



SE350 RTX LAB 3

P3-A I/O Services

P3-B Stress Testing

Irene Yiqing Huang


Electrical and Computer Engineering Department

Reading the Lab Manual


Section	Topics	How to Read
1.2	Summary of RTX Requirements	Review
3.1.2	System Console I/O Processes	Study
3.2.2	The UART I-Process	Study
3.3.2	Wall Clock Display Process	Study
3.3.3	Set Priority Command Process	Study
3.3.4	Stress Test Processes	Study
5.2	Third-party testing framework	Review
6.4 – 6.5	FAQ – System Processes and Interrupts	Skim
9.4	C and Assembly Programming	Skim
9.6	UART Programming	Skim

P3 Highlights

- Console I/O Services
 - UART I-Process
 - Keyboard Command Decoder (KCD) Process
 - Cathode Ray Tube (CRT) Display Process
- Stress Testing
 - The 24-hour wall clock display process
 - The set process priority command process
 - The A, B and C Processes
- Six User Test processes test cases (optional)



P3-A



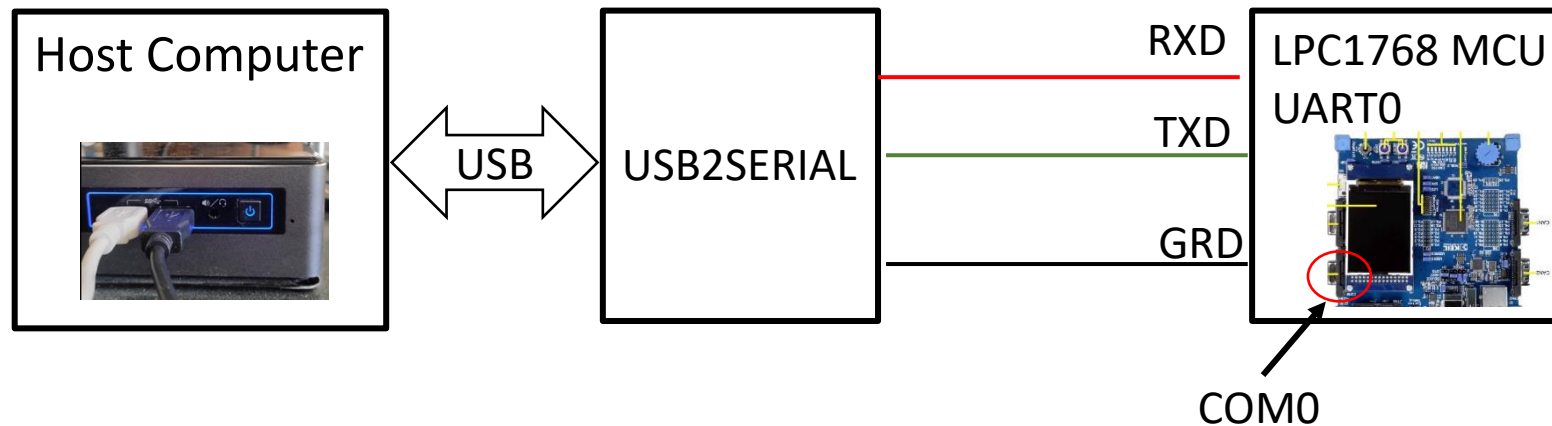
P3-B

RTX Initialization

- What operations need to be carried out at start-up?
- Initialize all hardware, incl.
 - Board system Initialization
 - Interrupts (hardware and software: vector table & traps)
 - Timer(s) and Serial port(s)
- Create all kernel data structures
 - Memory management kernel data structure
 - Process-control kernel data structure: PCB, kernel stacks
- Create PCBs of all processes
 - allocate stacks
 - privilege level setting using CONTROL register
 - Exception stack frame creation for new processes

Console I/O Services

- Input Device
 - The keyboard attached to the host machine
- Output Device
 - The Host machine Monitor – CRT/LCD



UART Registers

- Status Registers
 - Line Status Register - LSR
 - LSR_RDR (Receiver Data Ready)
 - LSR_THRE (THR Empty)
- Control Registers
 - Interrupt Enable Register - IER
 - IER_RBR bit – RX Interrupt
 - IER_THRE bit – TX interrupt
 - IER_RLS bit
- Data Registers
 - Input: Read RBR
 - Output: Write THR when it is empty
- Section 9.6 of lab manual

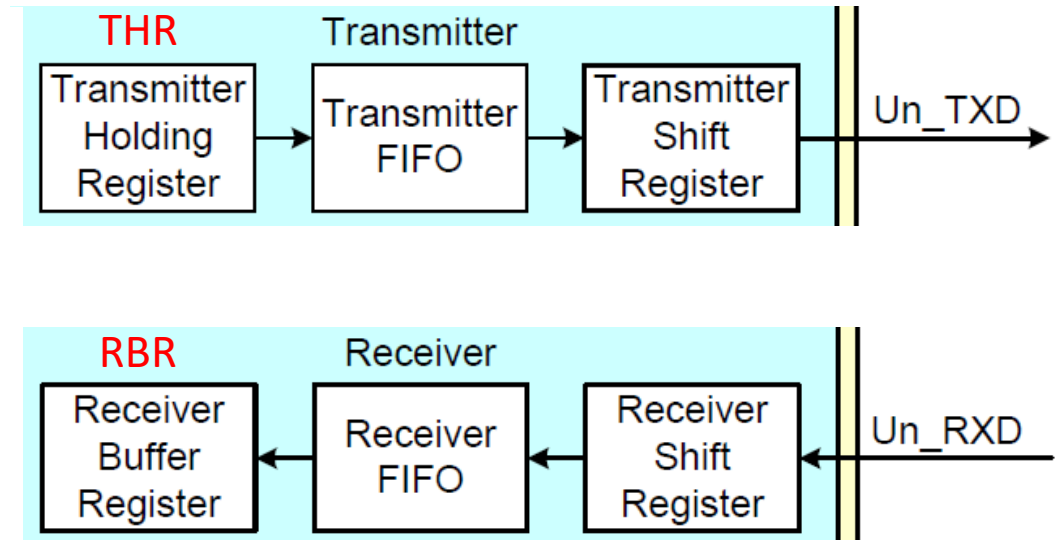


Image courtesy of [2]

The TX Interrupt and CRT Process

- TX interrupt is disabled when there are no data to transfer
 - Who disables the TX interrupt?
- TX interrupt needs to be enabled when there are data to transfer
 - Who enables the TX interrupt?
- CRT Display process kick-starts the TX interrupt

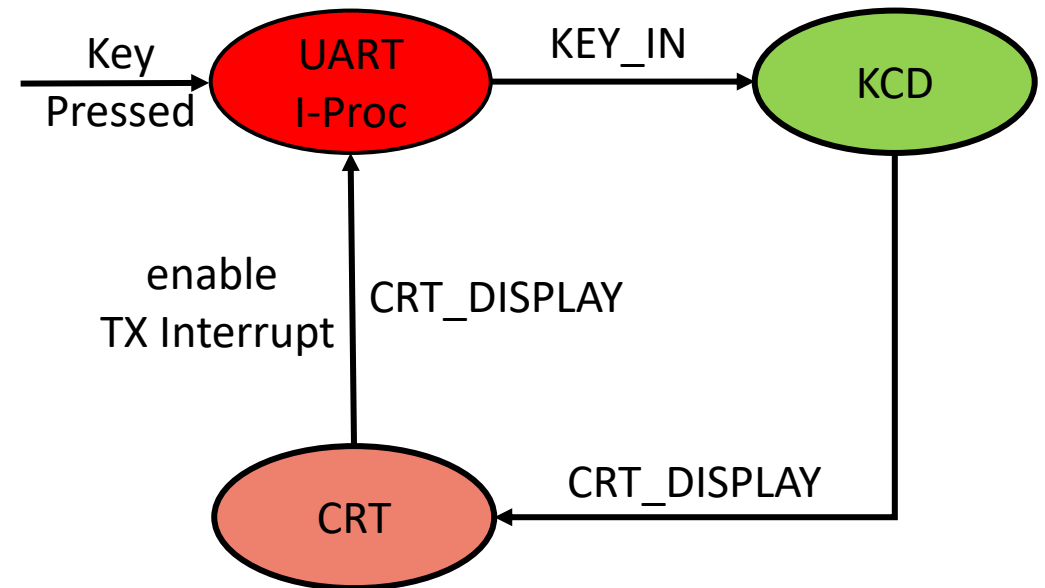
Commands and KCD Process

- Format
 - Start with % followed by a single char and then argument and the carriage return.
 - Examples:%WS 20:00:00, %C 10 3
- How does the Kernel know where the command is ?
 - General purpose OS: the path environment variable
 - Our Kernel: Keyboard Command Decoder (KCD) Process
 - Maps commands to processes

Console I/O: UART I-Process

- RX interrupt
 - Reads input to the UART
 - Processes hot-key
 - Output to the debug terminal
 - Sends **KEY_IN** messages to KCD
- TX interrupt
 - Receives **CRT_DISPLAY** messages from CRT
 - Writes to the UART
 - Turn off TX interrupts when done

- UART0: interrupt driven, the RTX console
- UART1: polling, debug terminal

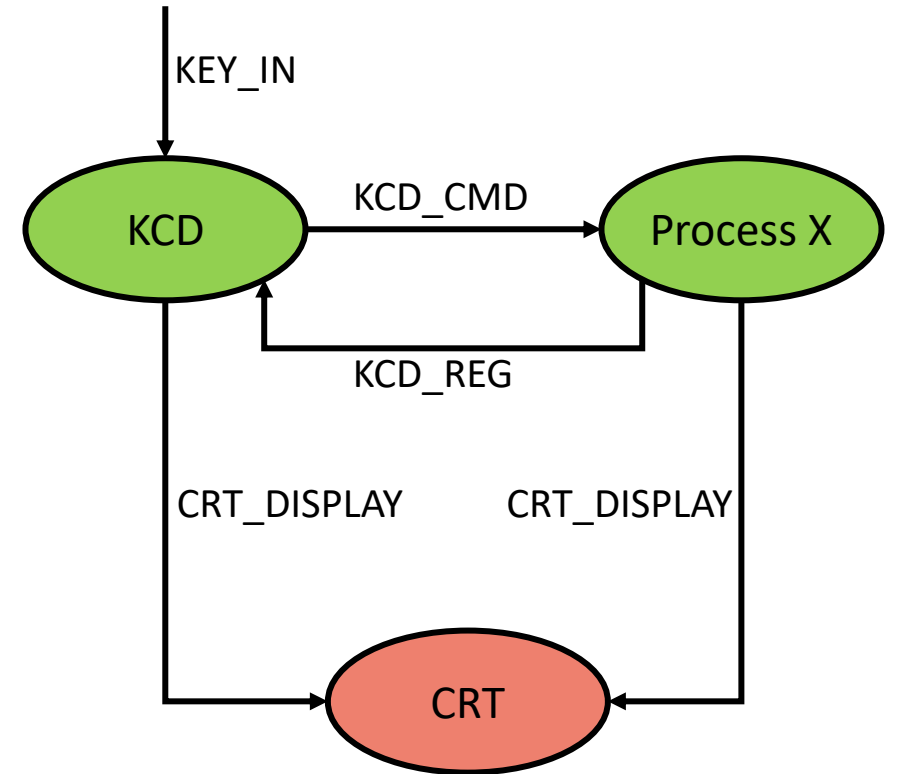


Hot Keys

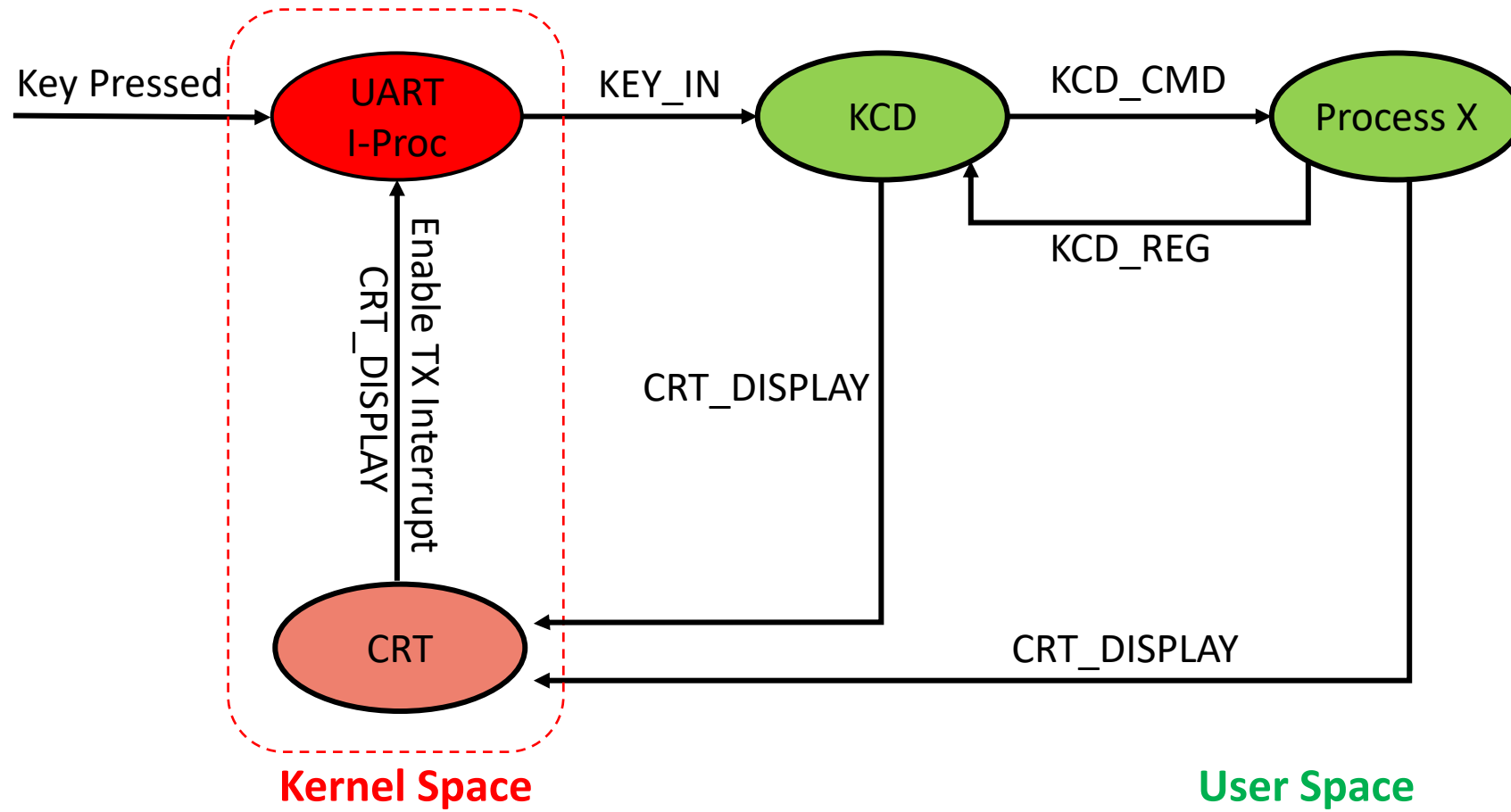
- Output to the debug terminal by polling
- Output format is your own design
 - Make it easier to read by others
- Required hot keys
 - ! Ready processes and their priorities
 - @ blocked on memory processes and their priorities
 - # blocked on receive processes and their priorities
- **Bonus hot keys**
 - \$ all processes' priorities and states (1 pts)
 - & number of memory blocks available (1 pt)
 - ^ 10 most recent IPC messages (3 pts)

Console I/O: System Processes

- CRT process
 - receives **CRT_DISPLAY** messages
 - sends messages to the UART I-Process
 - Kick-starts the TX interrupt
- KCD process:
 - Registers commands at run time
 - Receives **KCD_REG** messages from any process
 - Maintains records of registered commands
 - Max command length is 64 bytes
 - Decodes commands
 - Receives **KEY_IN** messages and build **KCD_CMD** upon Enter key
 - Sends **CRT_DISPLAY** to CRT to echo keyboard input and error messages
 - Sends **KCD_CMD** messages to the correct process



System I/O Communications



The 24-hour Wall Clock Display Process

- It is an unprivileged user process
- Which API should it call to get the service?
- How to display the clock?
 - must be displayed on the interrupt driven RTX console terminal
 - **Bonus:** ANSI escape sequences to display the clock at the upper right corner of the terminal (5 pts)
- Required Commands
 - %WR, %WT
 - %WS hh:mm:ss (should handle invalid input arg.)

Set Priority Command Process

- Syntax
 - %C <pid> <prio>
 - Example: %C 10 2
- Affects the OS internal states
 - May change the ready queue status
 - May change the blocked-on-resource queue status
- Error Handling
 - If pid or prio is illegal, print an error message of “Invalid command input” to the console

Process A

```
Register %Z command with the KCD
Wait till %Z command is received
num = 0
loop forever
    get a message envelope
    set message_type field of to count_report
    set message_text field of to num
    send the message to process B
    num = num + 1
    release_processor()
endloop
```


Note that Process A does not deallocate any message envelope inside the infinite loop.

Process B

```
loop forever  
    receive a message  
    send the message to process C  
endloop
```

Note that Process B does not deallocate any message envelope inside the infinite loop.

Process C

```
create a local message queue
loop forever
  if (local message queue is empty) then
    p <- receive a message
  else
    p <- dequeue the first message from the
      local message queue
  endif
  if msg_type of p == count_report then
    if message_text % 20 == 0 then
      send "Process C" to CRT with msg envelope p
      hibernate for 10 sec
    endif
  endif
  deallocate message envelope p 
  release_processor()
endloop
```

Process C: hibernate

```
q <- request_memory_block()
use q to delayed_send to itself with 10 sec delay and
msg_type=wakeup10
loop forever
  /* block and let other process execute */
  p <- receive a message
  if (msg_type of p == wakeup10) then
    exit this loop
  else
    put message (p) on the local message queue
    for later processing
  endif
endloop
```

Note that Process C keeps receiving messages during hibernation

P3 Processes Summary

PID	PID Macro	Process	Mode	Privilege Level
15	PID_UART_IPROC	Timer I Process	Handler	Privileged
13	PID_CRT	CRT	Thread	Privileged
12	PID_KCD	KCD	Thread	Unprivileged
11	PID_CLOCK	Wall Clock	Thread	Unprivileged
10	PID_SET_PRIO	Set Proc. Priority Command	Thread	Unprivileged
7-9	PID_[ABC]	A, B, C Processes	Thread	Unprivileged

How to Stress Tests?

- You need the Wall clock process, KCD and CRT
- Set A, B, C to different priority combination
- Under some priority setting, the system might not function normally
 - You are not required to do extra work
 - You need to explain why your system's behavior is expected given the API requirements in the project.
 - Tips: Use hotkeys

Starter Code and Submission

- Use your existing P2
- Create a directory called P3
 - Copy existing P2/Context_Switching Project to P3

```
├── P1
│   └── Context_Switching
├── P2
│   └── Context_Switching
├── P3
│   └── Context_Switching
└── README.md
```

- Tag your commit with “p3-submit”

References

1. Dasiewicz, Paul, A non-preemptive RTX Design Documentation
2. LPC17xx User's Manual
3. ARM Compilation Tools Version 5.0 Developer Guide
4. Software Interface Standard for Arm Cortex-based Microcontrollers, CMSIS Version 5.7.0



Thank you!



Electrical and Computer Engineering Department