OS- Class 3

No pencil ☹

Project 1 end of the week

User threads vs kernel threads

Kernel\_ os is aware of processes and threads that they contain

User\_ os knows nothing about the existence of threads

P\_thread doesn’t care HOW its implemented

How do we schedule?

* assuming OS has a scheduler ( selects a ready thing and runs it )
* what can scheduler select? ( kernel-process AND thread ) ( user- process only )
* Greedy thread (infinite loop)
  + Kernel – it runs until hardware interrupt ( preemption timer )can select another thread in another process
  + User – can only select the other process (‘starves out’ other process in the same thread ) (each process can have its own threading algorithm) ( programmers fault?) threads should be cooperative
    - Still no solution to do anything about the greedy thread
* Cant guarantee that processes are good or bad – threads are user written! Entirely cooperative, it’s a programming mistake.
* YIELD WHEN PROGRAMMING A THREADING APPLICATION
* How does this work in kernel thread? Pthread\_yield in user threading. Bad in kernel because we have to CONTEXT SWITCH worse because more context switches ( don’t HAVE to yield, could implement as an empty function )
* Thread does something very bad. Blocking system call. ( can block an individual thread, instead of the entire process )

Blocking issues in user threading. Killed the idea of programming.

Non blocking system call – data, error, try again later.

Some sort of hybrid of the two.

Kernel threading for the most part…

User threads ( green threads )

Scheduling

Reasonable?

Multiprogramming?

Pseduoparallelism, don’t have n cores n processes

Juggle, we believe we have time to run other code, convince its true

Lots of cpu time , little io , most of the time computing – cpu bound

More time waiting then computing – io bound

Run other programs in the background hiding in the waiting stages

Running two io bound apps - > amazing

Running two cpu bound apps - > might actually be slower!

This is because we’re adding context switches for no real reason

Schedule because something about the processes has changed, the number of them or their states.

Scheduling…only 1 type of scheduler ( cpu scheduler ) selects amongst processes in ram ( which ready process should get time )

We can also schedule from memory, and admission.

Evict a process out of memory and write it to disk, with the hope we can later write it back, cant be a candidate if not in ram

‘good’ scheduling algorithm

define good

Metrics

fairness- comparable processes get comparable service

define fair

throughput – number of jobs completed per unit of time

Turnaround time – time from submission (ready) to completion

Avg turnaround time

Batch Scheduling – Non-interactive jobs that can be run ‘overnight’

First come first serve scheduling.

4 3 6 3 queue

4 3 6 3 execution

throughput – 4/ 16 -> ¼ job/unit

throughputs will be different with preemption, overhead, context switches and such

avg turnaround time.

P1 = 4-0 = 4

P2 = 7-0 = 7

P3 = 13-0 = 13

P4 = 16-0 = 16

= 40/ 4 = 10

shortest job first

3 3 4 6