CS 350 Notes

Matthew Visser

Sep 29, 2011

1 Deadlocks

Use "no hold and wait" and "resource ordering" in OS/161.

1.1 Deadlock Detection and Recovery

Determine if there is a deadlock and terminate a thread. The thread you terminate doesn't matter, it's a choice that's up to the scheduler. Ideally choose one that makes sense with regard to resources, but there are trade-offs.

We can decide if the system is deadlocked by looking at a resource allocation graph. We want an algorithm that looks at these graphs and tries to make the same decisions about them. We can define state by

- D_i demand vector for thread T_i.
- A_i current allocation for thread T_i.
- U unallocated resources

We also have R for a scratch resource and f_i a boolean that determines if T_i can finish.

See the algorithm on slide 38 of `synch.pdf`.

2 C Programming Tips

2.1 Modular Code

The OS/161 source tree is separated into different modules with `.h` (headers with function prototypes and type declarations) and `.c` files (with function definitions and global variables.)

These modules will be using and calling each other. When one module uses another, all it cares about is the header file and what is defined there. For example, using `synch.h` with `#include "synch.h"`, will give you all of the functions in that header file.

2.2 Arrays and Pointers

Consider the code:

```
int a[5];
int x;
int *p;
```

The definition of `a` is an array that holds 5 integers on the stack. The variable `a` is actually just a pointer to the first address in the array, so `*a` is equivalent to `a[0]`. `*(a+1)` is equivalent to `a[1]`.

A pointer can be set tothe value of another pointer, such as `p = a`, or to the address of a variable such as `p = &x`. It can also be equal to an allocated array pointer by doing

```
p = (int *) malloc( 16 * sizeof(int));
```

In OS/161 we will use `kmalloc` instead of `malloc`. We will also use `kprintf` instead of `printf`.

2.3 The `const` Keyword

We can declare a variable that should never change like this

```
const int x = 5;
const int *x;
int * const x;
```

The first is a constant number. The second is a pointer to a constant integer. The third is a pointer that is constant, that points to an integer value. A trick to understand this is to read from right to left.

2.4 Other Notes

- Don't access pointers after they have been freed
- Don't return a pointer to a variable on the stack, *i.e.*, a variable in the argument list or declared in the function without `kmalloc`.

3 Processes

A **process** is an abstraction of a program in execution. It contains

- An address space
- one or more thread of execution
- Other resources such as files, sockets, attributes, *etc.*

A process is a user level program, not something in the kernel.

3.1 Multiprogramming

- Means having multiple processes at the same time
- Most modern computers do this
- All processes share the resources of the machine. The kernel needs to co-ordinate these resources.
- Processes *timeshare* the processor. This is controlled by the operating system.
- The OS ensures that processes are isolated from each other. Interprocess communication should be possible, but only when explicitly requested by the processes.

4 The Kernel

- The kernel is a program with code and data.
- The kernel runs in a privileged execution mode, while other programs don't. This is a special mode offered by the CPU.
- The hardware has to switch between privileged execution mode and user mode.
- What does privileged mode mean? What can we do with it?
  - control hardware: disable interrupts, halt the machine,...
  - protect and isolate itself (the kernel) from processes.
- privileges vary from platform.

5 System Calls

- Interface between processes and the kernel. They allow a process to tell the kernel they want a resource or to do something with hardware.
- The process can use this to do things like
  - Open a file: `opn`
  - create a pipe: `write`
  - Create another process: `fork`
  - increase address space: `brk`
- This is done by calling a special instruction (`syscall` in MIPS) This jumps to a specific address in the kernel. This address in MIPS is `0x80000080`.
- We don't want user code to jump in arbitrary addresses in the kernel code, but this is prevented by the fixed address.