CS 1550 – Chapter 5
I/O

Jonathan Misurda
jmisurda@cs.pitt.edu

Block Devices

A device that stores data in fixed-sized blocks, each uniquely addressed, and can be randomly accessed

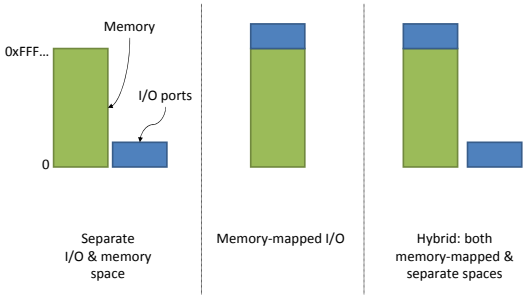
Character Devices

Device that delivers or accepts a stream of characters

Device Controllers

The electronic component of an I/O unit, in contrast with the physical component.

Memory-Mapped I/O



Dynamic Frequency on XScale

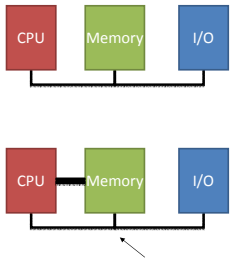
Core Run Freq (MHz)	CLKCFG[T]	Core Turbo Freq (MHz)	CLKCFG[T]	CLKCFG[HT]	CCCR[L]	CCCR[2N]	System Bus (MHz)	CLKCFG[B]	CLK_MEM (MHz)	CCCR[A]
104	0	208	1	0	8	4	104	1	104	1
208	0	208	1	0	16	2	208	1	104	0
208	0	208	1	0	16	2	208	1	208	1
104	0	312	1	0	8	6	104	1	104	1
208	0	312	1	0	16	3	208	1	104	0
208	0	312	1	0	16	3	208	1	208	1
208	0	416	1	0	16	4	208	1	104	0
208	0	416	1	0	16	4	208	1	208	1
208	0	520	1	0	16	5	208	1	104	0
208	0	520	1	0	16	5	208	1	208	1
208	0	624	1	0	16	6	208	1	104	0
208	0	624	1	0	16	6	208	1	208	1

Setting CPU Freq. in WinCE

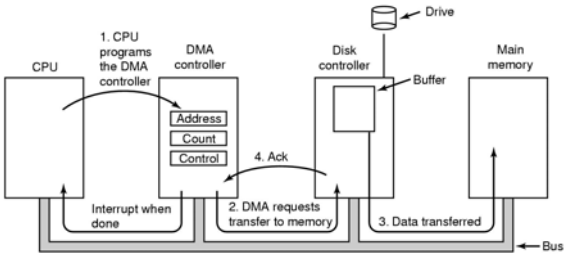
```
// Allocate some space for the virtual reference to CCCR
LPVOID virtCCCR = VirtualAlloc(0, sizeof(DWORD), MEM_RESERVE, PAGE_NOACCESS);

//0x41300000 is the memory-mapped location of the CCCR register
LPVOID CCCR = (LPVOID)(0x41300000 / 256); // shift by 8 bits for ability to address 2^40 bytes

// Map writing the virtual pointer to the physical address of the CCCR register
VirtualCopy((LPVOID)virtCCCR, CCCR, sizeof(DWORD), PAGE_READWRITE | PAGE_NOCACHE | PAGE_PHYSICAL);

// Set the CCCR register with the new speed
*(int *)virtCCCR = new_speed;

// Call the assembly function to actually perform the switch
doSwitch(0x02 | 0x01); //0x02 means turbo mode, 0x01 means the clock is being switched

// Clean up memory by freeing the virtual register.
VirtualFree(virtCCCR, 0, MEM_RELEASE);
virtCCCR = NULL;
```

```
; Coprocessor 14, register C6 (CLKCFG) initiates the changes programmed in CCCR
; when CLKCFG is written.

doSwitch
      MOV r3, r0              ; Move r0, the argument to doSwitch, into register r3
      MCR p14, 0, r3, c6, c0, 0  ; Copy the contents of r3 into register c6 on coprocessor 14.
      MOV pc, lr              ; return execution to where it last left off
```

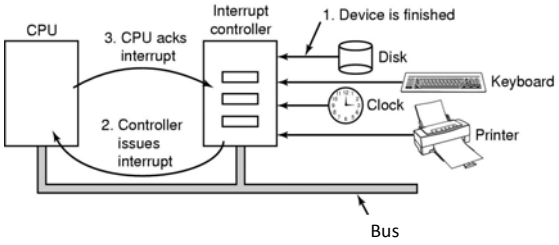
Bus Communication



DMA



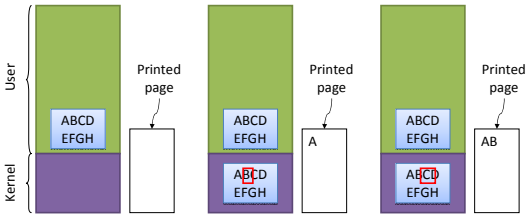
Interrupts



I/O Software Goals

- Device independence
- Uniform naming
- Error handling
- Synchronous vs. asynchronous transfers
- Buffering
- Sharable vs. dedicated devices

Programmed I/O



Polling

```
copy_from_user (buffer, p, count);  // copy into kernel buffer

for (j = 0; j < count; j++) {       // loop for each char
  while (*printer_status_reg != READY)
    ;                               // wait for printer to be ready
  *printer_data_reg = p[j];         // output a single character
}

return_to_user();
```

Interrupt-Driven I/O

System Call

```
copy_from_user (buffer, p, count);
j = 0;
enable_interrupts();
while (*printer_status_reg != READY)
  ;
*printer_data_reg = p[0];
scheduler(); // and block user
```

Interrupt Handler

```
if (count == 0) {
  unblock_user();
} else {
  *printer_data_reg = p[j];
  count--;
  j++;
}
acknowledge_interrupt();
return_from_interrupt();
```

DMA

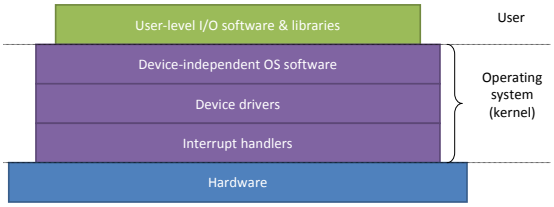
System Call

```
copy_from_user (buffer, p, count);
set_up_DMA_controller();
scheduler(); // and block user
```

Interrupt Handler

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

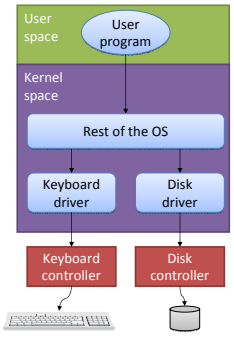
I/O Software Layers



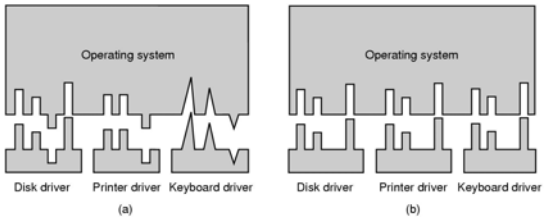
Interrupt Handling

1. Set up stack for interrupt service procedure
2. Ack interrupt controller, re-enable interrupts
3. Copy registers from where saved
4. Run service procedure
5. (optional) Pick a new process to run next
6. Set up MMU context for process to run next
7. Load new process' registers
8. Start running the new process

Device Drivers



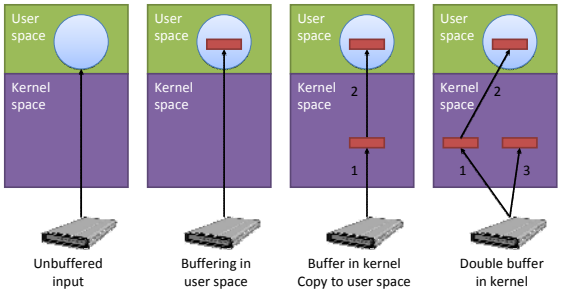
Driver Interfacing



Non-standard driver interfaces

Standard driver interfaces

Buffering



I/O Request