Interrupt handling and context switching

- these two topics are separate and we will examine them in turn

Interrupts

- ellipse at 3.740,7.547 wid 1.181 ht 0.394 ellipse at 3.740,7.547 wid 1.969 ht 1.181 ellipse at 3.740,7.547 wid 3.150 ht 1.969 ellipse at 3.740,7.547 wid 4.724 ht 3.150 line from 3.740,9.122 to 3.740,8.531 line from 1.378,7.547 to 2.165,7.547 line from 5.315,7.547 to 6.102,7.547 line from 3.740,6.563 to 3.740,5.972 line from 2.756,6.760 to 2.257,6.340 "HW" at 3.648,7.528 ljust "Shell" at 3.543,8.329 ljust "Device drivers" at 3.228,7.869 ljust "daemons" at 4.331,8.565 ljust "applications" at 2.165,8.565 ljust "utilities" at 1.772,6.990 ljust "commands" at 2.953,6.203 ljust "compiler" at 4.724,6.596 ljust "Kernel" at 3.386,7.148 ljust

- the user programs and hardware communicates with the kernel through interrupts

Four different kinds of interrupts

- device interrupt, such as a hardware timer, for example the 8253 `counter0` reaching `0` on an IBM-PC
- user code issuing a software interrupt, often called a **system call**
- an illegal instruction (divide by zero, or an opcode which the processor does not recognise)
- or a memory management fault interrupt (occurs when code attempts to read from non existent memory)

First level interrupt handler

- the kernel must detect which kind of interrupt has occurred and call the appropriate routine
  - this code is often termed the **first level interrupt handler**

- the pseudo code for the FLIH follows:

First level interrupt handler

```
save program registers and disable interrupts
k = get_interrupt_kind ();
if (k == source 1) service_source1 ();
else if (k == source 2) service_source2 ();
else if (k == source 3) service_source3 ();
else if (k == source 4) service_source4 ();
else if (k == source 5) service_source5 ();
  etc
restore program registers and enable interrupts
return
```

- you may find the hardware on the microprocessor performs the save and restore program registers and disabling/enabling interrupts
  - possibly by one instruction

First level interrupt handler

- you might also find the hardware enables you to determine the source of the interrupt easily
  - most microprocessors have an interrupt vector table
    - typically one vector per source is implemented

- equally, however the code can be ugly as it depends upon the hardware specifications

Example of FLIH in GNU LuK

- GNU LuK (Lean uKernel) is a very small microkernel which allows premptive processes, interrupt driven devices and semaphores

Example of FLIH in GNU LuK

```
IsrTemplate[ 0] := 0FCH ;    (* cld (disable interrupts) *)
IsrTemplate[ 1] := 050H ;    (* push eax *)
IsrTemplate[ 2] := 051H ;    (* push ecx *)
IsrTemplate[ 3] := 052H ;    (* push edx *)
IsrTemplate[ 4] := 01EH ;    (* push ds *)
IsrTemplate[ 5] := 006H ;    (* push es *)
IsrTemplate[ 6] := 00FH ;    (* push fs *)
IsrTemplate[ 7] := 0A0H ;
IsrTemplate[ 8] := 0B8H ;    (* movl 0x00000010, %eax *)
IsrTemplate[ 9] := 010H ;
IsrTemplate[10] := 000H ;
IsrTemplate[11] := 000H ;
IsrTemplate[12] := 000H ;
IsrTemplate[13] := 08EH ;    (* mov  ax, ds *)
IsrTemplate[14] := 0D8H ;
IsrTemplate[15] := 08EH ;    (* mov  ax, es *)
IsrTemplate[16] := 0C0H ;
IsrTemplate[17] := 08EH ;    (* mov  ax, fs *)
IsrTemplate[18] := 0E0H ;
```

Example of FLIH in GNU LuK

```
    IsrTemplate[19] := 068H ;    (* push  interruptnumber *)
    IsrTemplate[20] := 000H ;    (* vector number to be overwritten.  *)
    IsrTemplate[21] := 000H ;    (* this is the single parameter.  *)
    IsrTemplate[22] := 000H ;    (* to function.  *)
    IsrTemplate[23] := 000H ;
    IsrTemplate[24] := 0B8H ;    (* movl function, %eax *)
    IsrTemplate[25] := 000H ;    (* function address to be overwritten *)
    IsrTemplate[26] := 000H ;
    IsrTemplate[27] := 000H ;
    IsrTemplate[28] := 000H ;
```

Example of FLIH in GNU LuK

```
IsrTemplate[29] := 0FFH ;    (* call  %eax *)
IsrTemplate[30] := 0D0H ;
IsrTemplate[31] := 058H ;    (* pop  %eax   // remove parameter *)
IsrTemplate[32] := 00FH ;    (* pop  %fs *)
IsrTemplate[33] := 0A1H ;
IsrTemplate[34] := 007H ;    (* pop  %es *)
IsrTemplate[35] := 01FH ;    (* pop  %ds *)
IsrTemplate[36] := 05AH ;    (* pop  %dx *)
IsrTemplate[37] := 059H ;    (* pop  %cx *)
IsrTemplate[38] := 058H ;    (* pop  %ax *)
IsrTemplate[39] := 0CFH ;    (* iret *)
```

Example of FLIH in GNU LuK

- GNU LuK uses a routine `ClaimIsr` which will copy the `IsrTemplate` into the correct interrupt vector and then overwrite the vector number and function address in the template

Context switching

- the scheduler runs inside the kernel and it decides which process to run at any time
  - processes might be blocked waiting on a semaphore or waiting for a device to respond
  - a process might need to be preemptively interrupted by the scheduler if it were implementing a round robin algorithm

- the minimal primitives to manage context switching in a microkernel or operating system were devised by Wirth 1983 (Programming in Modula-2)
  - `NEWPROCESS`, `TRANSFER` and `IOTRANSFER` (covered later on)

A tiny example of two simple processes in an operating system

```
void Process1 (void)
{
   while (TRUE) {
      WaitForACharacter();
      PutCharacterIntoBuffer();
   }
}

void Process2 (void)
{
   while (TRUE) {
      WaitForInterrupt();
      ServiceDevice();
   }
}
```

Primitives to manage context switching

- firstly let us look at a conventional program running in memory (single program running on a computer) box with .sw at (4.737,8.012) width 0.750 height 0.750 line from 4.862,8.887 to 4.862,8.762 line from 5.050,8.887 to 5.050,8.762 line from 5.237,8.887 to 5.237,8.762 line from 5.425,8.887 to 5.425,8.762 line from 4.862,8.012 to 4.862,7.888 line from 5.050,8.012 to 5.050,7.888 line from 5.237,8.012 to 5.237,7.888 line from 5.425,8.012 to 5.425,7.888 box with .sw at (1.988,7.013) width 2.000 height 0.750 box with .sw at (1.988,8.012) width 2.000 height 0.750 box with .sw at (1.988,9.012) width 2.000 height 0.750 "i486" at 4.862,8.546 ljust "eax" at 4.862,8.296 ljust "ebx" at 4.862,8.108 ljust "ecx" at 5.175,8.296 ljust "esp" at 5.175,8.108 ljust "STACK" at 2.425,7.296 ljust "DATA" at 2.425,8.296 ljust "CODE" at 2.425,9.296 ljust

Primitives to manage context switching

- four main components
  - code
  - data
  - stack
  - processor registers (volatiles)

Concurrency

- suppose we want to run two programs concurrently?
  - we could have two programs in memory. (Two stacks, code, data and two copies of a volatile environment)
  - on a single processor computer we can achieve apparent concurrency by running a fraction of the first program and then run a fraction of the second.
  - if we repeat this then apparent concurrency will be achieved
  - in operating systems multiple concurrent programs are often called *processes*

Concurrency

- what technical problems need to be solved so achieve apparent concurrency?
  - require a mechanism to switch from one process to another

- remember our computer has one processor but needs to run multiple processes
  - the information about a process is contained within the volatiles (or simply: processor registers)

Implementing concurrency

- we can switch from one process 1 to process 2 by:
  - copying the current volatiles from the processor into an area of memory dedicated to process 1
  - now copying some new volatiles from memory dedicated to process 2 into the processor registers dashwid = 0.038 line dotted <-> from 4.237,9.137 to 5.300,9.137 to 5.300,8.762 line dotted <-> from 4.237,6.638 to 5.362,6.638 to 5.362,7.325 dashwid = 0.050 box dashed with .sw at (1.488,6.013) width 3.000 height 2.000 box dashed with .sw at (1.488,8.262) width 3.000 height 2.000 box with .sw at (1.875,6.150) width 1.075 height 0.412 box with .sw at (1.875,6.688) width 1.075 height 0.412 box with .sw at (1.875,7.225) width 1.075 height 0.413 box with .sw at (1.875,9.463) width 1.075 height 0.412 box with .sw at (1.875,8.925) width 1.075 height 0.412 box with .sw at (1.875,8.387) width 1.075 height 0.413 dashwid = 0.038 box dotted with .sw at (3.487,6.263) width 0.750 height 0.750 box dotted with .sw at (3.487,8.762) width 0.750 height 0.750 box with .sw at (4.925,7.513) width 0.750 height 0.750 line from 5.050,8.387 to 5.050,8.262 line from 5.237,8.387 to 5.237,8.262 line from 5.425,8.387 to 5.425,8.262 line from 5.612,8.387 to 5.612,8.262 line from 5.050,7.513 to 5.050,7.388 line from 5.237,7.513 to 5.237,7.388 line from 5.425,7.513 to 5.425,7.388 line from 5.612,7.513 to 5.612,7.388 "Volatiles" at 3.612,9.608 ljust "Volatiles" at 3.550,7.108 ljust "Processor" at 4.987,8.546 ljust "Process 2" at 2.987,7.796 ljust "Process 1" at 2.925,10.046 ljust "STACK" at 2.100,6.308 ljust "DATA" at

2.100,6.858 ljust "CODE" at 2.100,7.408 ljust "CODE" at 2.100,9.646 ljust "DATA" at 2.100,9.096 ljust "STACK" at 2.100,8.546 ljust "i486" at 3.550,6.796 ljust "eax" at 3.550,6.546 ljust "ebx" at 3.550,6.358 ljust "ecx" at 3.862,6.546 ljust "esp" at 3.862,6.358 ljust "esp" at 3.862,8.858 ljust "ecx" at 3.862,9.046 ljust "ebx" at 3.550,8.858 ljust "eax" at 3.550,9.046 ljust "i486" at 3.550,9.296 ljust "i486" at 5.050,8.046 ljust "eax" at 5.050,7.796 ljust "ebx" at 5.050,7.608 ljust "ecx" at 5.362,7.796 ljust "esp" at 5.362,7.608 ljust

Implementing concurrency

- this operation is call a context switch (as the processors context is switched from process 1 to process 2)
  - by context switching we have a completely new set of register values inside the processor
  - so on the i486 we would change **all** the registers. Some of which include: `EAX, EBX, ECX, EDX, ESP` and `flags`
  - note that by changing the `ESP` register (stack pointer) we have effectively changed stack

Context switching primitives in GNU LuK

- the previous description of context switching is very low level
- in a high level language it is desirable to avoid the assembler language details as far as possible
  - `NEWPROCESS`
  - `TRANSFER`
  - `IOTRANSFER`
- **it is possible to build a microkernel which implements context switching and interrupt driven devices using these primitives without having to descend into assembly language**
  - these are the primitives as defined by Wirth in 1983

Context switching primitives in GNU LuK

- the primitives `NEWPROCESS`, `TRANSFER` and `IOTRANSFER` are concerned with copying ***Volatiles between process and processor***
- the procedure `TRANSFER` transfers control from one process to another process
- these primitives are *low level* primitives
  - they are normally wrapped up by higher level functions:
    - for example: `initProcess` uses `NEWPROCESS` which is similar to `new_thread` in Python

TRANSFER

- the C definition is:

- ```
  typedef void *PROCESS;

  extern void SYSTEM_TRANSFER (PROCESS *p1, PROCESS p2);
  ```

- and it performs the following action: dashwid = 0.038 box dotted with .sw at (3.487,6.263) width 0.750 height 0.750 box dotted with .sw at (1.738,6.263) width 0.750 height 0.750 line from 3.300,8.262 to 3.300,8.137 line from 3.112,8.262 to 3.112,8.137 line from 2.925,8.262 to 2.925,8.137 line from 2.737,8.262 to 2.737,8.137 line from 3.300,9.137 to 3.300,9.012 line from 3.112,9.137 to 3.112,9.012 line from 2.925,9.137 to 2.925,9.012 line from 2.737,9.137 to 2.737,9.012 box with .sw at (2.612,8.262) width 0.750 height 0.750 line <- from 3.362,8.700 to 3.409,8.700 to 3.454,8.699 to 3.497,8.699 to 3.539,8.698 to 3.580,8.696 to 3.618,8.695 to 3.656,8.693 to 3.692,8.690 to 3.726,8.688 to 3.759,8.685 to 3.791,8.681 to 3.822,8.678 to 3.852,8.673 to 3.880,8.669 to 3.907,8.664 to 3.933,8.659 to 3.983,8.647 to 4.028,8.633 to 4.070,8.618 to 4.109,8.601 to 4.144,8.582 to 4.178,8.561 to 4.208,8.538 to 4.237,8.512 line from 4.237,8.512 to 4.276,8.476 to 4.314,8.437 to 4.352,8.396 to 4.390,8.353 to 4.428,8.308 to 4.465,8.261 to 4.501,8.213 to 4.536,8.163 to 4.554,8.138 to 4.571,8.112 to 4.588,8.086 to 4.604,8.059 to 4.620,8.033 to 4.636,8.005 to 4.651,7.978 to 4.666,7.950 to 4.681,7.922 to 4.695,7.894 to
  ```

4.709,7.866 to 4.722,7.837 to 4.735,7.808 to 4.748,7.779 to 4.759,7.750 to 4.771,7.721 to 4.781,7.691 to 4.792,7.661 to 4.801,7.632 to 4.810,7.602 to 4.819,7.572 to 4.826,7.541 to 4.833,7.511 to 4.840,7.481 to 4.845,7.450 to 4.850,7.420 to 4.854,7.389 to 4.858,7.359 to 4.860,7.328 to 4.862,7.298 to 4.863,7.267 to 4.863,7.237 to 4.863,7.206 to 4.861,7.176 to 4.859,7.145 to 4.855,7.115 to 4.851,7.085 to 4.846,7.055 to 4.840,7.025 to 4.833,6.995 to 4.824,6.965 to 4.815,6.936 to 4.805,6.906 to 4.794,6.877 to 4.781,6.848 to 4.768,6.819 to 4.753,6.791 to 4.737,6.763 line from 4.737,6.763 to 4.707,6.719 to 4.671,6.684 to 4.626,6.658 to 4.601,6.647 to 4.573,6.639 to 4.542,6.633 to 4.509,6.628 to 4.472,6.626 to 4.432,6.625 to 4.389,6.625 to 4.342,6.628 to 4.318,6.630 to 4.292,6.632 to 4.265,6.634 to 4.237,6.638 line <- from 1.738,6.700 to 1.706,6.723 to 1.675,6.746 to 1.646,6.767 to 1.618,6.789 to 1.591,6.810 to 1.566,6.830 to 1.518,6.869 to 1.475,6.906 to 1.436,6.942 to 1.401,6.977 to 1.369,7.010 to 1.342,7.043 to 1.318,7.075 to 1.297,7.106 to 1.280,7.137 to 1.265,7.168 to 1.253,7.199 to 1.244,7.231 to 1.238,7.263 line from 1.238,7.263 to 1.232,7.302 to 1.228,7.341 to 1.226,7.382 to 1.226,7.424 to 1.228,7.466 to 1.231,7.509 to 1.236,7.552 to 1.243,7.596 to 1.251,7.639 to 1.261,7.683 to 1.272,7.728 to 1.285,7.772 to 1.299,7.816 to 1.314,7.860 to 1.330,7.903 to 1.348,7.946 to 1.366,7.989 to 1.386,8.031 to 1.407,8.073 to 1.428,8.114 to 1.451,8.154 to 1.474,8.193 to 1.498,8.231 to 1.522,8.268 to 1.548,8.304 to 1.573,8.338 to 1.600,8.371 to 1.627,8.403 to 1.654,8.433 to 1.682,8.461 to 1.709,8.488 to 1.738,8.512 line from 1.738,8.512 to 1.766,8.535 to 1.797,8.556 to 1.831,8.576 to 1.866,8.593 to 1.905,8.609 to 1.947,8.624 to 1.992,8.637 to 2.041,8.648 to 2.068,8.654 to 2.095,8.659 to 2.123,8.663 to 2.153,8.668 to 2.183,8.672 to 2.215,8.676 to 2.249,8.679 to 2.283,8.683 to 2.319,8.686 to 2.357,8.688 to 2.395,8.691 to 2.436,8.693 to 2.478,8.695 to 2.521,8.697 to 2.566,8.699 to 2.612,8.700 "2" at 4.237,8.296 ljust "1" at 1.800,8.296 ljust "Volatiles" at 3.550,7.108 ljust "Process 2" at 3.550,7.546 ljust "Process 1" at 1.863,7.546 ljust "i486" at 3.550,6.796 ljust "eax" at 3.550,6.546 ljust "ebx" at 3.550,6.358 ljust "ecx" at 3.862,6.546 ljust "esp" at 3.862,6.358 ljust "i486" at 1.800,6.796 ljust "eax" at 1.800,6.546 ljust "ebx" at 1.800,6.358 ljust "ecx" at 2.112,6.546 ljust "esp" at 2.112,6.358 ljust "Volatiles" at 1.863,7.108 ljust "esp" at 3.050,8.358 ljust "ecx" at 3.050,8.546 ljust "ebx" at 2.737,8.358 ljust "eax" at 2.737,8.546

ljust "i486" at 2.737,8.796 ljust "Processor" at 2.675,9.296 ljust

IOTRANSFER

```
extern void SYSTEM_IOTRANSFER (PROCESS *first,
                               PROCESS *second,
                               unsigned int interruptNo);
```

- the procedure `IOTRANSFER` allows process contexts to be changed when an interrupt occurs

- its function can be explained in two stages
  - firstly it transfers control from one process to another process (in exactly the same way as `TRANSFER`)
  - secondly when an interrupt occurs the processor is context switched back to the original process

- the implementation of `IOTRANSFER` involves interaction with the `FLIH`

NEWPROCESS

```
extern void SYSTEM_NEWPROCESS (void (*p)(void), void *a,
                               unsigned long n,
                               PROCESS *new);
```

- `p` is a pointer to a function.
  - this function will be turned into a process
  - `a` the start address of the new processes stack
  - `n` the size in bytes of the stack
  - `new` a variable of type `PROCESS` which will contain the volatiles of the new process

How is TRANSFER implemented?

- or how do we implement a context switch?
  - first we push all registers onto the stack
  - second we need to save the current running processes stack pointer into the running process control block
  - third we need to restore the next process stack pointer into the microprocessors stack pointer
  - fourth we pop all registers from the stack

How is TRANSFER implemented?

```
void SYSTEM_TRANSFER (PROCESS *p1, PROCESS p2)
{
   onOrOff toOldState;

   toOldState = turnInterrupts(Off);
   asm volatile ("pusha ; pushf");   /* push all registers */
   /* remember p1 is the address of a PROCESS */
   asm volatile ("movl %[p1], %%eax ; movl %%esp, (%%eax)"
      :: [p1] "rm" (p1));            /* p1 := top of stack  */
   asm volatile ("movl %[p2], %%eax ; movl %%eax, %%esp"
      :: [p2] "rm" (p2));            /* top of stack := p2 */
   asm volatile ("popf ; popa");     /* restore all registers */
   toOldState := turnInterrupts(toOldState);
}
```

- `asm volatile`
  - means inline an assembly instruction

How is TRANSFER implemented?

- the parameters `("movl %[p1], %%eax ; movl %%esp, (%%eax)" :: [p1] "rm" (p1));`

- means
  - move `p1` into register `%eax`
  - move `%esp` into the address pointed to by `%eax`
  - p1 is a variable which may be in a register or in memory
  - p1 is an input to the assembly instruction

Conclusion

- we have seen the structure of a `FLIH`
- we have seen how three primitives can be used to create processes, context switch between processes and react to interrupts
- we have seen how a context switch might be implemented

Further reading

- Abraham Silberschatz, Operating System Concepts
  - section 3.2.3 (Context Switch)
  - section 19.3.2.5 Exceptions and Interrupts

- newprocess, transfer and iotransfer ⟨`http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/operating/luk/system.h.html`⟩

- newprocess, transfer and iotransfer ⟨`https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/68683/eth-3135-01.pdf?sequence=1&isAllowed=y`⟩
  - pages 27, 28, 29