

# Real-time Operating Systems and Systems Programming

Introduction  
Lecture 1

# About the Course

- Lectures by: Jaagup Irve  
jaagup.irve@ttu.ee
- Webpage:  
<http://courses.cs.ttu.ee/ITS8020/>

# Expectations

- Familiarity with C programming language
  - There will be a test on Oct 5
- Some familiarity with command-line helps

# Test

- Oct 3.
- Devious puzzles
- Everything you can do without libraries
  - C keywords
  - Precedence
  - Pointers and arrays
- Goal is to brainwash you
  - Tricks your brain into remembering things
  - Gives extra points for the exam
- Feedback on general C proficiency

# Extremely nasty C (test is easier)

```
typedef unsigned char B;char*x[]={
#include "dict.h"
0};typedef struct L{B*s;struct L*n;}L;
L*h[128],*l[128],*s[128],Z[sizeof x/sizeof*x],*F=Z;int c[256],m,a=1;
int k(B*q){int g=0;B*p=q;while(*p)g|=!c[*p++]--;return g-1&p-q;}
void u(B*p){while(*p)c[*p++]++;}
void S(int N,int r,int t,L*W){L*w;int i,n;
for(n=r<N?r:N;n>0;n--)for(w=n==N?W:h[n];s[t]=w;u(w->s),w=w->n)if(k(w->s))
if(n==r){if(t==m-1)for(i=a=0;i<=t;i++)printf("%s%c",s[i]->s,i<t?' ':'\n');}
else if(t<m-1)S(n,r-n,t+1,s[t]=w);}
int main(int C,B**A){int i=0,g,n=0;B*p;while(--C)for(p=**++A;n<127&&*p;)c[*p++]++,n++;
for(;p=x[i++];u(p))if(g=k(p))(l[g]=*(l[g]?&l[g]->n:&h[g])=F++)->s=p;
while(++m<128)S(127,n,0,h[127]);
return a;}

```

- Peter Klausler, IOCCC 2006 (<http://www.ioccc.org/>)

# Grades

- Programming project(s) (40%)
  - Small practice tasks (show them!)
  - At least one larger program
- Exam (60%)
  - Terminology, some functions, code reading, coding on paper

# Topics

- Hardware IO; interrupts
- Stack, heap
- Signals, threads, processes, mutexes
- Scheduling
- Standard IO, file, dir management
- Programming an Operating System
- Networking
- Optimizing, security, Localization

# C keywords

- Types

- char double enum float int long short struct union void

- Parameters to variables

- auto const extern register signed static unsigned volatile

- Flow control

- break case continue default do else for goto if return switch while

- Operators

- sizeof, typeof



# Operator precedence

```
>>  ( )  [ ]  ->  .  
<<  !  ~  ++  --  +  -  *  &  (type)  sizeof  
>>  *  /  %  
>>  +  -  
>>  <<  >>  
>>  <  <=  >  >=  
>>  ==  !=  
>>  &  
>>  ^  
>>  |  
>>  &&  
>>  ||  
<<  ?  :  
<<  =  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=  
>>  ,
```

# Maths and Logic

(things I want done before maths)

>> \* / % (maths)

>> + -

(things between maths & logic)

>> &

>> ^

>> | (logic)

>> &&

>> ||

(things I want after logic)

# Really important things

```
>> ( ) [ ] -> . {language constructs}
<< ! ~ ++ -- + - * & (type) sizeof {unary}
>> * / % {maths}
>> + - }
```

  

```
>> & {logic}
>> ^
>> |
>> &&
>> || }
```

# What goes between maths and logic?

```
>>  ( )  [ ]  ->  .  {language constructs}
<<  !  ~  ++  --  +  -  *  &  (type)  sizeof  {unary}
>>  *  /  %  {maths
>>  +  -  }

>>  <  <=  >  >=  {comparison}
>>  ==  !=  {equality}
>>  &  {logic
>>  ^
>>  |
>>  &&
>>  ||  }
```

# Finally

```
>> ( ) [ ] -> .  
<< ! ~ ++ -- + - * & (type) sizeof  
>> * / %  
>> + -  
>> << >>  
>> < <= > >=  
>> == !=  
>> &  
>> ^  
>> |  
>> &&  
>> ||  
<< ? :  
<< = += -= *= /= %= &= ^= |= <<= >>=  
>> ,
```

# Variables

- Name to an address.
- Type says amount of memory to reserve
- Must be declared before use

# Some reading

- Main books:
  - Brian W. Kernighan, Dennis M. Ritchie *The C Programming Language, Second Edition*, Prentice Hall 1988
  - Randal E. Bryant and David R. O'Hallaron *Computer Systems: A Programmer's Perspective (CS:APP)*, Prentice Hall, 2003
    - *Note: also has a newer edition*

# Real-Time Systems

- Hardware or software which has a time constraint for reactions
- For our purposes, also embedded systems
  - What would be the difference?



# Characteristics

- Specified limit on system response latency
- Event-driven scheduling
- Low-level programming
- Software coupled to special hardware
- Volatile Data
- Multi-tasking implementation
- Unpredictable environment
- Runs continuously
- Life-critical applications

# Example: Anti-lock brakes

- Must prevent locking of wheels while braking
- Inputs: Brake pedal, Wheel rotation
- Actuators: Brakes

# Human brain?

- "The human brain runs a Real-Time Operating System. Conscious thought is a low priority task."
  - Bob Cross on c2 wiki
- Real-time system or not?

# Pathfinder Rover

- Initially successful: July 4, 1997
- Software resets start
  - Serious data losses
  - Problem: bus overloaded with data
  - Low priority data collection locks the bus, medium priority tasks interrupt it
  - High priority data distribution task fails: cannot get bus
  - Scheduler detects pending high-priority task & resets

# Solutions

- Priority inversion: high priority task delayed in a critical section by low priority tasks
- Solution was priority inheritance: low priority tasks entering critical section will inherit the highest priority of waiting tasks
- Solved the Pathfinder reset problem

# More examples

- Microwave, dishwasher, toaster
- Cars: cruise control, drive-by-wire
- Computers: peripheral devices, applications
- Planes: auto-pilot, stability, fly-by-wire

# Terminology

- System: black box with  $n$  inputs and  $m$  outputs
- Response time: time between presentation of a set of inputs and the appearance of the corresponding outputs
- Events: Changes of state which cause changes in flow-of-control of a program
  - Synchronous: events occur at predictable times
  - Asynchronous: events interrupt flow-of-control

# State vs Event based

- State based:
  - System constantly reads system inputs and reacts to their combination
- Event based
  - System is in standby and events “wake” it to make it work



# Deterministic RTS

- A deterministic RTS: you can determine a unique set of outputs and next state from a given set of possible states and inputs.

# Real time Correctness

- Correctness depends on result and the time of delivery.
- Soft – missing some deadlines not a problem
- Firm – missing deadline: result worthless, but not a problem
- Hard – missing a deadline makes result worthless and is a problem

# Misconceptions

- “Really fast” is real-time.
  - Might not be predictable enough
- Interactive is real-time.
  - Again: interactive optimized for “average” case.
- Real-time = “Bug free”:
  - Often the case, but bug free is wider concept

# Static Predictability

- RT system: satisfying time constraints
  - Assumptions about workload and sufficient resources
  - Certified at design time, that all constraints will be met
- For static systems, 100% guarantees can be given at design time
  - Requires immutable workload and system resources
  - System must be re-certified on any change

# Dynamic Predictability

- Dynamic systems: not statically defined
  - Systems configurations might change
  - Workload might change
- Dynamic predictability
  - Under appropriate assumptions (sufficient resources)
  - Tasks will satisfy time constraints

# Latency minimization

- Latency is the time between an event and the system's reaction to it.
- We want to minimize latencies
  - For different applications, different latencies are required.
  - 10 ms might be barely enough (probably a dedicated system)
  - 500 ms might be enough (we could use an external kernel)

# Multiple Requirements

- Real-time
- Power constraints
- Size constraints
- Cost limits
- Security requirements
- Fault tolerance
  
- *Often conflicting*

# New Environments

- Ubiquitous Computing
  - Computers become invisible, so embedded and natural that we use them without thinking of using them.
- Autonomous Computing
  - Self-configurable
  - Self-adapting
  - Optimizing
  - Self-healing



# End of buzzwords

- Lab: Linux environment and command-line and hello world
- No points for the first one, just to get to know if the environment still works.