute highest priority task first

$$\sum \frac{E_i}{T_i} \leq n \left( 2^{1/n} - 1 \right) \rightarrow \ln(2)$$

Algorithm to find schedule with minimum jitter

Design parameters
- Period for each task $T_i$
- Maximum execution for each task $E_i$

Fundamental issues
- Find the largest $\Delta t$, and convert $T_i$ and $E_i$ specifications to integers
- Find time at which the pattern repeats, least common multiple of $T_i$

Code posted on class web site and here http://codepad.org/dGy6fk9P
http://users.ece.utexas.edu/~valvano/EE345M/ScheduleFinder.c

by Jonathan W. Valvano
Example 1
Task A runs every 1.0 ms $T_1=1.0\text{ms}$, maximum time is 0.1 ms
Task B runs every 1.5 ms $T_2=1.5\text{ms}$, maximum time is 0.1 ms
Task C runs every 2.5 ms $T_3=2.5\text{ms}$, maximum time is 0.1 ms
Task D runs every 3.0 ms $T_4=3.0\text{ms}$, maximum time is 0.1 ms

Time quanta $= \Delta t = 0.1 \text{ms}$

LCM 10, 15, 25 and 30 is 150

Let $n$ be the number of tasks

Search space is about $(T_2/\Delta t)^*(T_3/\Delta t)^* \ldots *(T_n/\Delta t)$

A

B

C

D

Time quanta = 0.1ms, pattern repeats every 15ms

$E_1/T_1 + E_2/T_2 + E_3/T_3 + E_4/T_4 = 0.24$

Search space is about $(15)*(25)*(30)=11250$ possible schedules

Schedule Task A at times 0,1,2... times $n*10$ (units 0.1ms)
Schedule Task B at times $j,j+1.5,j+3.0,...$ times $n*15+j$
Schedule Task C at times $k,k+2.5,k+5.0,...$ times $n*25+k$
Schedule Task D at times $l,l+3,l+6, ...$ times $n*30+l$

Inputs to the ScheduleFinder are

desired four periods 1, 1.5, 2.5 and 3 ms
maximum execution time 0.1 ms for each

Outputs of the ScheduleFinder are

$j$ slide factor (in 0.1ms units) for Task B
$k$ slide factor (in 0.1ms units) for Task C
$l$ slide factor (in 0.1ms units) for Task D

jitter is the number of missed tasks

Performance criteria minimum overlap (jitter) call it ScheduleFinder(10,15,25,30)

```
#define MAX 150
char Times[MAX]; // filled with spaces
char bestTimes[MAX];
int bestj,bestk,bestl;
int ScheduleFinder(int ta, int tb, int tc, int td){
    int i,j,k,l; int jitter=0;
    int thei, thejitter; // find schedule for task b with min jitter
    jitter = 100000;
    for(j=0; j<tb; j++) { // slide factor task B
        for(k=0; k<tc; k++) { // slide factor task C
            for(l=0; l<td; l++) { // slide factor task D
                for(i=0; i<MAX; i++) Times[i] = ' '; // space means time not used
                for(i=0; i<MAX; i=i+ta) Times[i] = 'a'; // schedule task a
                thejitter = 0;
            }
        }
    }
    // more code...
}
```
for(i=j; i<MAX; i=i+tb){
    if(Times[i] == ' '){
        Times[i] = 'b'; // schedule B no jitter
    } else{
        thei = i; // search for place to schedule
        do{
            thei++;
            thejitter++;
        } while(Times[thei] != ' ' && thei<MAX);
        if(thei<MAX) Times[thei] = 'B'; // schedule B with jitter
    }
}
for(i=k; i<MAX; i=i+tc){
    if(Times[i] == ' '){
        Times[i] = 'c'; // schedule C no jitter
    } else{
        thei = i; // search for place to schedule
        do{
            thei++;
            thejitter++;
        } while(Times[thei] != ' ' && thei<MAX);
        if(thei<MAX) Times[thei] = 'C'; // schedule C with jitter
    }
}
for(i=l; i<MAX; i=i+td){
    if(Times[i] == ' '){
        Times[i] = 'd'; // schedule D no jitter
    } else{
        thei = i; // search for place to schedule
        do{
            thei++;
            thejitter++;
        } while(Times[thei] != ' ' && thei<MAX);
        if(thei<MAX) Times[thei] = 'D'; // schedule D with jitter
    }
}
if(thejitter<jitter){
    bestj = j; bestk = k; bestl = l;
    jitter = thejitter;
    for(i=0; i<MAX; i++) bestTimes[i] = Times[i]; // best schedule
}
}
return jitter;
}

abcdaabbacabdbdbdabda
012345678901234567890123456789012345678901234567890123456789012345
bcabadcabcabcabcabcba5678901234567890123456789012345678901234567890123456789

Schedule Task A at times 0, 10, 20... times n*10 (units 0.1ms)
Schedule Task B at times 1, 16, 31,... times n*15 + 1
Schedule Task C at times 2, 27, 52,... times n*25 + 2
Schedule Task D at times 3, 33, 63,... times n*30 + 3
Jitter = 0

**Example 2**
Task A runs every 0.4 ms T1=0.4ms, maximum time is 0.1ms
Task B runs every 0.6 ms T2=0.6ms, maximum time is 0.1ms
Task C runs every 1.0 ms T3=1.0ms, maximum time is 0.1ms
Task D runs every 1.5 ms T4=1.5ms, maximum time is 0.1ms
Time quanta = 0.1ms, pattern repeats every 9ms

\[ \frac{E1}{T1} + \frac{E2}{T2} + \frac{E3}{T3} + \frac{E4}{T4} = 0.58 \]

Schedule Task A at times 0,4,8,... times n*4 (units 0.1ms)
Schedule Task B at times j,j+6,j+12,... times n*6 + j
Schedule Task C at times k,k+10,k+20,... times n*10 + k
Schedule Task D at times l,l+15,l+30,... times n*15 + l

call `ScheduleFinder(4,6,10,15)`
Schedule Task A at times 0,4,8,... times n*4 (units 0.1ms)
Schedule Task B at times 1,7,13,... times n*6 + 1
Schedule Task C at times 1,21,31,41,... times n*10 + 1
Schedule Task D at times 14,29,44,... times n*15 + 14

Jitter = 5

Red means one time quanta late
Blue means two time quanta late (or one time quanta early)

by Jonathan W. Valvano