6.033 Lecture 6 - 7
Operating Systems Part VI

Americo De Filippo

May 24, 2023

References

[1] Saltzer, Jerome H. and M. Frans Kaashoek. Principles of Computer System Design: An Introduction (2009): **Sections: 5.1 / .3**

1 Virtual Processors VP

The idea is to create a processor that runs the programs on his own process, indipendent of the others programs that are running. In order to have this, we are going to introduce the notion of a **thread**, which is a collection of instruction that the modules in running as well as the current state of the program PC and the stack pointer and the heap which is shared between threads.

1.1 How is gonna implement this?

The threads will run one at time, in this way we are going to switch between the threads while saving the progress of a threads in a slice of time.

1.2 Cooperative scheduling

In this approch each thread will tell at the processor 'im done executing' and the processor will switch to another module, the obvius problem here is that the thread will never stops and the processor will block on a single module.

1.3 Preemptive scheduling

The kernel in this case decide when stopping a thread and run another.

1.4 Yield() (routine for coopering scheduling)

Yield() is what the current thread calls when give up the processor, so yield() is going to save the state and schedule the next thread to run and dispach the next thread.

1.5 Thread scheduler

He is gonna decide which thread run next, is gonna have an array of all the threads in the system, an integer for the next thread index and id of the current executing thread. Look at the slide for an example of thread stack handling.

1.6 Pipes

The pipes allows the programs to communicate using the procedures from the file system call interface. This will communicate using the following two procedures SEND() RECEIVE(). Usually in UNIX we want to have this thing writing in a file, in this case we will use a bounded buffer so that many threads (also on different physical cores) can access to the buffer and send some messages or receive it.

1.7 Changing to Preemptive Scheduling

We have no explicit Yield() statements, the idea is that we have a special timer that clocks periodicly into the kernel, this timer is gonna be connected to what is called an **interrupt** (causes the processor to run a special piece of code). This timer is be gonna check by the microprocessor and if is true the microprocessor is gonna run a special instruction. The way that is gonna work is that when the kernel switch thread with the Yield(on the current thread) forcing it to stop.

2 Coopering in different address spaces

When we talk about process we mean an address space (virtual) + some collection of threads. The kernel provides routines to create and process and to destroy a process. The create() in gonna allocate the address space into the table (page-map) and is going to add the code into the address space, is gonna create a thread for the new process add to the thread table, setup PC etc... [1]

2.1 Hierarcy of threads

In this way we can make an entire tree of threads that doesnt have to know of the others threads in the same layer, but the ones that are blow the layer have to know which threads is running and have to handle the threads operations.

2.2 Cooperating Access between threads

let's say we are going to have global data structure that is shared between two threads, we want to handle the access on this global variable? So the thing that we want to do is to check for specifics charateristic on the data structure that the two threads have to do.

[1] look at the slides

2.3 Sequence coordination operators

Allow threads that are waiting for a condition to run. In order to introduce this we are going to add a new data type: eventcount, integer that indicates the number of time that something has occurred and we are going to introduce two new routine; Wait(eventcount, value) if (eventcount ¡= value), Notify(eventcount) notify wake up everybody who is waiting on the variable passed.