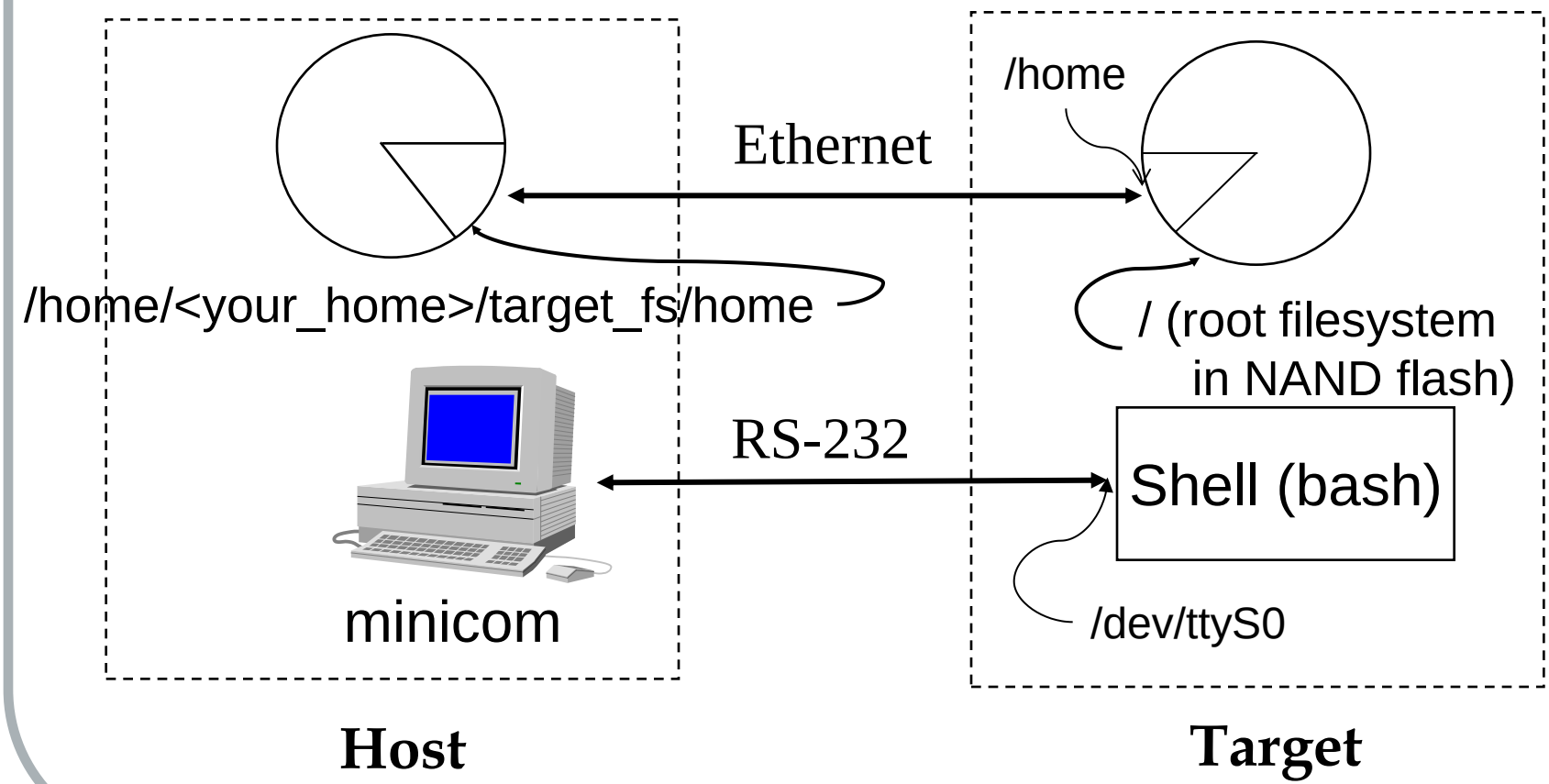
Session #4: Building and Debugging Target Applications

Cross-platform development

What's Going On Here



© Douglas Abbott 2009 to 2015 | UCSD Extension Online Learning Course: Embedded Linux

Accessing hardware from Linux

- Peripherals share address space with memory
- Must “map” peripheral space into user space process
  - open (“/dev/mem”);
  - mmap ();

Data Acquisition Example

- ADC channel 0 has pot connected to it
  - Pretend that it represents "temperature"
- measure <interval>
  - Reads ADC channel 0 every *interval* seconds (default = 2)
  - Prints reading on stdout
- "Driver" functions encapsulated in separate file

Create a “Makefile” Project

- File > New > C Project
  - Project name: measure
  - Uncheck Use default location
  - Browse to ~/target_fs/home/src/measure
  - Project type: **Makefile** project
- Finish

Source code for measure

- measure.c
  - main() function
  - Reads pushbuttons -> LEDs
  - Calls driver functions in …
- trgdrive.c
  - Implements simple device driver APIs
- Makefile
  - Has multiple targets
  - Cross-compiler = arm-linux-gcc

Minor Problem

- Eclipse can't find header files
  - Only a problem with Makefile projects
- Right click measure in Project Explorer
  - Properties
  - Expand C/C++ General
  - In Includes tab, click Add…
  - Enter /usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/usr/include
  - Check Add to all languages and Add to all configurations

Minor Problem II

- Click Add… again
  - Enter ../../include
  - Check Add to all languages and Add to all configurations
- Click Apply
  - Say yes to message about rebuilding index
- Click OK

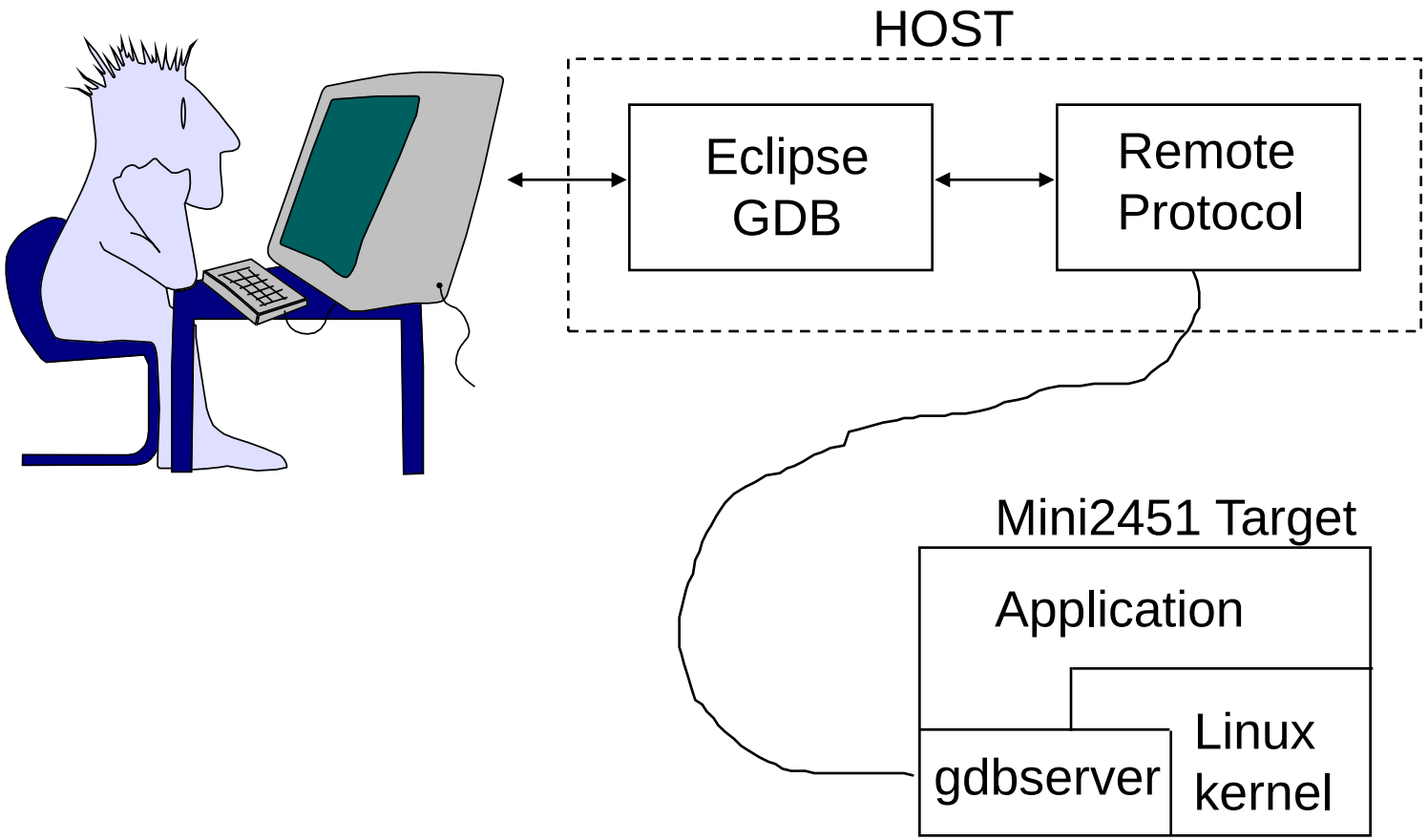
Make targets

- Select Make Targets view
- Right-click measure project name
  - Add Make Target
    - Target Name: measure
    - Make Target: measure
- Right-click measure target
  - Build Make Target
- Uses ARM cross-compiler to build measure for target board

Run measure on target

- Start minicom on the workstation
- Boot the target board
- cd /home/src/measure
- ./measure 1
- Turn the pot

Remote Debugging with Eclipse



gdbserver on Target Board

- GDB stubs
  - Links with target program
  - RS-232 only
- GDB server
  - Separate program that runs target program
  - Works over Ethernet

- gdbserver :10000 measure

Setting up Remote Debug

- Right-click measure in Project Explorer
  - Debug As > Debug Configurations
  - New Default name is measure Default
- Main tab
  - Select Standard Create Process Launcher
- Debugger tab
  - Debugger: GDB server
  - GDB Debugger: arm-linux-gdb
  - Connection: TCP
    - IP address: 192.168.1.50
  - Check Stop at main()
- Apply, Debug

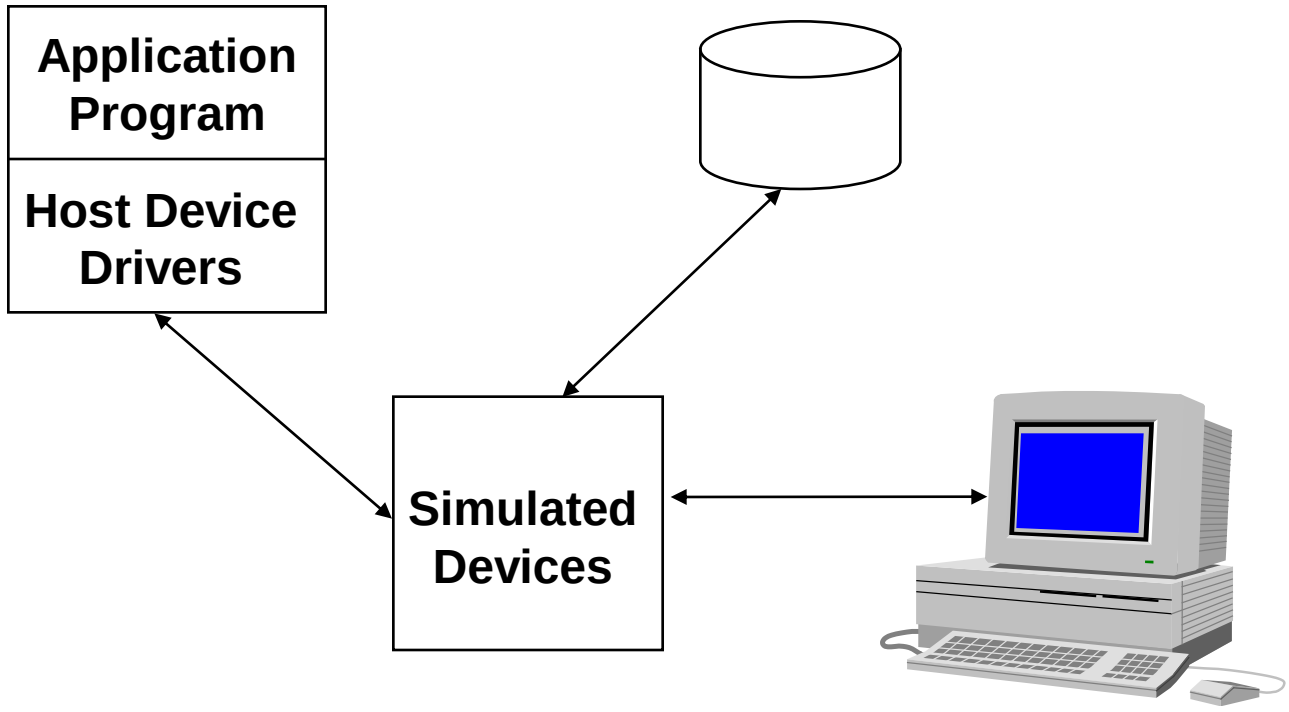
The Development Host as Debug Environment

- Available early in development cycle
- Convenient user interaction (GUI)
- Powerful debugging tools
- File system to document test results

Embedded Software Content

Hardware-dependent startup code	1%
Hardware-dependent interrupt code	1%
Other hardware-dependent code	1%
Hardware-independent interrupt code	1%
Everything else	96%

Testing with a Simulated Device Driver



“Thermostat” Enhancement

- Use LEDs to represent:
  - Cooler - Turns on if temperature rises above setpoint (= 50)
  - Alarm - Flashes if temperature exceeds limit (= 53)
- “Deadband” for hysteresis (= 1)
- State Machine
  - States = OK, High, Limit

Thermostat State Machine

Current state Normal
- Temperature above setpoint + deadband
  - Turn on COOLER
  - Next state = High
- Temperature above limit
  - Turn on ALARM
  - Next state = Limit

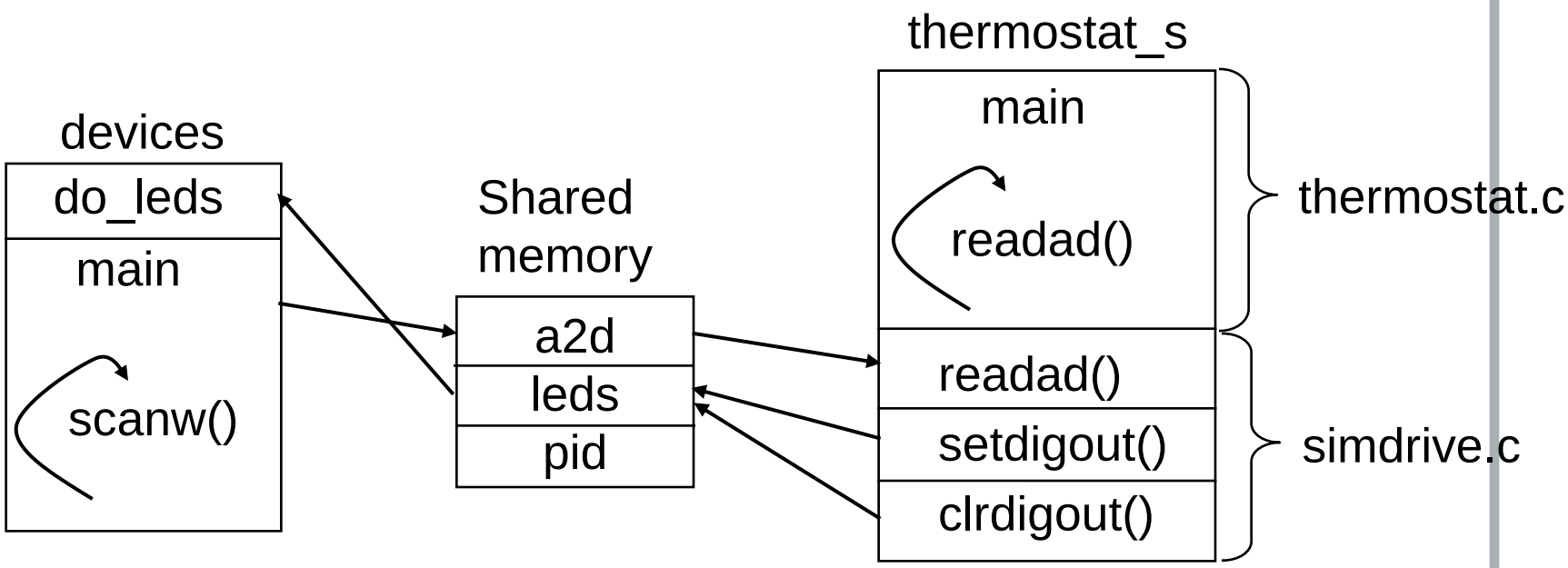
Current state High
- Temperature below setpoint - deadband
  - Turn off COOLER
  - Next state = Normal
- Temperature above limit
  - Turn on ALARM
  - Next state = Limit

Current state Limit
- Temperature below limit
  - Turn off ALARM
  - Next state = High
- Temperature below setpoint - deadband
  - Turn off COOLER
  - Next state = Normal

Procedure

- Copy measure.c to thermostat.c
- Implement thermostat state machine
  - Divide A/D input by 10
- Alarm light must blink at one second rate independent of the sample period

Thermostat Simulation



Additional make targets

- Target Name: devices
  - Make Target: devices
- Target Name: ARMthermo
  - Make Target: all
  - Uncheck Use default
  - Build command: make TARGET=1

Debug Configurations

- Host launch configuration
  - Rename record_sort as host
  - Change Project to measure
  - Change Application to thermostat_s
- Target launch configuration
  - Rename measure to target
  - Leave Project and Application as is for now

Debug thermostat_s

- In a shell window on the workstation
  - cd target_fs/home/src/measure
  - ./devices
- In Eclipse
  - Click Debug in the host launch configuration
- Use breakpoint condition feature to watch the state machine change states

Debug thermostat_t on Target

- Delete thermostat.o
- Build ARMthermo target
- Point target launch configuration at thermostat_t application
- On target
  - gdbserver :10000 thermostat_t 1
- On workstation
  - Click Debug in launch configuration

Make setpoint and limit remotely settable

- Create independent thread of execution to monitor console input
- Possible Implementations:
  - fork() in thermostat.c
  - Start a second process
  - Threads

Creating a Linux Process--the fork() function

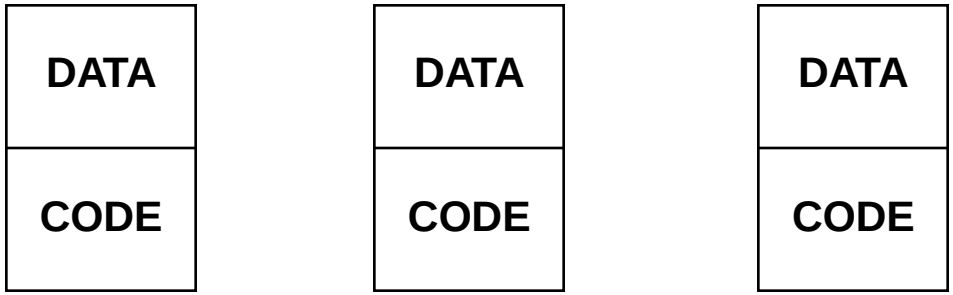
- Creates a complete *copy* of the process
  - Code, Data, File descriptors, etc
  - Uses *copy-on-write* so only page tables and task structure copied initially
- Both processes continue from the return from fork()
  - Returns 0 to *child* process
  - Returns PID of child to *parent* process

Fork examples

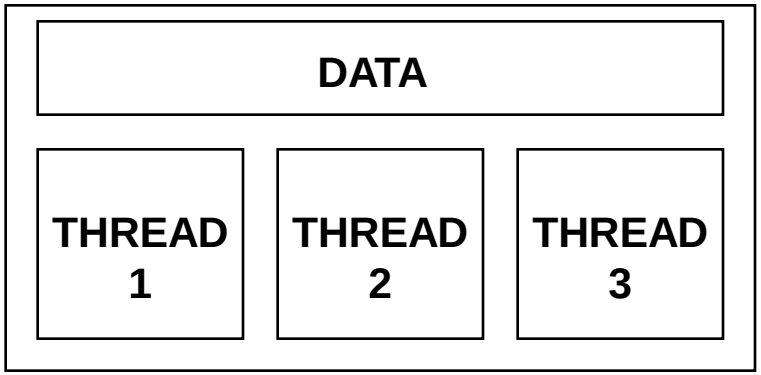
- Download pseudo_code.pdf from Week 4 folder

“Processes” vs “Threads”

UNIX Process Model



Multi-threaded Process



Thread API

```
int pthread_create (pthread_t *thread, pthread_attr_t *attr, void *(* start_ routine)
        (void *), void *arg);
void pthread_exit (void *retval);
int pthread_join (pthread_t thread, void **thread_return);
pthread_t pthread_self (void);
int sched_yield (void);
```

Thread Termination

- Thread function just returns
- Thread function calls pthread_exit
- Thread is *cancelled* by another thread
- detachstate
  - PTHREAD_CREATE_JOINABLE - Can be "joined" by another thread when it terminates
  - PTHREAD_CREATE_DETACHED - Can't be joined. Resources reclaimed immediately.

Thread cancellation

pthread_cancel (pthread_t thread);
pthread_cleanup_push (void (*routine) (void *), void *arg);
pthread_cleanup_pop (int execute);

- Can only be called from the function that called cleanup_push()

- Deferred cancellation – Cancellation points
  - Any system call that can block is a cancellation point
  - pthread_test_cancel (void);

Attribute Objects

- All pthread objects can have additional, separate *attribute* objects
- Extended argument list for object creation
- Attribute is second argument to create call
  - if NULL, use defaults
- Objects are "opaque"
  - access with specific attribute functions

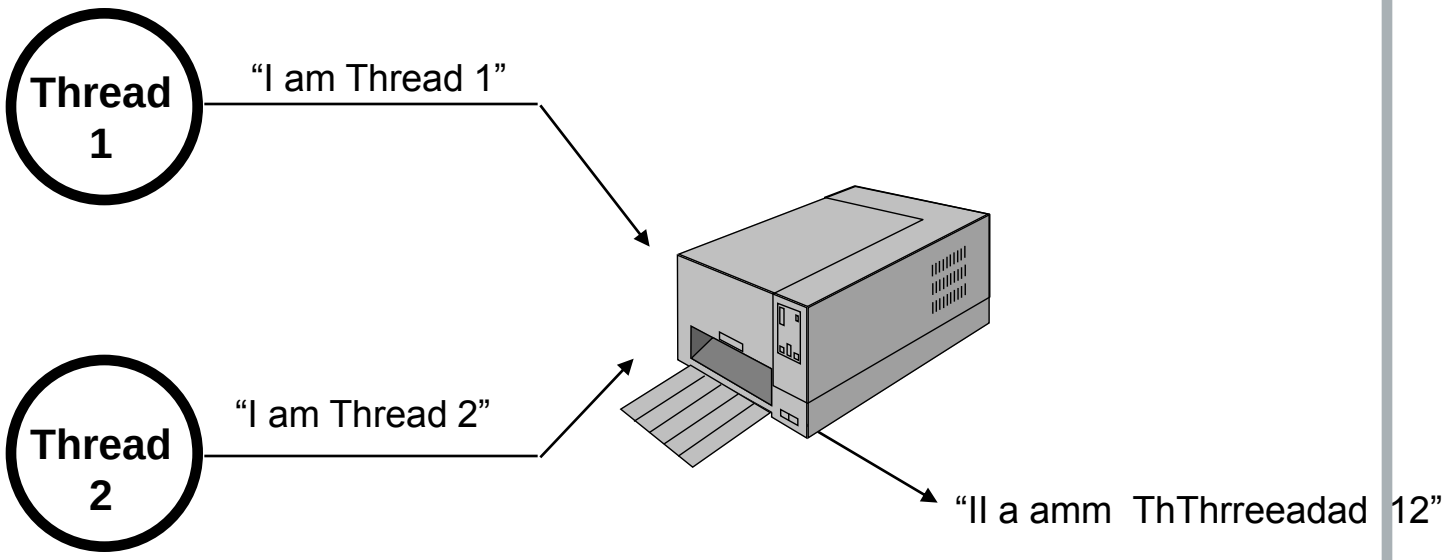
Thread Attributes

```
int pthread_attr_init (pthread_attr_t *attr);
int pthread_attr_destroy (pthread_attr_t *attr);
int pthread_attr_getdetachstate (pthread_attr_t *attr, int *detachstate);
int pthread_attr_setdetachstate  (pthread_attr_t *attr, int detachstate);

#ifdef _POSIX_THREAD_ATTR_STACKSIZE
int pthread_attr_getstacksize (pthread_attr_t *attr, size_t *stacksize);
int pthread_attr_setstacksize (pthread_attr_t *attr, size_t stacksize);
#endif

#ifdef _POSIX_THREAD_ATTR_STACKADDR
int pthread_attr_getstackaddr (pthread_attr_t *attr, void **stackaddr);
int pthread_attr_setstackaddr (pthread_attr_t *attr, void *stackaddr);
#endif
```

Sharing Resources



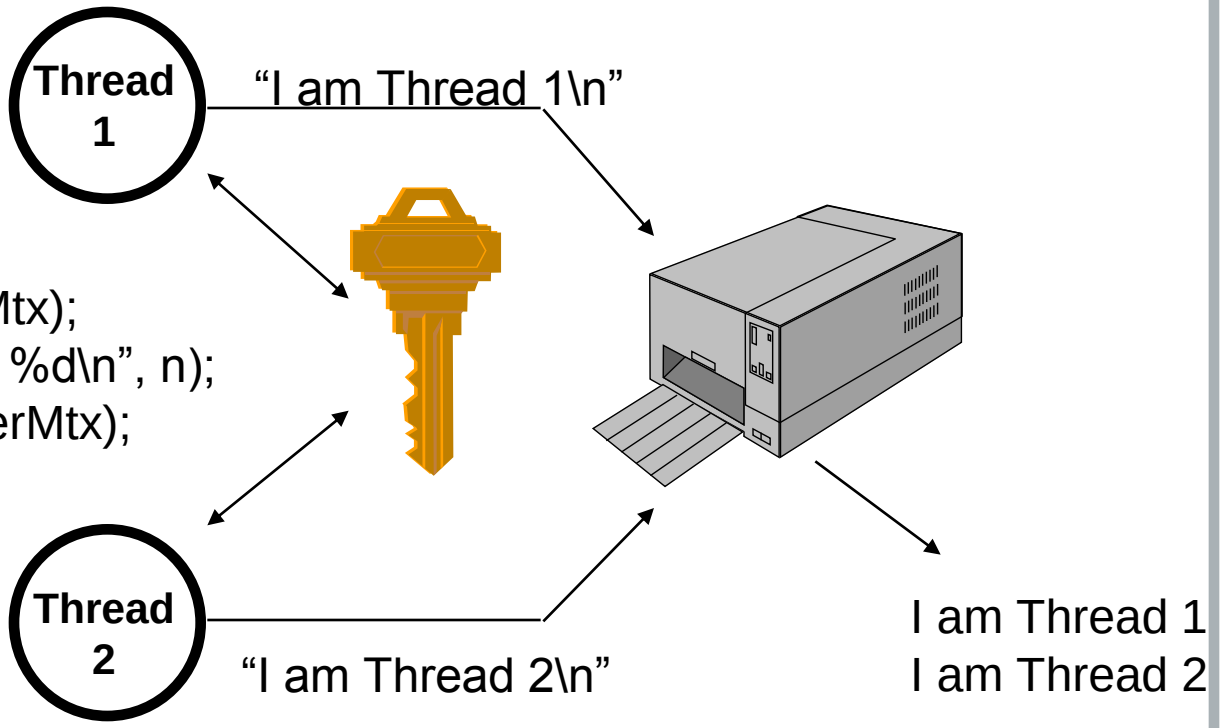
Code in Thread n

```
printf ("I am Thread %d\n", n);
```

Sharing Resources with a Mutex

Code in Thread n

```
lockMutex (PrinterMtx);
printf ("I am Thread %d\n", n);
unlockMutex (PrinterMtx);
```



Mutex API

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t
        *mutex_attr);
int pthread_mutex_destroy (pthread_mutex_t *mutex);

int pthread_mutex_lock (pthread_mutex_t *mutex);
int pthread_mutex_unlock (pthread_mutex_t *mutex);
int pthread_mutex_trylock (pthread_mutex_t *mutex);
```

Remote Command Syntax

- s <nn> -- change setpoint
- l <nn> -- change limit
- d <nn> -- change deadband

Example:

s 65 setpoint = 65

- Open monitor.c in the posix/ directory

In Eclipse

- Create a new makefile project, posix
  - Browse to Posix directory
- Add a make target for the posix project
  - Name: ARMthermo
  - Target: all
  - Uncheck Use default
  - Command: make TARGET=1
- Point the host and target debug configurations at the posix project

Changes to thermostat.c

- Copy thermostat.c to ../posix
- #include pthread.h and thermostat.h
  - Make sure variable names agree
- Call createThread() and test the return value at the top of main()
- Lock paramMutex just before the switch statement on the state variable and unlock it at the end of the switch statement
- Call terminateThread() at the end of main()

Debugging multi-threaded programs

- Set breakpoint before createThread()
  - Debug view shows one thread
- Step over createThread()
  - Second thread (monitor) shows up
  - Suspended inside clone() function
- Set breakpoint after sleep()
  - Monitor inside fgets()
- Set breakpoint after fgets() call

Review

- Accessing hardware from Linux
- Eclipse Makefile projects
  - Multiple make targets
- High level simulation
- Multiple threads of execution
  - Forking processes
  - Posix threads